Слободанка Ђенић
Јелена Митић ▪ Светлана Штрбац

Програмирање на језику "C" и основи програмирања на језику "C++"

Збирка примера и задатака
за лабораторијске вежбе из предмета
Програмски језици

Висока школа електротехнике и рачунарства
струковних студија
Београд, 2019.

Аутори: Слободанка Ђенић, Јелена Митић, Светлана Штрбац

Рецензенти: Ласло Краус, Слободан Обрадовић

Лектор: Анђелка Ковачевић

Корице: Јелена Лабус (MASSVision)

Издавач: Висока школа електротехнике и рачунарства струковних студија у Београду

Штампа: Развојно-истраживачки центар графичког инжењерства ТМФ, Београд

Тираж: 120

Издање: 11.

ISBN 978-86-7982-161-4

CIP - Каталогизација у публикацији - Народна библиотека Србије, Београд

004.42(075.8) (076)
004.432.2C(075.8) (076)
004.432.2C++(075.8) (076)

ЂЕНИЋ, Слободанка, 1965-
Програмирање на језику "C" и основи програмирања на језику "C++" : збирка примера и задатака за лабораторијске вежбе из предмета Програмски језици / Слободанка Ђенић, Јелена Митић, Светлана Штрбац. - 11. изд. - Београд : Висока школа електротехнике и рачунарства струковних студија, 2019 (Београд : Развојно-истраживачки центар графичког инжењерства ТМФ). – 259 стр. : илустр.; 24 cm

Тираж 120. - Библиографија: стр. 259.

ISBN 978-86-7982-161-4

1. Митић, Јелена, 1966- [аутор] 2. Штрбац, Светлана, 1972- [аутор]
a) Програмирање - Вежбе b) Програмски језик "C" - Задаци v) Програмски језик "C++" - Задаци
COBISS.SR-ID 279600908

Предговор

Збирка примера и задатака “Програмирање на језику *C* и основи програмирања на језику *C++*” намењена је студентима друге године Високе школе електротехнике и рачунарства струковних студија у Београду и свима који су савладали основе програмског језика *C* а желе да усавршe своје знање овог језика и науче основе програмског језика *C++*.

Дуги низ година програмски језици *C* и *C++* представљају најмоћније програмске језике јер су блиски хардверу, имају моћ и брзину асемблерског језика и задовољавају захтеве који се постављају пред више програмске језике.

Коришћењем језика *C* и *C++* може се написати било који програм, од најједноставније апликације, преко програма оперативних система *Unix*, *Linux* и свих *Windows* оперативних система као и великог броја програмских алата под овим оперативним системима, до програма за пројектовање најкомплекснијих електронских склопова.

Збирка обухвата три дела. Први део објашњава употребу функција, динамичке доделе меморије, структура и датотека у програмима на језику *C*. Други део збирке објашњава употребу свих ових компоненти у необјектно-оријентисано пројектованим програмима на језику *C++*, као и основе објектно-оријентисаног пројектовања програма на језику *C++.* Трећи део збирке садржи решене програмске задатке који се могу срести у пракси.

Примери првог дела збирке (првих шест вежби) објашњавају детаље рада са улазно / излазним токовима података и са функцијама улаза / излаза у програмима на језику *C*. У даљим примерима детаљно су објашњени примена показивача код функција и динамичка додела меморије, који омогућавају писање веома ефикасних програма на језику *C*. Ту су даље структуре, низови и листе структура, који се користе у програмима на језику *C* за ефикасно руковање подацима повезаним у логичке целине. Примери овог дела збирке садрже и рад са датотекама из програма на језику *C*.

Други део збирке (последње четири вежбе) садржи програме на језику *C++.* Синтакса језика *C* као и употреба елемената овог програмског језика могу се у великој мери применити и у програмима на језику *C++.* У овом делу збирке објашњене су неке нове могућности које се односе само на *C++* програме. Ту су нови механизми за улаз/излаз података, нове особине функција, динамичке доделе меморије и рада са датотекама, који су засновани на *C++* класама и објектима. У овом делу збирке има програма који користе само класе стандардних *C++* библиотека, као и програма који садрже пројектовање класа, креирање њихових објеката и коришћење основних објектно-оријентисаних концепата. С обзиром на то да објектно-оријентисани концепти нису домен само језика *C++*, овај део збирке може послужити као увод у објектно-оријентисано програмирање на свим језицима који подржавају ову методу пројектовања програма.

Трећи део збирке садржи осам програма на језику *C* који решавају различите програмске задатке из праксе. У овом делу дати су обимнији програми који су написани читљиво и уз пропратна објашњења, тако да се могу лако савладати на основу знања стеченог из претходних делова збирке, евентуално објектно-оријентисано испројектовати и прекодирати на језик C++.

Сваки пример у овој збирци има:

1. Изворни кôд (текст програма). Свака линија изворног кôда програма има на почетку свој редни број који није део кôда, ту је само ради лакше анализе програма;
2. Анализу програма (по линијама изворног кôда);
3. Слику са излазом програма. То је изглед *DOS* прозора (у питању су конзолне *DOS* апликације) по пуштеном програму у рад све до краја његовог извршавања;
4. Неке практичне савете и евентуалне грешке сличних програма.

У Београду, 2019. године Аутори

Садржај

Вежба 1.

Улаз / излаз података у програмима на језику *C*

Комплетан улаз / излаз података

Решавање проблема заосталих карактера улазних података

Комуникација са оперативним системом

Рад са подацима из командне линије

Примери

Изворни кôд програма **primer1_1**

```
1:      /*primer1_1.c*/
2:      /*Brisanje zaostalih karaktera sa standardnog ulaza*/
3:      /*(citanje manje od predvidjenog broja podataka pomocu f-je scanf())*/
4:
5:      #include <stdio.h>
6:      #define MAX1        5               /*dozvoljena duzina niza*/
7:      #define MAX2        80              /*broj korisnih karaktera u stringu*/
8:
9:      main()
10:     {       int i, niz[MAX1];
11:             char tekst[MAX2+1], bafer[MAX2+1];
12:
13:             /*Citanje manje od predvidjenog broja podataka*/
14:             printf("\nUnesite %d celih brojeva: ", MAX1*2);
15:             for(i=0; i<MAX1; i++)
16:                     scanf("%d", &niz[i]);
17:
18:             printf("Procitano samo %d brojeva: ", MAX1);
19:             for(i=0; i<MAX1; i++)
20:                     printf("%d ", niz[i]);
21:
22:             /*Nema brisanja zaostalih karaktera, greska pri daljem citanju*/
23:             printf("\n\nUnesite red teksta: ");
24:             gets(tekst);
25:             printf("\nProcitani red teksta: %s\n", tekst);
26:
27:             /*Citanje manje od predvidjenog broja podataka*/
28:             printf("\nUnesite %d celih brojeva: ", MAX1*2);
29:             for(i=0; i<MAX1; i++)
30:                     scanf("%d", &niz[i]);
31:
32:             /*Brisanje zaostalih karaktera na 1. nacin*/
33:             gets(bafer);
34:
35:             printf("Procitano samo %d brojeva: ", MAX1);
36:             for(i=0; i<MAX1; i++)
37:                     printf("%d ", niz[i]);
38:
39:             /*Nema greske pri daljem citanju*/
40:             printf("\n\nUnesite red teksta: ");
41:             gets(tekst);
42:             printf("Procitani red teksta: %s\n", tekst);
43:
```

```
44:             /*Citanje manje od predvidjenog broja podataka*/
45:             printf("\nUnesite %d celih brojeva: ", MAX1*2);
46:             for(i=0; i<MAX1; i++)
47:                     scanf("%d", &niz[i]);
48:
49:             /*Brisanje zaostalih karaktera na 2. nacin*/
50:             fflush(stdin);
51:
52:             printf("Procitano samo %d brojeva: ", MAX1);
53:             for(i=0; i<MAX1; i++)
54:                     printf("%d ", niz[i]);
55:
56:             /*Nema greske pri daljem citanju*/
57:             printf("\n\nUnesite red teksta: ");
58:             gets(tekst);
59:             printf("Procitani red teksta: %s\n\n", tekst);
60:
61:             return 0;
62:     }
```

Анализа програма **primer1_1**

Све функције улаза/излаза, које су досад коришћене у програмима, раде са улазно/излазним токовима података (*input/output streams*). Главни стандардни улазно/излазни токови *(stdin/stdout)*, за размену података са тастатуром и екраном, отварају се када се у програм укључи библиотека улаза/излаза *stdio.h*.

Стандардна функција *scanf()* омогућује програму да лако изврши форматирање улаза (један од начина да се учита са тастатуре више различитих типова података). Стандардне функције *getchar()* и *gets()* веома су корисне у програмима када је потребно читање са тастатуре једног по једног знака или низа знакова, респективно. Функција *scanf()* често се користи у комбинацији са овим функцијама, а онда се може десити да се не реализује читање онако како је предвиђено у програму.

Када се после позива функције *scanf()* позове функција *getchar()*, ова друга не ради како би требало због знака за прелаз у нови ред на крају, који је остао непрочитан од функције *scanf()* и који заварава функцију *getchar()* да је знак унет. Решење овог проблема је нађено у једном више позиву функције *getchar()*. Први пут позвана функција *getchar()* прочита знак за прелаз у нови ред, а други пут позвана иста функција чита знак који се чека са тастатуре.

Када се после позива функције *scanf()* позове функција *gets()* може се десити да не ради како би требало, ако се функцији *scanf()* зада више података него што је предвиђено, или реалан број уместо предвиђеног целог броја. У оваквим случајевима у главном стандардном улазном току заостају карактери, који могу правити проблеме током извршавања програма.

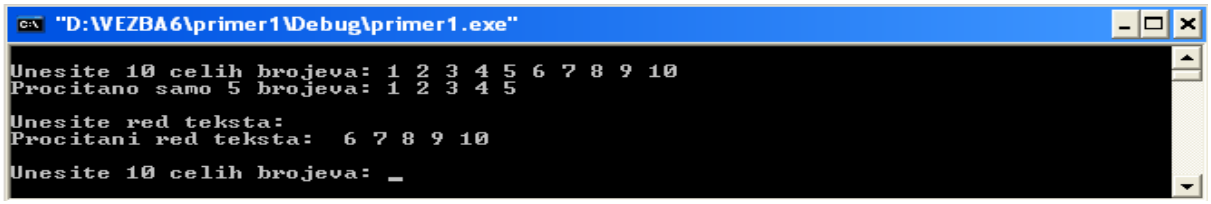
Некада је у програму неопходно комбиновати позив функције *scanf()* са позивима осталих функција улаза. Да не би било проблема због евентуално заосталих карактера у стандардном улазном току, после позива функције *scanf()* треба обезбедити брисање заосталих карактера са стандардног улаза на један од могућих начина (помоћу функције *fflush()* или *gets()).*

Програм овог примера има три дела. У првом (линије од 14. до 25.) није предвиђено брисање заосталих карактера из стандардног улазног тока, а намерно је направљена грешка код уноса података. Програм прво тражи да се учита низ од 10 целих бројева (слика 1.1 а)), а функција *scanf()* у једној *for* петљи (линије 15. и 16.) чита само 5 бројева. У програму је даље предвиђен приказ унетих бројева и унос реда текста. Програм неће чекати ред текста већ ће приказати на екрану заосталих 5 целих бројева (слика 1.1 б)).

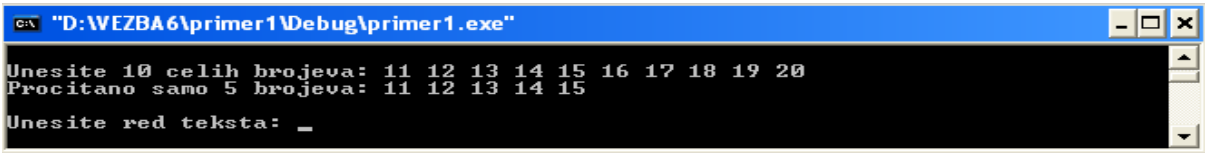
У другом делу програма (линије од 28. до 42.) предвиђено је брисање заосталих карактера из стандардног улазног тока помоћу позива стандардне функције *gets().* Програм прво тражи да се учита низ од 10 целих бројева (слика 1.1 б)), а функција *scanf()* у једној *for* петљи (линије 29. и 30.) чита само 5 целих бројева. Функција *gets()* прочита све заостале карактере унете преко тастатуре до краја реда (*bafer*). Из програма је даље предвиђен приказ на екрану учитаних бројева и порука да се унесе ред текста (слика 1.1 в)). После читања са тастатуре (линија 41.) програм приказује на екрану задати ред (слика 1.1 г)).

У трећем делу програма (линије од 45. до 59.) предвиђено је брисање заосталих карактера из стандардног улазног тока, помоћу позива стандардне функције *fflush()*, из библиотеке *stdio.h.* Програм прво тражи да се учита низ од 10 целих бројева (слика 1.1 г)), а функција *scanf()* у једној *for* петљи (линије 46. и 47.) чита само 5 целих бројева. Функција *fflush()* када јој се преда као аргумент име главног стандардног улазног тока *(stdin)* прочита све заостале карактере овог улазног тока. Из програма је даље предвиђен приказ учитаних бројева на екрану и порука да се унесе ред текста (слика 1.1 д)). После читања са тастатуре (линија 58.) програм приказује на екрану ред текста (слика 1.1 ђ)).

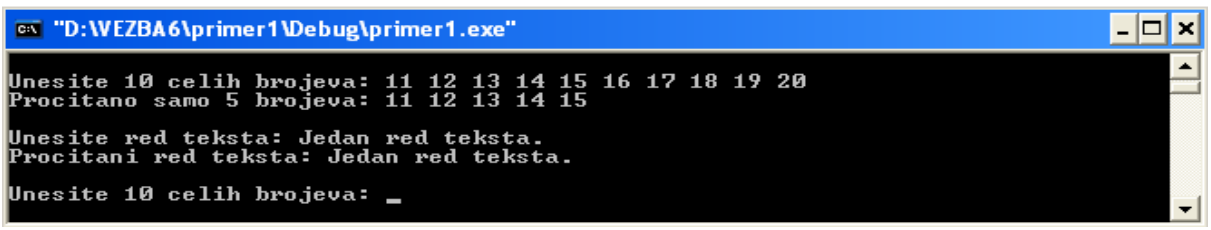
Слика 1.1 а) Излаз програма **primer1_1** (први део)



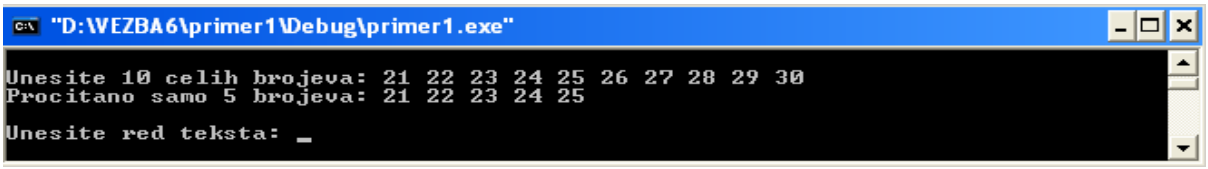
Слика 1.1 б) Излаз програма **primer1_1** (други део)



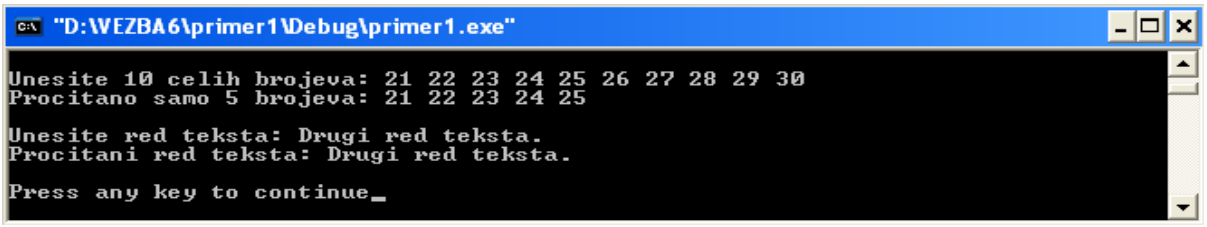
Слика 1.1 в) Излаз програма **primer1_1** (трећи део)



Слика 1.1 г) Излаз програма **primer1_1** (четврти део)



Слика 1.1 д) Излаз програма **primer1_1** (пети део)



Слика 1.1 ђ) Излаз програма **primer1_1** (шести део)

Изворни кôд програма **primer1_2**

```
1:      /*primer1_2.c*/
2:      /*Prikaz na ekranu reci, zadatih u komandnoj liniji*/
3:
4:      #include <stdio.h>
5:      #include <stdlib.h>
6:
7:      main(int argc, char *argv[])
8:      {       int i;
9:
10:             if(argc < 2)
11:             {       fprintf(stderr, "Za pokretanje programa potrebno je uneti\n");
12:                     fprintf(stderr, "reci u komandnoj liniji, posle imena programa\n");
13:                     exit(1);
14:             }
15:
16:             for(i=0; i<argc; i++)
17:                     printf("\nString na koji pokazuje argv[%d] je %s", i, argv[i]);
18:             printf("\n");
19:             return 0;
20:     }
```

Анализа програма **primer1_2**

Свака конзолна апликација може се покренути из командне линије на разне начине у зависности од тога како је написан њен изворни кôд: уносом само имена програма, уносом имена програма праћеног разним опцијама, уносом имена програма праћеног бројевима... .

Програм у овом примеру написан је да ради са следећом врстом позива из командне линије: име програма и произвољан број речи после имена програма.

Главна функција је по свему специфична и различита од осталих функција у програмима на језику *C*. У досадашњим програмима ове збирке

могла се видети једна верзија главне функције, са празном листом аргумената. Поред тога што може вратити статус оперативном систему (програм извршен до краја или превремено завршен, са грешком или без грешке), главна функција може користити аргументе од оперативног система.

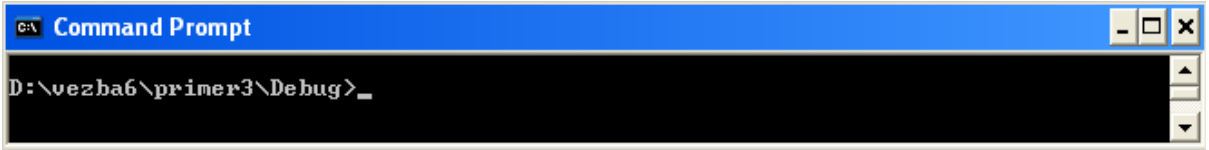
Главна функција у овом програму има два аргумента и то су подаци које она добија од оперативног система под којим се позива. Први аргумент *argc* је целобројна вредност, која садржи број података који се уносе у командној линији (узима у обзир и име програма). Други аргумент (*char *argv[]*) је показивач на низ показивача на податке из командне линије. Први показивач *argv[0]* је показивач на низ знакова који садржи име програма, други показивач *argv[1]* је показивач на низ знакова који је унет први после имена програма... .

Главна функција овог програма ради по позиву из командне линије. То значи да захтева отварање командног промпта (слика 1.2 а)). Наредба *if* у линији 10. проверава да ли је вредност аргумента *argc* мања од 2. Ако је то тачно значи да је у командној линији унето само име програма и у том случају програм пријављује на екрану да је била грешка и прекида са радом.

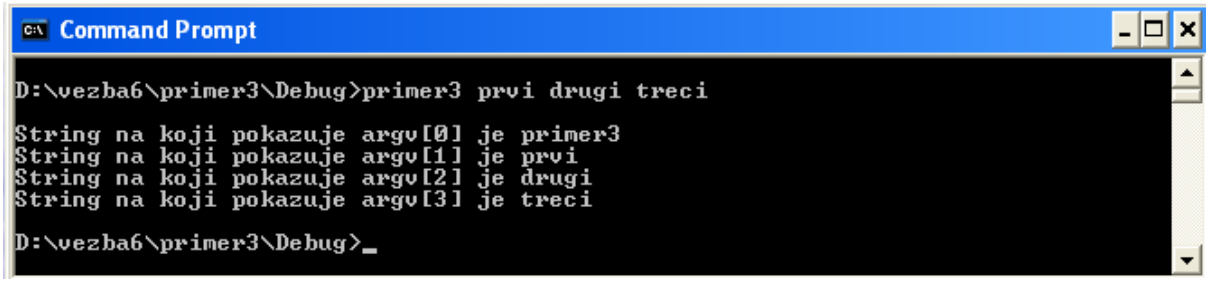
За приказ порука о грешкама, по први пут у примерима ове збирке, појављује се позив функције *fprintf()* (линије 11. и 12.). По укључењу стандардне библиотеке *stdio.h* аутоматски се отварају три стандардна тока: *stdin, stdout* и *stderr.* За прва два наведена тока важи да су баферовани и због тога је могуће у командној линији њихово преусмеравање (у програму се предвиди да се подаци шаљу на екран, али постоји могућност да се при пуштању програма каже да се уместо екрана као излазни уређај користи датотека). За поруке о евентуалним грешкама не би било добро да се, уместо на екран, пошаљу у неку датотеку и због тога се за овакве поруке користи стандардни ток за грешке (*stderr*) који се не може преусмерити. Када се у позиву функције *fprintf()* на месту првог аргумента напише име овог тока (*stderr*), а на месту другог аргумента показивач на низ карактера са поруком или сама порука написана између знака навода, та порука ће сигурно бити послата на екран.

Превремени прекид извршавања програма (линија 13.), у случају недовољног броја података у командној линији, остварује се помоћу стандардне функције *exit()* избиблиотеке *stdlib.h*.

Ако је у командној линији поред имена програма унето још података програм неће бити прекинут, већ ће у једној *for* петљи (линије 16. и 17.) приказати на екрану вредности ових података (чита их као знаковне низове и тако их и приказује на екрану). У формату *%s* функција *printf()* приказује на екрану садржаје низова каракатера чије адресе чувају показивачи: *argv[0], argv[1],.., argv[argc-1].* На слици 1.2 б) може се видети да је садржај на који показује *argv[0]* име програма (*primer3*) и да су садржаји на које показују *argv[1], argv[2]* и *argv[3]:* речи које су унете после имена програма (*prvi, drugi* и *treci*).



Слика 1.2 а) Излаз програма **primer1_2** (први део)



Слика 1.2 б) Излаз програма **primer1_2** (други део)

Изворни кôд програма **primer1_3**

```
1:      /*primer1_3.c*/
2:      /*Prikaz na ekranu vrednosti celog broja, zadatog u komandnoj liniji,*/
3:      /*u oktalnom, heksadecimalnom i binarnom obliku*/
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #define MAX 50                  /*dozvoljena duzina niza binarnih cifara*/
8:
9:      main(int argc, char *argv[])
10:     {       int i, n = 0, broj, bin[MAX];
11:
12:             if(argc < 2)
13:             {       fprintf(stderr, "Za pokretanje programa potrebno je uneti ceo\n");
14:                     fprintf(stderr, "broj u komandnoj liniji, posle imena programa\n");
15:
16:                     exit(1);
17:             }
18:
19:             broj = atoi(argv[1]);
20:
21:             printf("\nOktalna vrednost zadatog broja je:\t\t%o", broj);
22:             printf("\nHeksadecimalna vrednost zadatog broja je:\t%x", broj);
23:
24:             while(broj != 0)
25:             {       bin[n] = broj % 2;
26:                     broj /=2;
27:                     n++;
28:             }
29:
30:             printf("\nBinarna vrednost zadatog broja je:\t\t");
31:             for(i=n-1;i>=0; i--)
32:                     printf("%d", bin[i]);
33:             printf("\n");
34:
35:             return 0;
36:     }
```

Анализа програма **primer1_3**

Програм у овом примеру написан је да чита из командне линије: име програма и један цео број, унет после имена програма да би задати број био приказан на екрану у хексадецималном и бинарном облику.

Ако се у командној линији зада само име програма програм пријављује на екрану грешку и прекида се његово извршавање (линије од 12. до 17.).

У супротном, помоћу стандардне функције *atoi(),* из податка задатог после имена програма (на који показује *argv[1])* програм извлачи цео број и смешта га у променљиву *broj* (линија 19.).

Приказ на екрану задатог броја у октално и у хексадецималном облику остварује се помоћу одговарајућих формата *(%o* и *%x)*, а за бинарни облик користи се следећи алгоритам: све док садржај променљиве *broj* није једнак 0 елементу низа *bin[i]* додељује се остатак дељења садржаја променљиве *broj* бројем 2, дели се садржај променљиве *broj* бројем 2 и инкрементира индекс за следећи елемент у низу *bin* (линије од 24. до 28.).

Да би се добио бинарни облик броја програм на екрану приказује вредности елемената низа *bin*, од последњег елемента до првог.

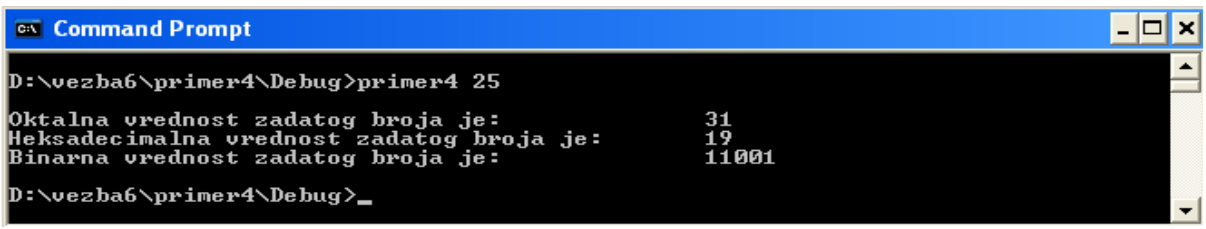
Главна функција програма захтева отварање командног промпта (слика 1.3 а)), унос имена програма и целог броја после имена програма. У конкретном примеру (слика 1.3 б)), за задати број 25 и *n=0*, добија се у *while* петљи:

broj = 25 => broj != 0 => bin[0] = 25%2 = 1, broj = 25/2 = 12, n = 1,
broj = 12 => broj != 0 => bin[1] = 12%2 = 0, broj = 12/2 = 6, n = 2,
broj = 6 => broj != 0 => bin[2] = 6%2 = 0, broj = 6/2 = 3, n = 3,
broj = 3 => broj != 0 => bin[3] = 3%2 = 1, broj = 3/2 = 1, n = 4,
broj = 1 => broj != 0 => bin[4] = 1%2 = 1, broj = 1/2 = 0, n = 5,
broj = 0.

Добијене су следеће вредности битова задатог броја:

bin[4]	*bin[3]*	*bin[2]*	*bin[1]*	*bin[0]*
1	*1*	*0*	*0*	*1*

Слика 1.3 а) Излаз програма **primer1_3** (први део)



Слика 1.3 б) Излаз програма **primer1_3** (други део)

Изворни кôд програма **primer1_4**

```
1:      /*primer1_4.c*/
2:      /*Izracunavanje sume brojeva, zadatih redom u komandnoj liniji, posle*/
3:      /*imena programa i opcije: i (za cele brojeve) ili f (za realne brojeve)*/
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #include <string.h>
8:
9:      typedef enum
10:             {NEMA_ARGUMENATA,NEMA_ZNAKA,NEMA_OPCIJE}
11:             Greska;
12:
13:     char *poruke_o_greskama[] = {
14:                     "Za pokretanje programa potrebno je uneti\n"
15:                     "vise reci u komandnoj liniji, posle imena programa!\n",
16:
17:                     "Za pokretanje programa potrebno je uneti\n"
18:                     "znak - i opciju u komandnoj liniji, posle imena programa!\n",
19:
20:                     "Nije uneta dozvoljena opcija\n"
21:                     "u komandnoj liniji, posle imena programa!\n"                   };
22:
23:     void greska(Greska status);
24:
25:     main(int argc, char *argv[])
26:     {       int i, s1 = 0;
27:             double s2 = 0;
28:
29:             if(argc < 2)
30:                     greska(NEMA_ARGUMENATA);
31:
32:             if(*argv[1] != '-')
33:                     greska(NEMA_ZNAKA);
34:
35:             switch(*(argv[1] + 1))
36:             {       case 'i':
37:                        for(i=2; i<argc; i++)
38:                             s1 += atoi(argv[i]);
39:                        printf("\nSuma celih brojeva iz komandne linije je %d\n", s1);
40:                        break;
41:                     case 'f':
42:                        for(i=2; i<argc; i++)
43:                             s2 += atof(argv[i]);
44:                        printf("\nSuma realnih brojeva iz komandne linije je %f\n", s2);
45:                        break;
46:                     default:
47:                        greska(NEMA_OPCIJE);
```

```
48:                         break;
49:             }
50:             return 0;
51:     }
52:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
53:     void greska(Greska status)
54:     {       fprintf(stderr, poruke_o_greskama[status]);
55:             exit(1);
56:     }
```

Анализа програма **primer1_4**

Програм у овом примеру написан је да ради са следећом врстом позива из командне линије:

- име програма,
- знак хоризонтална црта и опција *i* или *f* (без размака између знака хоризонтална црта и опције),
- низ целих бројева типа *int* или низ реалних бројева типа *float*, у зависности од претходно унете опције (*i* или *f).*

Ако се у командној линији зада само име програма, програм пријављује на екрану једну поруку и прекида се његово извршавање (линије 29. и 30.).

Наредба *if* (линија 32.) проверава да ли је први знак првог податка, унетог после имена програма *(*argv[1]*), хоризонтална црта. Ако то није случај програм пријављује на екрану другу поруку и прекида се његово извршавање (линија 33.). У супротном, програм се даље извршава.

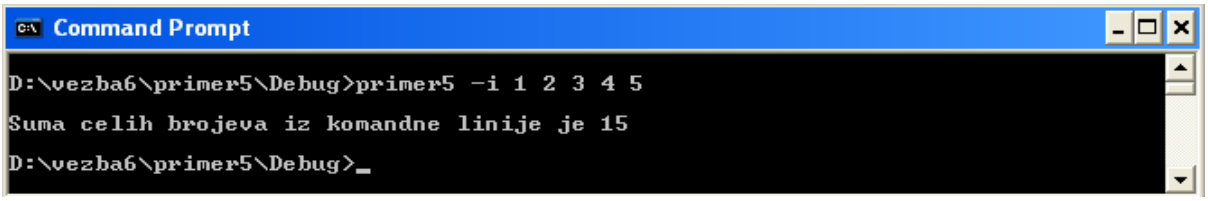
У зависности од вредности другог знака првог податка, унетог после имена програма *(*(argv[1] + 1)),* реализује се један од случајева *switch* наредбе (линије од 35. до 49.):

- ако је задата опција *i*, претходно иницијализованој суми *s1* додају се цели бројеви из свих података, унетих после ове опције:
  atoi(argv[2]), atoi(argv[3]), .., atoi(argv[argc-1]),
- ако је задата опција *f*, претходно иницијализованој суми *s2* додају се реални бројеви из свих података, унетих после ове опције:
  atof(argv[2]), atof(argv[3]), .., atof(argv[argc-1]).

Ако није био ни један од ова два случаја приказује се на екрану одговарајућа порука (линија 47.).

Главна функција програма захтева отварање командног промпта (слика 1.4 а)), унос имена програма, опције *-i* или *-f* после имена програма, као и низа целих или реалних бројева, чију суму израчунава (слика 1.4 б)).

Слика 1.4 а) Излаз програма **primer1_4** (први део)



Слика 1.4 б) Излаз програма **primer1_4** (други део)

Практични савети

- ✓ Најбезбедније је у програму не користити функцију *scanf()* у комбинацији са осталим функцијама улаза.
- ✓ Ако је неопходно у програму комбиновати позив функције *scanf()* са позивима осталих функција улаза, после позива функције *scanf()* треба обезбедити брисање заосталих карактера са стандардног улаза на један од могућих начина (помоћу функције *fflush()* или *gets()).*

Евентуалне грешке

- ➢ После позива функције *scanf()* за унос више различитих типова података, функција *getchar()* не ради како би требало због знака за прелаз у нови ред на крају који је остао непрочитан од функције *scanf()* и који заварава функцију *getchar()* да је знак унет.
- ➢ После позива функције *scanf()* за унос више различитих типова података, може се десити да функција *gets()* не ради како би требало, ако се функцији *scanf()* зада више података него што је предвиђено.
- ➢ Ако је у програму предвиђено коришћење података из командне линије, не треба заборавити да се као један од података из командне линије рачуна сам назив програма који се извршава.
- ➢ Ако је у програму предвиђено коришћење података из командне линије, на самом почетку програма неопходно је предвидети и проверу да ли је у командној линији задат тражени број података после назива програма.

Задаци за припрему вежбе 1. у лабораторији

Задатак 1.

Написати програм на језику *C*, чија је главна функција написана тако да програм прихвата аргументе из командне линије и приказује на екрану све речи из командне линије (реч је низ знакова између два празна знака), сваку у посебном реду.

Задаци за вежбу 1. у лабораторији

Урадити следеће програме са главном функцијом која прихвата аргументе из командне линије.

Задатак 1.

Написати програм на језику *C* који

а) из података задатих у командној линији прихвата низ речи (најмање једну, највише 10 њих) и уписује их у низ стрингова,
б) сортира учитане речи по абецедном реду и тако сортирани низ речи приказује на екрану.

Задатак 2.

Написати програм на језику *C* који:

а) из првог податка задатог у командној линији, после назива програма, треба да прихвати опцију облика –cpy или –cat, а из осталих података у командној линији n (n≤5) речи,
б) ако опција не почиње знаком црта на средини (-) завршава се извршавање програма,
в) ако је опција –cpy, сваку учитану реч уписује у посебан стринг, ако је опција –cat све учитане речи надовезује једну на крај друге у резултујућем стрингу и у сваком случају приказује на екрану садржај свих формираних стрингова.

Задатак 3.

Написати програм на језику *C* који из података задатих у командној линији прихвата целобројне вредности задатих углова и приказује на екрану у табели: синус и косинус сваког задатог угла.

Задатак 4.

Написати програм на језику *C* који из података задатих у командној линији прихвата низ реалних бројева и приказује на екрану суму и средњу вредност задатог низа.

Вежба 2.

Функције и показивачи у програмима на језику *C*

Функције које користе показиваче као аргументе

Функције које враћају вредности показивача

Показивачи на функције

Правила досега променљивих

Примери

Изворни кôд програма **primer2_1**

```
1:    /*primer2_1.c*/
2:    /*Stampanje na ekranu zadatog reda teksta u okviru zadatih dimenzija*/
3:
4:    #include <stdio.h>
5:    #include <string.h>
6:    #define HLIN        '-'     /*znak horizontalna crta okvira*/
7:    #define VLIN        '|'     /*znak vertikalna crta okvira*/
8:    #define UGAO        '+'     /*znak za ugao okvira*/
9:    #define PRAZNO      ' '     /*prazan znak*/
10:   #define MAX         70      /*dozvoljena duzina teksta*/
11:   #define RAZMAK      3       /*dozvoljeni razmak izmedju okvira i teksta*/
12:
13:   void hokvir(int l);
14:   void pise_tekst(char bafer[]);      /*Funkcija sa argumentom-adresom niza*/
15:   void znakovi(int n, char c);
16:
17:   main()
18:   {       int duzina;
19:           char tekst[MAX+1];
20:
21:           printf("\nUnesite red teksta:\n"); gets(tekst);
22:           duzina = strlen(tekst) +2*RAZMAK;
23:
24:           hokvir(duzina);             /*Prikaz gornjeg dela okvira*/
25:           pise_tekst(tekst);          /*Prikaz teksta i bocnih delova okvira*/
26:           hokvir(duzina);             /*Prikaz donjeg dela okvira*/
27:
28:           return 0;
29:   }
30:   /*Funkcija koja prikazuje na ekranu horizontalni deo okvira*/
31:   void hokvir(int l)
32:   {       putchar(UGAO);
33:           znakovi(l, HLIN);
34:           putchar(UGAO);
35:           putchar('\n');
36:   }
37:   /*Funkcija koja prikazuje bocne delove okvira, */
38:   /*kao i zadati tekst izmedju bocnih delova okvira*/
39:   void pise_tekst(char bafer[])
40:   {       putchar(VLIN);
41:           znakovi(RAZMAK, PRAZNO);
42:           printf("%s", bafer);
43:           znakovi(RAZMAK, PRAZNO);
```

```
44:             putchar(VLIN);
45:             putchar('\n');
46:     }
47:     /*Funkcija koja prikazuje zadati znak zadati broj puta*/
48:     void znakovi(int n, char c)
49:     {       while(n > 0)
50:             {       n--;
51:                     putchar(c);
52:             }
53:     }
```

Анализа програма **primer2_1**

Програм у овом примеру црта на екрану оквир и исписује задати текст унутар оквира. Поједини делови главног задатка програма (приказ хоризонталног дела, бочних делова оквира и текста између) издвојени су у посебне функције.

Функције *hokvir()* и *znakovi()* искоришћене су из поменутог примера без измена и овде неће бити анализиране.

У линији 14. написана је декларација функције *pise_tekst()* која показује да ће функција користити један аргумент и неће враћати вредност остатку програма (већ ће само цртати на екрану бочне вертикалне делове оквира и исписивати текст између ових бочних делова).

У овом случају аргумент није променљива основног типа већ показивач на променљиву основног типа *char* (адреса променљиве типа *char*). Ово је позив функције по референци (предаје се адреса променљиве).

Аргумент који је показивач може се написати у декларацији функције *pise_tekst()* на један од следећих начина:

1. *pise_tekst(char *bafer)* - декларација каже да је аргумент показивач на било коју променљиву типа *char* (може бити и показивач на почетак низа типа *char*),
2. *pise_tekst(char bafer[])* - декларација каже да је аргумент показивач на почетак низа типа *char.*

У линијама од 39. до 46. написана је дефиниција функције *pise_tekst()*, која:

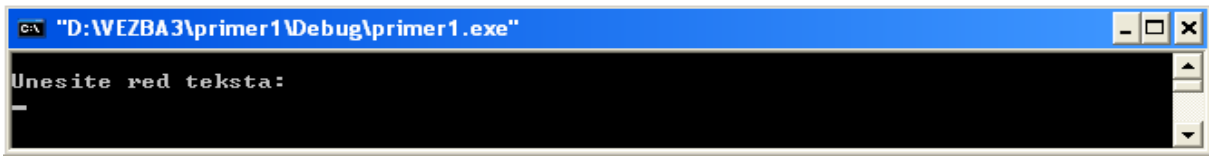
- приказује један знак вертикална црта за први вертикални део оквира (линија 40.),
- позива функцију *znakovi()* да одвоји три празна знака пре текста (линија 41.),
- исписује текст (линија 42.) који чува низ карактера *bafer* (аргумент функције је константна адреса низа *bafer,* а то је само име низа),
- позива још једном функцију *znakovi()* да одвоји три празна знака после текста (линија 43.),
- приказује један знак вертикална црта за други вертикални део оквира (линија 44.) и одрађује прелаз у нови ред (линија 45.).

Програм у главној функцији чита текст са тастатуре (слика 2.1 а)) и смешта га у низ карактера *tekst* (линија 21.), а онда дефинише дужину оквира за потребну дужину текста и потребне размаке између текста и оквира (линија 22.). Следе позиви одговарајућих функција:

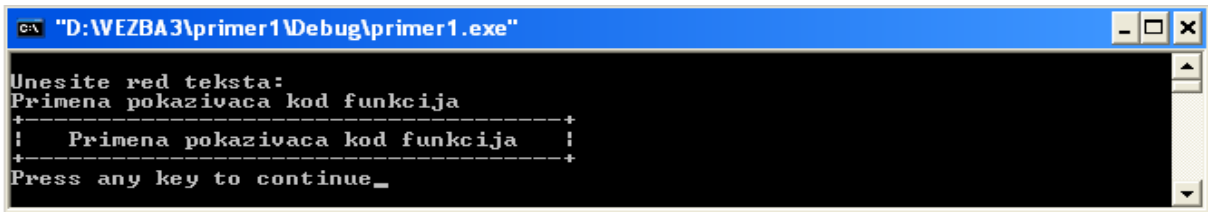
- позива једном функцију *hokvir(),* која црта горњи хоризонтални део оквира за стварну вредност дужине оквира (линија 24.),

- позива функцију *pise_tekst()* која црта бочне делове оквира и исписује текст који садржи низ карактера чији је показивач добила у аргументу. У позиву функције, као што се може видети у линији 25., на месту стварног аргумента пише се само име низа карактера *tekst*.
- позива још једном функцију *hokvir(),* која црта доњи хоризонтални део оквира, за стварну вредност дужине оквира (линија 26.).

Излаз програма може се видети на слици 2.1 б).



Слика 2.1 а) Излаз програма **primer2_1** (први део)



Слика 2.1 б) Излаз програма **primer2_1** (други део)

Изворни кôд програма **primer2_2**

```
1:      /*primer2_2.c*/
2:      /*Ciklicko pomeranje zadatog niza,za zadati pomeraj,u zadatom smeru*/
3:
4:      #include <stdio.h>
5:      #define MAX     10          /*dozvoljena duzina niza*/
6:
7:      void cita_niz(int x[], int n);
8:      void stampa_niz(int x[], int n);
9:      void levo_za_1(int x[], int n);
10:     void desno_za_1(int x[], int n);
11:
12:     main()
13:     {       int i, m, n, smer, greska = 0, niz[MAX];
14:
15:             do
16:             {       printf("\nBroj elemenata niza (n<=%d): ", MAX);
17:                     scanf("%d", &n);
18:             }while(n<1 || n>MAX);
19:
20:             cita_niz(niz, n);
21:
22:             printf("\nBroj ciklickih pomeraja elemenata niza: ");
23:             scanf("%d", &m); getchar();
24:             printf("\nSmer pomeranja (L/D): ");
25:             smer = getchar();
```

```
26:
27:             if(smer=='L' || smer=='l')
28:                     for(i=0; i<m; i++)
29:                             levo_za_1(niz,n);
30:             else if(smer=='D' || smer=='d')
31:                     for(i=0; i<m; i++)
32:                             desno_za_1(niz,n);
33:             else
34:             {       printf("\nNepoznata oznaka smera!\n");
35:                     greska = 1;
36:             }
37:
38:             if(!greska)
39:                     stampa_niz(niz, n);
40:
41:             return 0;
42:     }
43:     /*Funkcija koja cita sa tastature elemente niza*/
44:     void cita_niz(int x[], int n)
45:     {       int i;
46:
47:             printf("\nZadati celobrojni niz: ");
48:             for(i=0; i<n; i++)
49:                     scanf("%d", &x[i]);
50:     }
51:     /*Funkcija koja prikazuje na ekranu elemente niza*/
52:     void stampa_niz(int x[], int n)
53:     {       int i;
54:
55:             printf("\nNovi niz: ");
56:             for(i=0; i<n; i++)
57:                     printf("%d ", x[i]);
58:             printf("\n\n");
59:     }
60:     /*Funkcija koja pomera elemente niza za jedno mesto ulevo*/
61:     void levo_za_1(int x[], int n)
62:     {       int i, pom = x[0];
63:
64:             for(i=0; i<n-1; i++)
65:                     x[i] = x[i+1];
66:             x[n-1] = pom;
67:     }
68:     /*Funkcija koja pomera elemente niza za jedno mesto udesno*/
69:     void desno_za_1(int x[], int n)
70:     {       int i, pom = x[n-1];
71:
72:             for(i=n-1; i>0; i--)
73:                     x[i] = x[i-1];
74:             x[0] = pom;
75:     }
```

Анализа програма **primer2_2**

Програм у овом примеру циклички помера елементе задатог низа целих бројева. Оригинални низ, смер и број места за померање задају се преко тастатуре. У посебне функције издвојени су следећи делови програма: читање низа са тастатуре, померај за једно место улево, померај за једно место удесно и приказ резултујућег низа на екрану.

У линијама од 7. до 10. написане су декларације функција потребних програму. Из ових декларација може се видети да ће свака од четири функције користити два аргумента, показивач на низ типа *int* (*int x[]*) и дужину низа типа *int* (*int n*) и неће враћати вредност остатку програма. Предвиђене су функције:

- функција *cita_niz()* чита са тастатуре бројеве и смешта их у резервисани низ у меморији,
- функција *stampa_niz()* приказује на екрану садржај низа,
- функција levo_za_1() помера за једно место улево елементе низа,
- функција desno_za_1() помера за једно место удесно елементе низа.

Код свих поменутих функција један од аргумената је показивач на променљиву основног типа *int* (адреса првог елемента низа). За разлику од предаје аргумената по вредности (када се предају копије садржаја променљивих и функција не може мењати оригиналне садржаје променљивих), овде је заступљена предаја функцији аргумената по референци (предаје се адреса променљиве, тако да функција може изменити првобитни садржај променљиве чију адресу има).

Да би функција користила и евентуално мењала садржаје елемената низа, декларисаног ван те функције, поред адресе низа потребна је и дужина низа да би био могућ приступ свим елементима низа.

У линијама од 44. до 50. написана је дефиниција функције *cita_niz()*. Аргумент функције, који је показивач на низ, написан је помоћу заграда (*int x[])* што значи да се користи директан (индексни) приступ елементима низа: функција чита бројеве са тастатуре и смешта их од адреса свих елемената низа редом *(&x[i],i=0,..,n-1).* Могућ је и други начин приступа: ако се аргумент показивач на низ напише помоћу оператора индиректног адресирања (*int ∗x),* онда се користи индиректан (адресни) приступ елементима низа: за писање адреса елемената низа од којих функција смешта учитане вредности, користи се адресна аритметика *(x+i,i=0,..,n-1).* Без обзира који начин користи функција смешта све бројеве са тастатуре од адреса елемената низа, јер је у аргументима добила све потребне информације.

У линијама од 52. до 59. написана је дефиниција функције *stampa_niz()*, која приказује на екрану садржаје елемената низа *(x[i],i=0,..,n-1).*

У линијама од 61. до 67. написана је дефиниција функције *levo_za_1()* . Функција добија све потребне информације (адресу и дужину низа), тако да може одрадити померај за једно место улево (по већ познатом алгоритму), у оригиналном низу у оперативној меморији.

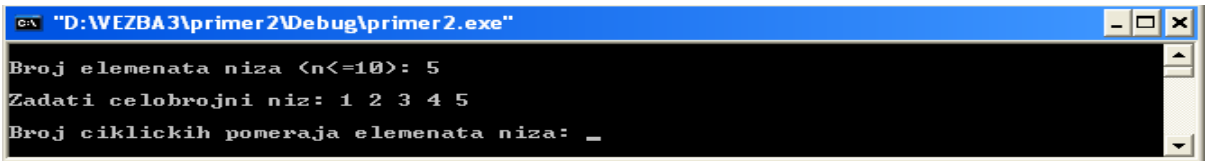
У линијама од 69. до 75. написана је дефиниција функције *desno_za_1(),* која ради један померај удесно (по већ познатом алгоритму), у оригиналном низу у оперативној меморији.

Програм из главне функције тражи са тастатуре број елемената низа (слика 2.2 а)), позива функцију *cita_niz()* (слика 2.2 б)), а онда тражи број помераја (слика 2.2 в)) и смер померања (слика 2.2 г)). Ако је задата једна од

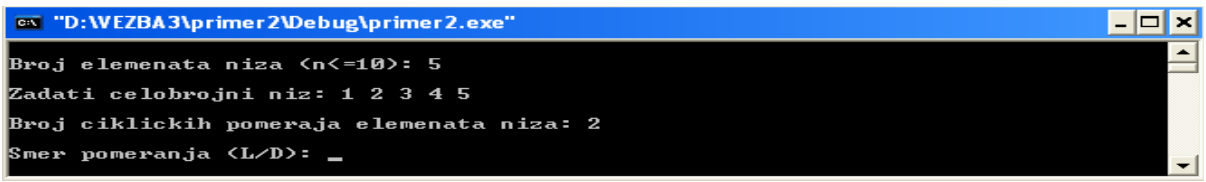
тражених ознака смера, позива одговарајућу функцију: *levo_za_1()* или *desno_za_1()*. У супротном приказује на екрану поруку. Излаз програма за изабрани смер и задати број помераја може се видети на слици 2.2 д).

Слика 2.2 а) Излаз програма **primer2_2** (први део)

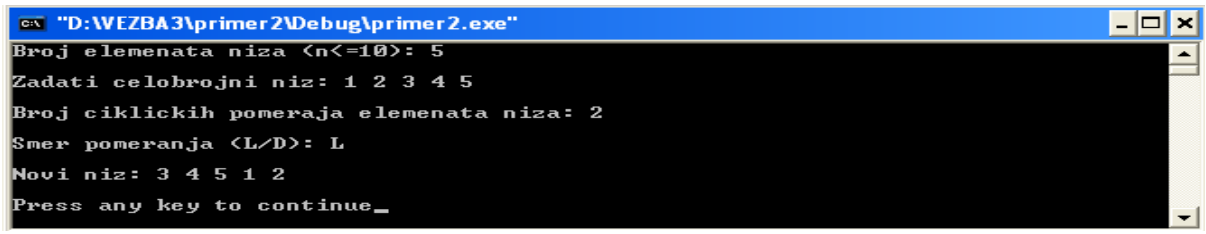
Слика 2.2 б) Излаз програма **primer2_2** (други део)



Слика 2.2 в) Излаз програма **primer2_2** (трећи део)



Слика 2.2 г) Излаз програма **primer2_2** (четврти део)



Слика 2.2 д) Излаз програма **primer2_2** (пети део)

Изворни кôд програма **primer2_3**

```
1:      /*primer2_3.c*/
2:      /*Sortiranje svake od zadatih n m-torki celih brojeva (n<=10,m=10)*/
3:      /*u neopadajuci poredak (sortiranje vrsta matrice dimenzija nxm)*/
4:
5:      #include <stdio.h>
6:      #define MAX     10      /*broj nizova, kao i dozvoljena duzina jednog niza*/
7:
8:      void citanje(int x[][MAX], int n);
9:      void prikaz(int x[][MAX], int n);
10:     void sortiranje(int x[], int n);
11:     void zamena(int *x, int *y);
12:
13:     main()
14:     {       int i, n, niz[MAX][MAX];
```

```
15:
16:             do
17:             {       printf("\nBroj nizova? "); scanf("%d", &n);
18:             }while(n<1 || n>MAX);
19:
20:             citanje(niz, n);
21:
22:             for(i=0; i<n; i++)
23:                     sortiranje(niz[i],MAX);
24:
25:             prikaz(niz, n);
26:
27:             return 0;
28:     }
29:     /*Funkcija koja cita sa tastature elemente dvodimenzionalnog niza*/
30:     void citanje(int x[][MAX], int n)
31:     {       int i, j;
32:
33:             for(i=0; i<n; i++)
34:             {       printf("\nElementi %d. celobrojnog niza: ", i+1);
35:                     for(j=0; j<MAX; j++)
36:                             scanf("%d", &x[i][j]);
37:             }
38:     }
39:     /*Funkcija koja prikazuje na ekranu elemente dvodimenzionalnog niza*/
40:     void prikaz(int x[][MAX], int n)
41:     {       int i, j;
42:
43:             printf("\nRezultat sortiranja:\n");
44:             for(i=0; i<n; i++)
45:             {       printf("\n");
46:                     for(j=0; j<MAX; j++)
47:                             printf("%d ", x[i][j]);
48:             }
49:             printf("\n\n");
50:     }
51:     /*Funkcija koja dovodi u neopadajuci poredak jednodimenzionalni niz*/
52:     void sortiranje(int x[], int n)
53:     {       int i, j;
54:
55:             for(i=0; i<n-1; i++)
56:                     for(j=i+1; j<n; j++)
57:                             if(x[i]>x[j])
58:                                     zamena(&x[i], &x[j]);
59:     }
60:     /*Funkcija koja zamenjuje vrednosti dvema promenljivama*/
61:     void zamena(int *x, int *y)
62:     {       int pom;
63:             pom = *x; *x = *y; *y = pom;
64:     }
```

Анализа програма **primer2_3**

Програм у овом примеру сортира низове задатог дводимензионалног низа целих бројева. Сваки низ од *n<=10* задатих низова (број елемената сваког низа је *MAX=10*) треба да се доведе у неопадајући поредак.

У посебне функције издвојени су следећи делови програма: читање дводимензионалног низа са тастатуре, сортирање једног низа у задатом дводимензионалном, замена места неуређеном пару елемената у једном низу и приказ на екрану резултујућег дводимензионалног низа.

У линијама 8. и 9. написане су декларације функција *citanje()* и *prikaz()*. Свака од ових функција треба да приступи елементима дводимензионалног низа, декларисаног у остатку програма. Функције добијају у својим аргументима следеће информације:

- показивач на дводимензионални низ, написан помоћу заграда и помоћу познате дужине сваког од низова (*int x[][MAX]*),
- број низова у дводимензионалном низу (*int n*).

Резултат функције *citanje()* је иницијализовани низ у оперативној меморији, а резултат функције *prikaz()* су вредности елемената низа на екрану. Од ових функција нема повратних вредности.

У линији 10. написана је декларација функције *sortiranje()*, која сортира елементе једнодимензионалног низа у неопадајући поредак и због тога користи аргументе: показивач на низ (*int x[]*) и дужину низа (*int n*). Резултат ове функције је сортирани низ у оперативној меморије и нема повратне вредности.

За поступак сортирања методом избора потребна је замена места елементима (ако нису међусобно у траженом поретку). У линији 11. написана је декларација функције *zamena().* И ова функција не враћа вредност остатку програма а користи два аргумента, показиваче на две локације у меморији чије садржаје треба да замени (*int *x,int *y*).

У линијама од 30. до 38. написана је дефиниција функције *citanje()*. Аргумент функције, који је показивач на дводимензионални низ, написан је помоћу заграда (*int x[][MAX])* што значи да се користи директан (индексни) приступ елементима низа: функција чита бројеве са тастатуре и смешта их од адреса свих елемената низа, редом *(&x[i][j],i=0,..,n-1, j=0,..,MAX-1)*.

У линијама од 40. до 50. написана је дефиниција функције *prikaz()*, која приступа директно елементима дводимензионалног низа *(x[i][j],i=0,..,n-1, j=0,..,MAX-1)* и приказује на екрану њихове садржаје.

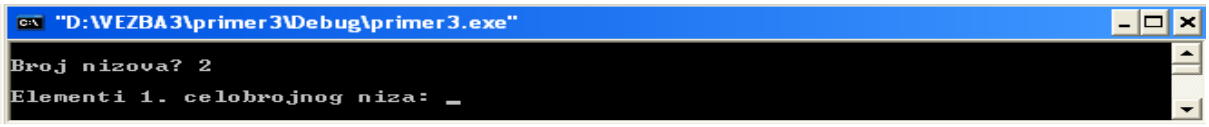
У линијама од 52. до 59. написана је дефиниција функције *sortiranje()*, која методом избора сортира једнодимензионални низ у неопадајући поредак. Када пронађе међу паровима елемената пар који није у траженом поретку, позива функцију *zamena()* и предаје јој стварне аргументе: адресе елемената у неуређеном пару *(&x[i], &x[j]).*

У линијама од 61. до 64. написана је дефиниција функције *zamena()*, која користи индиректан приступ двема локацијама у оперативној меморији (показиваче на ове локације) и помоћну променљиву *(pom)* да замени њихове садржаје *(pom = *x; *x = *y; *y = pom)* (линија 63.).

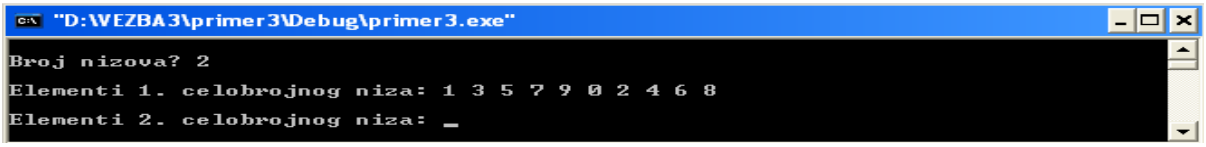
Програм из главне функције тражи са тастатуре број низова *n* (слика 2.3 а)), а онда позива функцију *citanje()*. Из ове функције исписује поруку на екрану, за унос сваког низа (слике 2.3 б) и 2.3 в)). Када добије све низове, програм у једној *for* петљи (линије 22. и 23.), у *n* итерација позива функцију *sortiranje()*, а

из ове, по потреби и функцију *zamena()*. Следи позив функције *prikaz()*, чији се резултат, за један задати дводимензионални низ, може видети на слици 2.3 г).

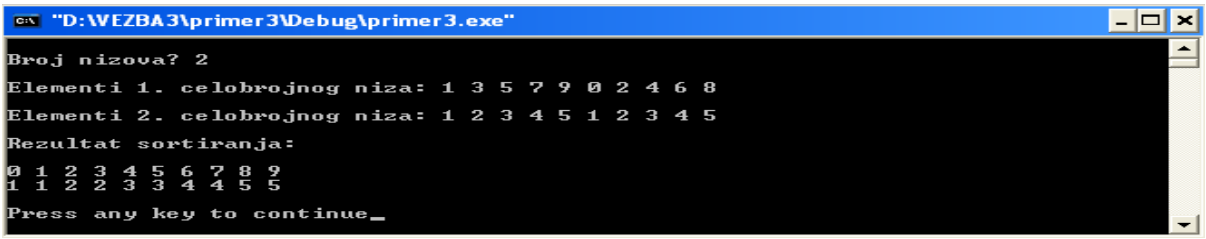
Слика 2.3 а) Излаз програма **primer2_3** (први део)



Слика 2.3 б) Излаз програма **primer2_3** (други део)



Слика 2.3 в) Излаз програма **primer2_3** (трећи део)



Слика 2.3 г) Излаз програма **primer2_3** (четврти део)

Изворни кôд програма **primer2_4**

```
1:      /*primer2_4.c*/
2:      /*Odredjivanje duzine znakovnog niza i kopiranje*/
3:      /*jednog u drugi znakovni niz (standardne funkcije strlen() i strcpy())*/
4:
5:      #include <stdio.h>
6:
7:      int duzina_stringa(char *s);
8:      char *kopira_string(char *s1, char *s2);
9:
10:     main()
11:     {       char bafer[]   = "BAFER BAFER BAFER BAFER";
12:             char poruka[] = "Zadati znakovni niz";
13:
14:             printf("\nTESTIRANJE FUNKCIJE: duzina_stringa()\n");
15:             printf("\nSadrzaj znakovnog niza je: %s\n", poruka);
16:             printf("\nBroj znakova je: %d\n", duzina_stringa(poruka));
17:
18:             printf("\n\nTESTIRANJE FUNKCIJE: kopira_string()\n");
19:             printf("\nBafer pre kopiranja: %s\n", bafer);
20:             printf("\nBafer posle kopiranja: %s\n\n", kopira_string(bafer, poruka));
21:
```

```
22:             return 0;
23:     }
24:     /*Funkcija koja odredjuje duzinu zadatog znakovnog niza*/
25:     int duzina_stringa(char *s)
26:     {       char *ptr;
27:
28:             ptr = s;
29:             while(*ptr != '\0')
30:                     ptr++;
31:
32:             return (ptr - s);
33:     }
34:     /*Funkcija koja kopira jedan u drugi znakovni niz*/
35:     char *kopira_string(char *s1, char *s2)
36:     {       char *ptr1, *ptr2;
37:
38:             ptr1 = s1;
39:             ptr2 = s2;
40:             while(*ptr2 != '\0')
41:             {       *ptr1 = *ptr2;
42:                     ptr1++;
43:                     ptr2++;
44:             }
45:             *ptr1 = '\0';
46:
47:             return s1;
48:     }
```

Анализа програма **primer2_4**

Програм у овом примеру садржи реализацију стандардних функција за одређивање дужине низа карактера и за копирање садржаја из једног у други низ карактера.

У линији 7. написана је декларација функције *duzina_stringa()*, која показује да ће функција користити показивач на почетак низа карактера *s* (*char *s*), одредити и враћати дужину овог низа (вредност типа *int*).

У линији 8. написана је декларација функције *kopira_string(),* која показује да ће функција користити показиваче на низове карактера s1 и s2, које користи при копирању (*char *s1, char *s2*) и враћати показивач на почетак резултујућег низа (ознака *char ** написана је испред имена функције *kopira_string*).

Декларације функција одговарају декларацијама стандардних функција: *strlen()* и *strcpy()* и може се видети из описа функција који следе да су то реализације поменутих *string* функција.

Већина функција које раде са низовима карактера користи као аргумент и враћа остатку програма: показивач на низ (показивач на први карактер у низу).

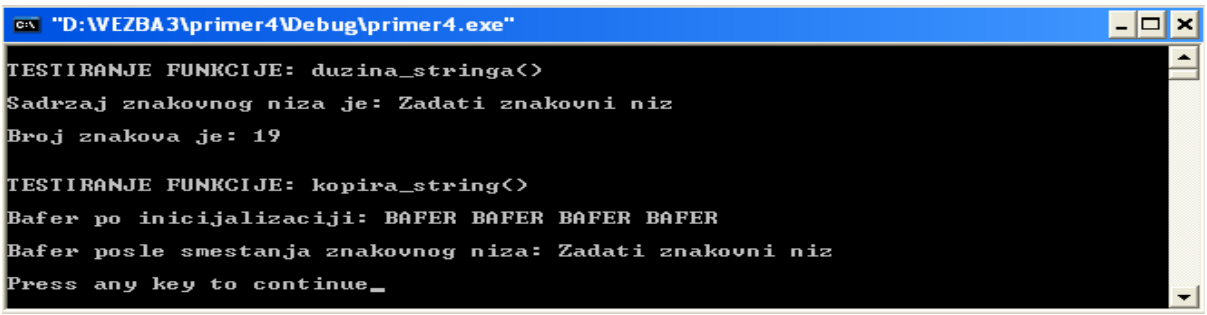
У линијама од 25. до 33. написана је дефиниција функције *duzina_stringa()*. На почетку функције декларисан је показивач *ptr* на променљиву типа *char* (линија 26.). У линији 28. овај показивач је иницијализован на адресу коју је функција добила у аргументу (показивач на

низ карактера *s*). Све док садржај на који показује *ptr* није ознака за крај *('\0')*, показивач *ptr* се инкрементира, што значи да показује на следећи карактер у низу (*while* петља у линијама 29. и 30.). Када се региструје крај низа, функција враћа разлику адреса последњег и првог карактера (ptr – s).

У линијама од 35. до 48. написана је дефиниција функције *kopira_string*(). На почетку функције декларисани су показивачи *ptr1* и *ptr2* на променљиве типа *char* (линија 36.). У линијама 38. и 39. ови показивачи су иницијализовани на адресе, које је функција добила у аргументима (показивачи на низове карактера *s1* и *s2*). Све док садржај на који показује *ptr2* није ознака за крај другог низа *('\0')*, садржај на који показује *ptr2* додељује се карактеру на који показује *ptr1* и показивачи *ptr1* и *ptr2* се инкрементирају, што значи да показују на следеће карактере у низовима *s1* и *s2*, респективно (*while* петља у линијама од 40. до 44.). Када се региструје крај другог низа, карактеру на који показује *ptr1* додељује се ознака за крај *('\0')* и функција враћа адресу резултујућег, првог низа *(s1).*

Програм у главној функцији декларише и иницијализује два низа карактера (линије 11. и 12.)*: bafer* и *poruka.* После приказа на екрану садржаја низа *poruka* (линија 15.) програм приказује на екрану резултат позива функције *duzina_stringa()*, за аргумент показивач на низ *poruka* (линија 16.). Из линије 19. програм приказује првобитан садржај низа *bafer*, а из линије 20. садржај истог низа, али после копирања из низа *poruka* у овај низ.

На слици 2.4 може се видети да функција *duzina_stringa()* даје број само корисних карактера низа и да функција *kopira_string*() од почетка преписује постојећи садржај првог низа *(bafer)* садржајем другог низа карактера *(poruka).*



Слика 2.4 Излаз програма **primer2_4**

Изворни кôд програма **primer2_5∗**

```
1:      /∗primer2_5.c∗/
2:      /∗Prikaz na ekranu zadatog niza, u originalnom i inverznom poretku∗/
3:
4:      #include <stdio.h>
5:      #define MAX    10          /∗dozvoljena duzina niza∗/
6:
7:      void cita_niz(int x[], int n);
8:      void stampa_niz(int x[], int n, int broj);
9:      void originalni(int x[], int n);
10:     void inverzni(int x[], int n);
11:
```

```
12:    main()
13:    {       int n, opcija, niz[MAX];
14:
15:            do
16:            {       printf("\nDuzina niza (n<=%d): ", MAX);
17:                    scanf("%d", &n);
18:            }while(n<1 || n>MAX);
19:
20:            cita_niz(niz, n);
21:
22:            do
23:            {       printf("\n1. Originalni niz");
24:                    printf("\n2. Inverzni niz");
25:                    printf("\n\nIzbor: ");
26:                    scanf("%d", &opcija);
27:            }while(opcija<1 || opcija>2);
28:
29:            stampa_niz(niz, n, opcija);
30:
31:            return 0;
32:    }
33:    /*Funkcija koja cita sa tastature elemente niza*/
34:    void cita_niz(int x[], int n)
35:    {       int i;
36:            printf("\nUnesite elemente niza: ");
37:            for(i=0; i<n; i++)
38:                    scanf("%d", &x[i]);
39:    }
40:    /*Funkcija koja bira nacin prikaza niza: originalni ili inverzni*/
41:    void stampa_niz(int x[], int n, int broj)
42:    {       void (*ptr)(int x[], int n);      /*ptr je pokazivac na funkciju*/
43:
44:            if(broj==1)
45:                    ptr = originalni;         /*ptr pokazuje na pocetak f-je originalni()*/
46:            else
47:                    ptr = inverzni;           /*ptr pokazuje na pocetak f-je inverzni()*/
48:            ptr(x, n);                        /*poziv odgovarajuce funkcije*/
49:    }
50:    /*Funkcija koja prikazuje na ekranu originalni niz*/
51:    void originalni(int x[], int n)
52:    {       int i;
53:            printf("\nZadati niz: ");
54:            for(i=0; i<n; i++)
55:                    printf("%d ", x[i]);
56:            printf("\n\n");
57:    }
58:    /*Funkcija koja prikazuje na ekranu inverzni niz*/
59:    void inverzni(int x[], int n)
60:    {       int i;
```

```
61:             printf("\nInverzni niz: ");
62:             for(i=n-1; i>=0; i--)
63:                     printf("%d ", x[i]);
64:             printf("\n\n");
65:     }
```

Анализа програма **primer2_5***

Показивачи могу бити декларисани у програмима да чувају адресе објеката било ког типа (основног или изведеног), али и адресе функција у оперативној меморији.

У програму овог примера користи се показивач на функцију. Програм приказује на екрану задати низ целих бројева у директном и инверзном поретку (замењена места првом и последњем елементу, другом и претпоследњем... .

У линијама 7. и 8. написане су декларације функција *cita_niz()* и *stampa_niz().* Прва од ових функција користи као аргументе показивач на целобројни низ и дужину низа, тако да може све учитане бројеве да смести у низ. Друга наведена функција користи као аргументе показивач на целобројни низ, дужину низа и цео број за избор начина приказа низа (оригинални или инверзни). Ове две функције не враћају вредност остатку програма.

У линијама 9. и 10. написане су декларације функција *originalni()* и *inverzni().* Аргументи за сваку од ових функција су: адреса низа и дужина низа, повратних вредности нема.

Функција *cita_niz()* урађена је на већ анализиран начин (дефиниција је написана у линијама од 34. до 39.).

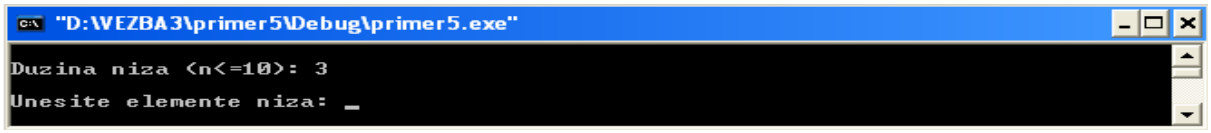
Функција *stampa_niz*() (дефиниција у линијама од 41. до 49.) има на почетку (линија 42.) декларацију показивача на било коју функцију која користи два аргумента (показивач на низ типа *int* и цео број типа *int*) и нема повратну вредност *(void (*ptr)(int x[], int n))*. У зависности од вредности трећег аргумента (*broj*) функције *stampa_niz*(), показивач се иницијализује на адресу једне од функција (*originalni()* или *inverzni()*), а онда позива изабрану функцију на коју *ptr* показује *(ptr(x, n)).*

Дефиниције функције *originalni() (*линије од 51. до 57.) и функције *inverzni()* (линије од 59. до 65.) садрже: наредбе за приказ на екрану свих елемената низа од првог (прва наведена функција) и приказ на екрану свих елемената низа од последњег (друга наведена функција).

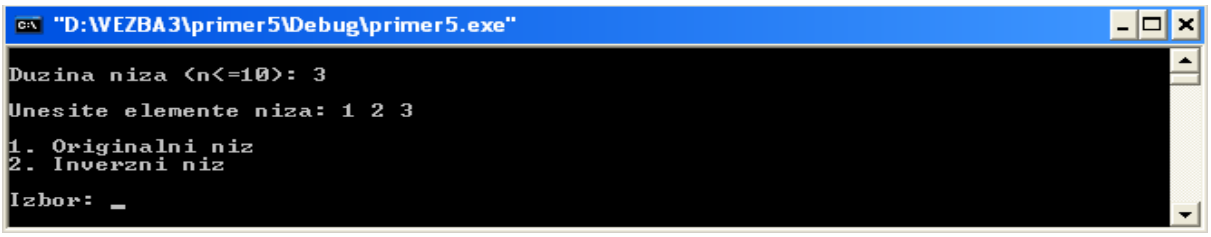
Програм из главне функције тражи са тастатуре дужину низа (слика 2.5 а)), а када учита дужину низа (линије од 15. до 18.) и саме елементе низа тако што позива функцију *cita_niz()* (слика 2.5 б)). Помоћу *do..while* петље (линије од 22. до 27.) чита са тастатуре опцију 1. или 2. (слика 2.5 в)), а онда позива функцију *stampa_niz()* предајући јој изабрану опцију. Ова опција биће даље коришћена за избор функције *originalni()* или *inverzni().*

На слици 2.5 г) може се видети излаз прогама за изабрани низ и инверзан поредак.

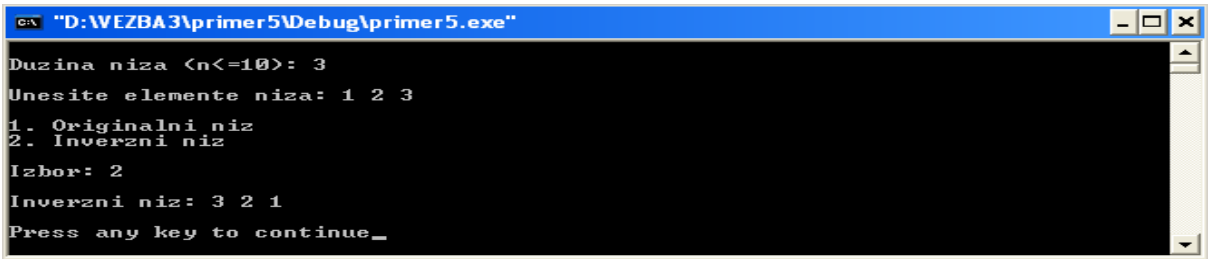
Слика 2.5 а) Излаз програма **primer2_5** (први део)



Слика 2.5 б) Излаз програма **primer2_5** (други део)



Слика 2.5 в) Излаз програма **primer2_5** (трећи део)



Слика 2.5 г) Излаз програма **primer2_5** (четврти део)

Изворни кôд програма **primer2_6**

```
1:      /*primer2_6.c*/
2:      /*Prikaz na ekranu vrednosti globalnih i lokalnih promenljivih*/
3:      /*u raznim tackama programa*/
4:
5:      #include  <stdio.h>
6:
7:      void stampa(void);
8:
9:      int x = 10, y = 20;
10:
11:     main( )
12:     {       printf("\nPre poziva funkcije:  x = %d, y = %d\n", x, y);
13:             stampa( );
14:             printf("\nPosle poziva funkcije:  x = %d, y = %d\n\n", x, y);
15:
16:             return 0;
17:     }
18:     /*Funkcija koja prikazuje na ekranu sadrzaje lokalnih promenljivih x i y*/
19:     /*ako postoje deklaracije i inicijalizacije globalnih promenljivih x i y*/
20:     void stampa(void)
21:     {       int x = 100, y = 200;
22:             printf("\nUnutar funkcije:  x = %d, y = %d\n", x, y);
23:     }
```

Анализа програма **primer2_6**

Променљива декларисана унутар једне функције има локални досег (досег функције) јер је види само та функција. Да би садржај овакве променљиве био доступан и изменљив у некој другој функцији програма користи се механизам предаје аргумената и повратних вредности од функције.

Други начин за размену података између појединих функција једног програма је декларација променљиве ван сваке функције. У том случају променљива има глобални досег (досег датотеке) и виде је све функције датотеке у којој је декларисана.

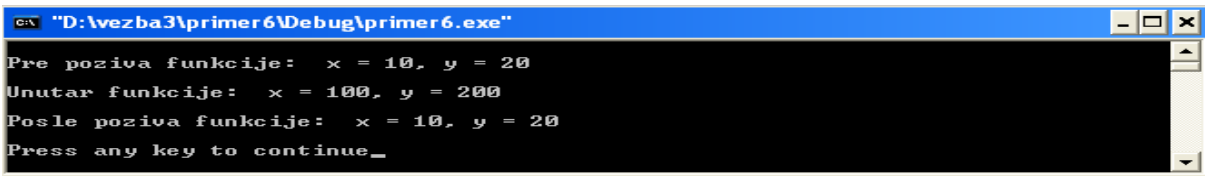
У програму овог примера декларисане су глобалне и локалне променљиве.

Ван главне функције (линија 9.) декларисане су и иницијализоване глобалне променљиве *x* и *y.* Из линије 12. програм приказује на екрану садржаје поменутих глобалних променљивих (пре позива функције *stampa()).*

Из линије 13. програм позива функцију *stampa()*. Унутар дефиниције ове функције (линије од 20. до 23.) декларисане су и иницијализоване локалне променљиве *x* и *y* и приказани њихови садржаји.

Дакле, могуће је користити исто име за променљиву локалног и глобалног досега. Локални досег има предност над глобалним досегом и то се може видети на слици 2.6:

- пре позива функције *stampa*(), програм даје на излазу вредности глобалних променљивих,
- у току извршавања функције *stampa()*, програм даје на излазу вредности локалних променљивих за ову функцију,
- после повратка из функције *stampa( )*, програм опет даје на излазу вредности глобалних променљивих, због тога што главна функција нема локалне променљиве са истим именима као што су глобалне које се користе.



Слика 2.6 Излаз програма **primer2_6**

Изворни кôд програма **primer2_7**

```
1:      /*primer2_7.c*/
2:      /*Prikaz na ekranu vrednosti staticke i automatske*/
3:      /*promenljive u raznim iteracijama jedne petlje*/
4:
5:      #include  <stdio.h>
6:
7:      void stampa(void);
8:
9:      main( )
10:     {       int i;
```

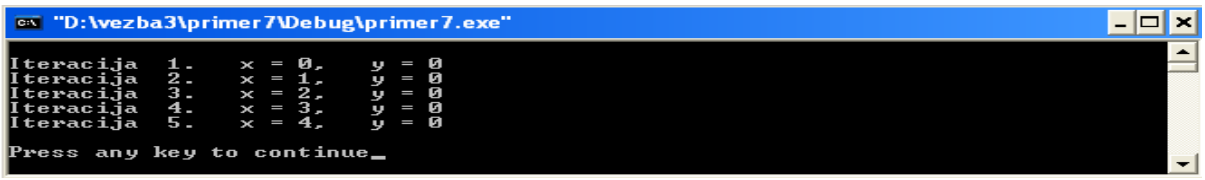
```
11:
12:             printf("\n");
13:             for(i = 0; i < 5; i++)
14:             {       printf("Iteracija %2d.   ", i+1);
15:                     stampa( );
16:             }
17:             printf("\n");
18:
19:             return 0;
20:     }
21:     /*Funkcija koja stampa i inkrementira vrednosti */
22:     /*staticke promenljive x i automatske promenljive y*/
23:     void stampa(void)
24:     {       static int x = 0;
25:             int y = 0;
26:
27:             printf("x = %d,   y = %d\n", x++, y++);
28:     }
```

Анализа програма **primer2_7**

Све променљиве, које су до сада биле коришћене у примерима ове збирке, декларисане су као аутоматске (подразумевани начин). Статичка променљива има реч *static* испред типа променљиве.

Дефиниција функције *stampa()* (линије од 23. до 28.) има декларацију и иницијализацију локалних променљивих: статичке (x) и аутоматске (y). Наредба за приказ на екрану садржаја ових променљивих има и инкремент сваке од њих, за следећи позив функције *stampa()*.

Програм из главне функције у више итерација једне петље позива функцију *stampa()*. У свакој итерацији статичка променљива чува вредност из претходне итерације, док за аутомаску то не важи (слика 2.7).



Слика 2.7 Излаз програма **primer2_7**

Практични савети

- ✓ Вредност стварног аргумента може бити измењена унутар функције само ако је аргумент предат функцији по референци.
- ✓ Када се аргумент предаје по референци функција добија његову стварну адресу и има могућност измене његовог садржаја, без наредбе повратка.

Евентуалне грешке

- ➢ Низови се увек предају функцији по референци а не по вредности.
- ➢ Код дефиниције показивача на функцију неопходно је написати заграде око идентификатора показивача.

Задаци за припрему вежбе 2. у лабораторији

Задатак 1.

Које ће вредности приказати на екрану следећи програм на језику *C*?

```
#include <stdio.h>
void f(int *x, int *y, int *z);
main()
{       int a = 0, b = 0, c = 0;
        f(&a, &b, &c);
        printf("\n%d %d %d", a, b, c);
}
void f(int *x, int *y, int *z)
{       *x = 2; *y = 4; *z = 6;       }
```

Задатак 2.

Написати програм на језику *C* који помоћу одговарајућих функција одређује да ли је низ од *n (n≤10)* целих бројева, задатих преко тастатуре, сортиран у нерастући или неопадајући поредак.

Задаци за вежбу 2. у лабораторији

Урадити следеће програме на језику *C* издвајањем подзадатака програма у посебне функције, уз прослеђивање овим функцијама из главне функције, свих потребних информација о низовима.

Задатак 1.

Написати програм на језику *C* који за низ од *n* (*n*≤10) целих бројева, задатих преко тастатуре, даје извештај на екрану о средњој, највећој и најмањој вредности.

Задатак 2.

Написати програм на језику *C* који сортира низ од *n* (*n*≤10) целих бројева, задатих преко тастатуре, у нерастући / неопадајући поредак, у зависности од команде задате преко тастатуре.

Задатак 3.

Написати програм на језику *C* који прихвата са тастатуре два реда текста (два знаковна низа) све док се за један од њих не унесе празан ред и након уноса свака два реда текста приказује на екрану извештај о томе да ли су ови редови међусобно истог садржаја.

Задатак 4.*

Написати програм на језику *C* који од *n* (*n*≤10) низова, сваки од *m* (*m*≤10) целих бројева, задатих преко тастатуре, формира и приказује на екрану низ чији су елементи редом:

а) највеће вредности редом задатих низова,

б) суме елемената редом задатих низова.

Задатак 5.*

Написати програм на језику *C* који у матрици димензија *nxn* (*n*≤10) од целих бројева, задатих преко тастатуре:

а) сортира све елементе прве врсте у неопадајући поредак,

б) сортира све остале врсте по првој врсти.

Вежба 3.

Динамичка додела меморије у програмима на језику *C*

Динамичка додела простора у меморији

Измена величине динамички додељеног простора у меморији

Ослобађање динамички додељеног простора у меморији

Примери

Изворни кôд програма **primer3_1**

```
1:    /*primer3_1.c*/
2:    /*Dinamicka dodela/oslobadjanje memorije za string od 100 karaktera*/
3:    /*za niz od 100 celih brojeva ili za niz od 100 realnih brojeva (opciono)*/
4:
5:    #include <stdio.h>
6:    #include <stdlib.h>
7:    #define MAX                    100    /*dozvoljena duzina niza*/
8:    #define GRESKA_STRING          0      /*status prve poruke greske*/
9:    #define GRESKA_CELI            1      /*status druge poruke greske*/
10:   #define GRESKA_REALNI          2      /*status trece poruke greske*/
11:   #define GRESKA_TIP             3      /*status cetvrte poruke greske*/
12:
13:   char  *poruke_o_greskama[] =
14:   {       "\nNije uspela dodela memorije za string!\n",
15:           "\nNije uspela dodela memorije za celobrojni niz!\n",
16:           "\nNije uspela dodela memorije za realni niz!\n",
17:           "\nNedozvoljena oznaka tipa!\n"                          };
18:
19:   void greska(int status);
20:
21:   main()
22:   {       char   *string;
23:           int    *celi, tip;
24:           float  *realni;
25:
26:           printf("\n\nIzaberite tip niza: c (char), i (int), ili f (float): ");
27:           tip = getchar(); getchar();   /*Tip niza i znak za prelaz u novi red*/
28:
29:           switch(tip)
30:           {
31:                   case 'c':             /*dodela memorije za string*/
32:                           if((string = malloc(MAX * sizeof(char))) == NULL)
33:                                   greska(GRESKA_STRING);
34:                           puts("\nDodeljena je memorija za string.\n");
35:                           free(string);
36:                           break;
37:                   case 'i':             /*dodela memorije za celobrojni niz*/
38:                           if((celi = malloc(MAX * sizeof(int))) == NULL)
39:                                   greska(GRESKA_CELI);
40:                           puts("\nDodeljena je memorija za celobrojni niz\n");
41:                           free(celi);
42:                           break;
```

```
43:                     case 'f':               /*dodela memorije za realni niz*/
44                              if((realni = malloc(MAX * sizeof(float))) == NULL)
45:                                     greska(GRESKA_REALNI);
46:                             puts("\nDodeljena je memorija za realni niz\n");
47:                             free(realni);
48:                             break;
49:                     default:
50:                             greska(GRESKA_TIP);
51:             }
52:             return 0;
53:     }
54:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
55:     void greska(int status)
56:     {       puts(poruke_o_greskama[status]);
57:             exit(1);
58:     }
```

Анализа програма **primer3_1**

Програми почетничког нивоа углавном користе само статичку доделу меморије за променљиве. Овај начин декларације је једноставан и применљив код малих објеката у програму као што су: променљиве основног типа, низови веома малог броја променљивих основног типа... . Недостатак овог начина декларације код низа је што током извршавања програма нису могуће измене броја елемената, као ни укидање низа када више није потребан.

Динамичка додела меморије за променљиве омогућује по потреби програма: доделу меморијског простора, измене и ослобађање величине додељеног простора. То значи да није неопходна декларација на почетку програма. Програм почне да се извршава и онда открије да му је потребан низ. Током свог извршавања овај програм тражи у меморији места за низ, ако је касније потребно тражи још места у меморији и онда када му низ више није потребан тражи да се укину сва додељена места у меморији. Све ово реализују одговарајуће стандардне функције а прате показивачи, резултати ових функција. Дакле, нема прекидања извршавања програма, измена у изворном коду, активирања преводиоца и поновног покретања програма.

Програм овог примера динамички додељује и ослобађа меморијски простор за низове разних типова података.

Поред главне програм садржи и функцију *greska()* која приказује на екрану поруке о евентуалним грешкама и прекида извршавање програма.

После дефиниција симболичких константи, потребних програму (линије од 7. до 11.) декларисан је ван сваке функције низ показивача на низове карактера (линије од 13. до 17.) и то је назначено у декларацији, испред имена низа *(char *poruke_o_greskama[]).* Празне заграде говоре о томе да је у питању низ (преводилац одређује дужину на основу иницијализације), а ознака испред имена низа *char ** да се ради о низу показивача на тип *char.* Ови константни знаковни низови, поруке о разним могућим грешкама у програму (написане у линијама од 14. до 17.), смештају се негде у оперативну меморију, а њихове адресе у низ *poruke_o_greskama[]*. Први показивач у овом низу *poruke_o_greskama[0]* садржи адресу прве поруке *"\nNije uspela dodela memorije za string!\n"... .* Низ показивача на поруке ефикасније користи оперативну

меморију у односу на низ порука (резервише простор за низ показивача и за сваку поруку тачно онолико бајтова колико је потребно, док у случају низа порука, неопходно је резервисати за сваку поруку онолико бајтова колико је потребно за најдужу поруку у низу).

У линијама од 55. до 58. написана је дефиниција функције *greska()*. Из линије 56. функција приказује на екрану садржај низа карактера на који показује елемент низа *poruke_o_greskama*, са индексом *status* (добијеном у аргументу функције) (*puts(poruke_o_greskama[status]);*). Прекид програма остварује се помоћу функције *exit()* (линија 57.) из библиотеке *stdlib.h*. Када се овој функцији преда вредност 1, оперативном систему се прослеђује статус грешке.

У главној функцији (линије од 22. до 24.) декларисани су показивачи на променљиве разних типова (*char, int* и *float*) који ће бити коришћени у програму као показивачи на низове (*string, celi* и *realni*). За тип елемента низа, изабран са тастатуре (линија 26. и 27), програм у наредби *switch* (линије од 29. до 51.) реализује динамичку доделу и ослобађање меморије.

У сва три случаја меморија се динамички додељује помоћу функције *malloc()* из стандардне библиотеке *stdlib.h* и ослобађа помоћу функције *free()* из исте библиотеке. Биће анализиран само један случај за низ карактера (*string*), због тога што се и сви остали реализују на сличан начин.

Позивом функције *malloc()*, програм тражи резервисање континуалног блока меморије (линија 32.). Функцији се предаје број бајтова потребног меморијског простора (који се за низ добија када се број елемената низа помножи величином типа елемента низа, што је резултат оператора *sizeof()*). Ако има довољно простора у меморији функција *malloc()* враћа показивач генеричког типа *void*, који се конвертује у жељени тип показивача (у овом случају у показивач на низ карактера *string*). Помоћу наредбе *if* (линије 32. и 33.), програм тестира резултат функције *malloc()*. Ако је овај резултат вредност *NULL* показивача то значи да није било довољно меморије (функција није успешно обавила свој задатак) и да на екрану треба да се испише (помоћу позива функције *greska()*) одговарајућа порука (*"\nNije uspela dodela memorije za string!\n"*). Функцији *greska()* предаје се вредност симбола *GRESKA_STRING (0),* што значи да иста функција приказује на екрану поруку чију адресу чува показивач *poruke_o_greskama[0]*. У супротном случају тражени број бајтова (*MAX*sizeof(char)*) резервисан је у меморији и приказује се порука о томе на екрану (линија 34.). Из линије 35. позива се функција *free()* која ослобађа меморију дозвољавајући да иста поново буде додељена.

За низ карактера у позиву функције *malloc()* може се изоставити *sizeof(char)* (вредност је 1). Код осталих низова то није случај (линије од 38. до 41. и линије од 44. до 47.). На сликама 3.1 а) и 3.1 б) приказани су порука на екрану за избор типа и излаз програма за изабрани тип *float*.

Слика 3.1 а) Излаз програма **primer3_1** (први део)

Слика 3.1 б) Излаз програма **primer3_1** (други део)

Изворни кôд програма **primer3_2**

```
1:    /*primer3_2.c*/
2:    /*Sazimanje zadatog niza celih brojeva na niz pozitivnih brojeva*/
3:    /*Verzija1:dinamicka dodela/oslobadjanje memorije za niz iz glavne f-je*/
4:
5:    #include <stdio.h>
6:    #include <stdlib.h>
7:    #define MAX                    100    /*dozvoljena duzina niza*/
8:    #define GRESKA_DODELA          0      /*status prve poruke greske*/
9:    #define GRESKA_PROMENA         1      /*status druge poruke greske*/
10:
11:   char *poruke_o_greskama[] =
12:   {       "\nNije uspela dinamicka dodela memorije za niz!\n",
13:           "\nNije uspela promena dinamicki dodeljene memorije za niz!\n"     };
14:
15:   void cita_niz(int niz[], int n);
16:   int sazima_niz(int niz[], int n);
17:   void stampa_niz(int niz[], int n);
18:   void greska(int status);
19:
20:   main()
21:   {       int n1, n2, *niz;
22:
23:           do
24:           {       printf("\nBroj elemenata niza, n<=%d? ", MAX);
25:                   scanf("%d", &n1);
26:           }while(n1<1 || n1>MAX);
27:
28:           if((niz = malloc(n1 * sizeof(int))) == NULL)
29:                   greska(GRESKA_DODELA);
30:
31:           cita_niz(niz, n1);
32:           n2 = sazima_niz(niz, n1);
33:
34:           if(n2<n1)
35:                   if((niz = realloc(niz,n2 * sizeof(int))) == NULL)
36:                           greska(GRESKA_PROMENA);
37:
38:           stampa_niz(niz, n2);
39:
40:           free(niz);
41:           return 0;
42:   }
43:   /*Funkcija koja cita sa tastature niz*/
44:   void cita_niz(int niz[], int n)
45:   {       int i;
46:
47:           printf("\nElementi niza:\t");
```

```
48:             for(i=0; i<n; i++)
49:                     scanf("%d", &niz[i]);
50:     }
51:     /*Funkcija koja sazima zadati niz na niz pozitivnih brojeva*/
52:     int sazima_niz(int niz[], int n)
53:     {       int i, *ptr1, *ptr2;
54:             ptr1 = ptr2 = niz;
55:
56:             for(i=0; i<n; i++)
57:             {       if(*ptr1 > 0)
58:                     {       *ptr2 = *ptr1;
59:                             ptr2++;
60:                     }
61:                     ptr1++;
62:             }
63:             return (ptr2 - niz);
64:     }
65:     /*Funkcija koja prikazuje na ekranu niz*/
66:     void stampa_niz(int niz[], int n)
67:     {       int i;
68:
69:             printf("\nNovi niz:\t");
70:             for(i=0; i<n; i++)
71:                     printf("%d ", niz[i]);
72:             printf("\n\n");
73:     }
74:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
75:     void greska(int status)
76:     {       puts(poruke_o_greskama[status]);
77:             exit(1);
78:     }
```

Анализа програма **primer3_2**

Програм овог примера има задатак да из задатог низа целих бројева избаци све елементе који нису позитивни и да промени дужину низа на број преосталих елемената (сажимање низа).

У линијама од 15. до 18. декларисане су функције различите од главне: *cita_niz(), sazima_niz(), stampa_niz()* и *greska().*

Функција *greska()* реализована је на исти начин као у претходном примеру и неће бити анализирана у овом и у следећим примерима.

Из декларација осталих функција може се видети да свака од њих користи показивач на низ типа *int* и дужину низа и да функција *sazima_niz()* враћа цео број а функције *cita_niz()* и *stampa_niz()* немају повратну вредност.

Имена функција описују и њихове задатке. Функција *cita_niz()* прихвата са тастатуре целе бројеве и смешта их у низ чију је адресу и дужину добила у аргументима (дефиниција функције је у линијама од 44. до 50.).

Функција *stampa_niz()* узима вредности елемената низа, чију је адресу и дужину добила у аргументима и приказује вредности елемената на екрану (дефиниција функције је у линијама од 66. до 73.).

Функција *sazima_niz* (линије од 52. до 64.) уводи два показивача *ptr1* и *ptr2* и иницијализује их на адресу првог елемента низа *(ptr1 = ptr2 = niz;).* Показивач *ptr1* пратиће даље адресе свих елемената оригиналног низа, а показивач *ptr2* биће искоришћен за формирање новог низа од позитивних елемената оригиналног (пратиће адресе свих елемената новог низа). Алгоритам реализован у *for* петљи (линије од 56. до 62.) за оригинални низ, који има три елемента (на пример), следећег је облика:

- поставља се питање да ли је вредност првог елемента оригиналног низа, на који показује *ptr1* већа од 0. Ако је то случај (линија 57.), формира се први елемент новог низа (на који показује *ptr2)* који добија вредност првог елемента оригиналног низа *(*ptr2=*ptr1;),* оба показивача се инкрементирају и прелази се на следећу итерацију петље (линије 58., 59. и 61.);
- поставља се питање да ли је вредност другог елемента оригиналног низа на који показује *ptr1* већа од 0. Ако то није случај (линија 61.), инкрементира се само показивач *ptr1* и прелази се на следећу итерацију петље;
- поставља се питање да ли је вредност трећег елемента оригиналног низа на који показује *ptr1* већа од 0. Ако је то случај (линија 57.) формира се други елемент новог низа (на који показује *ptr2)* који добија вредност трећег елемента оригиналног низа *(*ptr2=*ptr1;)*, оба показивача се инкрементирају и прелази се на следећу итерацију петље (линије 58., 59. и 61.).

Алгоритам се наставља све док има елемената оригиналног низа.

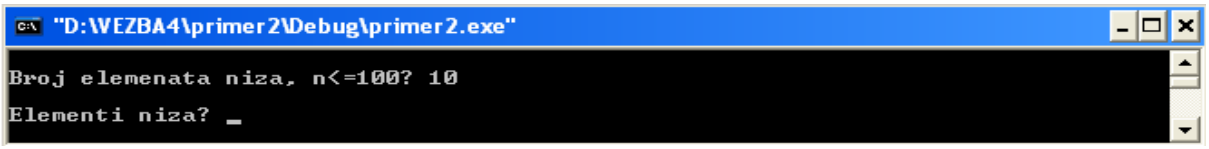
У главној функцији (линије од 20. до 42.) програм тражи (слика 3.2 а)) да се унесе број елемената оригиналног низа *n1* и чита са тастатуре овај број. Онда позива функцију *malloc()* да се динамички додели меморија за тражени број елемената низа (*n1*sizeof(int)*). Ако није прошао захтев и функција врати *NULL* показивач, на екрану се приказује порука о неуспелој додели (линија 29.).

Ако је додељена меморија следи позив функције *cita_niz()* са стварним вредностима аргумената (адреса *niz* и број елемената *n1*) (слика 3.2 б)).

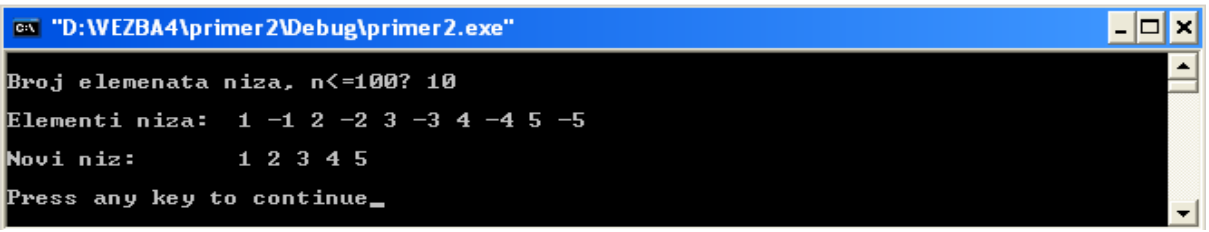
Програм даље позива функцију *sazima_niz()* за стварне аргументе (адресу *niz* и број елемената *n1),* а резултат функције додељује променљивој *n2.* Ако је нови број елемената мањи од претходно додељеног (линија 34.), програм позива функцију *realloc()* и предаје јој адресу низа (*niz)* и број елемената на који хоће да промени низ, помножен са величином типа једног елемента *(n2*sizeof(int))*. И овде је потребна провера резултата функције (линије 35. и 36.) без обзира што се ради о промени (додељени простор може бити померен од неке друге адресе). Следи позив функције *stampa_niz()* за стварне вредности аргумената (адреса *niz* и број елемената *n1).*

Да би меморија била даље ослобођена позвана је функција *free()*. Овој функцији потребно је предати само адресу у меморији од које је динамички резервисан простор за низ и она ослобађа простор за цео низ (линија 40.). На слици 3.2 в) приказан је резултат програма за један задати низ.

Слика 3.2 а) Излаз програма **primer3_2** (први део)



Слика 3.2 б) Излаз програма **primer3_2** (други део)



Слика 3.2 в) Излаз програма **primer3_2** (трећи део)

Изворни кôд програма **primer3_3∗**

```
1:      /∗primer3_3.c∗/
2:      /∗Sazimanje zadatog niza celih brojeva na niz pozitivnih brojeva∗/
3:      /∗Verzija2: dinamicka dodela/oslobadjanje memorije za niz iz raznih f-ja∗/
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #define MAX                     100     /∗dozvoljena duzina niza∗/
8:      #define GRESKA_DODELA           0       /∗status prve poruke greske∗/
9:      #define GRESKA_PROMENA          1       /∗status druge poruke greske∗/
10:
11:     char ∗poruke_o_greskama[] =
12:     {       "\nNije uspela dinamicka dodela memorije za niz!\n",
13:             "\nNije uspela promena dinamicki dodeljene memorije za niz!\n"     };
14:
15:     void cita_niz(int ∗∗ptr_ptr_niz, int n);
16:     int sazima_niz(int ∗∗ptr_ptr_niz, int n);
17:     void stampa_niz(int ∗ptr_niz, int n);
18:     void greska(int status);
19:
20:     main()
21:     {       int n1, n2, ∗niz;
22:
23:             do
24:             {       printf("\nBroj elemenata niza, n<=%d? ", MAX);
25:                     scanf("%d", &n1);
26:             }while(n1<1 || n1>MAX);
27:
28:             cita_niz(&niz, n1);
29:             n2 = sazima_niz(&niz, n1);
30:             stampa_niz(niz, n2);
31:
32:             return 0;
33:     }
```

```
34:   /*Funkcija koja cita niz sa tastature*/
35:   void cita_niz(int **ptr_ptr_niz, int n)
36:   {       int i;
37:
38:           if((*ptr_ptr_niz = malloc(n * sizeof(int))) == NULL)
39:                   greska(GRESKA_DODELA);
40:
41:           printf("\nElementi niza:\t");
42:           for(i=0; i<n; i++)
43:                   scanf("%d", *ptr_ptr_niz+i);
44:   }
45:   /*Funkcija koja sazima zadati niz na niz pozitivnih brojeva*/
46:   int sazima_niz(int **ptr_ptr_niz, int n)
47:   {       int i, n2, *ptr1, *ptr2;
48:           ptr1 = ptr2 = *ptr_ptr_niz;
49:
50:           for(i=0; i<n; i++)
51:           {       if(*ptr1 > 0)
52:                   {       *ptr2 = *ptr1;
53:                           ptr1++; ptr2++;       }
54:                   else
55:                           ptr1++;
56:           }
57:           n2 = ptr2 - *ptr_ptr_niz;
58:
59:           if(n2<n)
60:                   if((*ptr_ptr_niz = realloc(*ptr_ptr_niz,n2 * sizeof(int))) == NULL)
61:                           greska(GRESKA_PROMENA);
62:
63:           return n2;
64:   }
65:   /*Funkcija koja prikazuje na ekranu rezultujuci niz*/
66:   void stampa_niz(int *ptr_niz, int n)
67:   {       int i;
68:
69:           printf("\nNovi niz:\t");
70:           for(i=0; i<n; i++)
71:                   printf("%d ", *(ptr_niz+i));
72:           printf("\n\n");
73:
74:           free(ptr_niz);
75:   }
76:   /*Funkcija koja prikazuje na ekranu poruke o greskama*/
77:   void greska(int status)
78:   {       puts(poruke_o_greskama[status]);
79:           exit(1);
80:   }
```

Анализа програма **primer3_3∗**

Програм овог примера решава исти задатак као и програм претходног примера, само што су у посебне функције издвојени подзадаци: динамичка додела и читање низа, сажимање и промена динамички додељене меморије, као и приказ на екрану низа и ослобађање динамички додељене меморије. Ову верзију програма могуће је реализовати помоћу показивача на показивач.

Показивач на показивач је променљива целобројног типа (опсег зависи од меморије рачунара), која садржи адресу другог показивача. Да би се декларисао показивач који садржи адресу другог показивача треба додати још један оператор ∗ у декларацији. Тако за следеће наредбе важи:

int ∗ptr; - декларише показивач *ptr* на променљиву типа *int* (показивачу *ptr* може се доделити адреса било које променљиве типа *int*).

int ∗∗ptr_ptr; - декларише показивач *ptr_ptr* на показивач на променљиву типа *int* (показивачу *ptr_ptr* може се доделити адреса места у меморији које чува адресу било које променљиве типа *int*).

Број оператора ∗ испред имена показивача одговара броју показивача који треба следити да би се приступило променљивој, у овом случају типа *int.*

Механизам показивача на показивач искоришћен је у овом програму код доделе и ослобађања меморије за низ.

У линијама од 15. до 18. декларисане су функције са истим именима као у претходном примеру: *cita_niz(), sazima_niz(), stampa_niz()* и *greska().* Разлика у односу на истоимене функције из претходног примера је што сада функције различите од главне садрже динамичку доделу, промену величине и ослобађање меморије, а у главној функцији написани су само позиви ових функција. При томе све информације о низу размењују се у програму преко аргумената и повратних вредности од функција. Да би функције радиле како је предвиђено неопходне су им следеће информације: показивач на показивач на низ и дужина низа (то се може видети из њихових декларација).

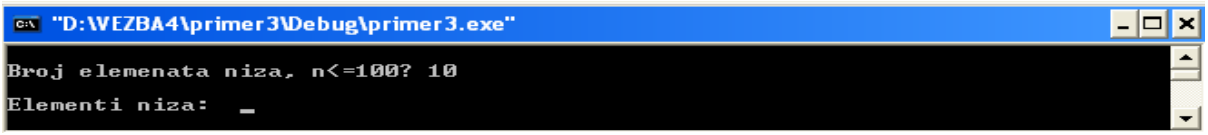
Дефиниција функције *cita_niz()* написана је у линијама од 35. до 44. Први аргумент функције је показивач на показивач на тражени низ, а други је дужина низа, прочитана је са тастатуре (слика 3.3 а)).

Функција може изменити садржај сваке променљиве чији је показивач добила као аргумент, као и садржај сваког показивача чији је показивач (показивач на показивач) добила као аргумент. Из линије 38. позвана је функција *malloc()* за доделу меморије низу целих бројева, дужине *n* и њен резултат је додељен показивачу *∗ptr_ptr_niz* (адреса оригиналног низа). Ово је било могуће због тога што је аргумент функције била адреса показивача (показивач на показивач) *∗∗ptr_ptr_niz*. Поменути показивач *∗ptr_ptr_niz* је адреса уведеног низа и сваком елементу овог низа може се приступити на индиректан начин: *scanf("%d",∗ptr_ptr_niz+i)* (линија 43.). Функција *cita_niz()* прихвата са тастатуре и елементе низа (слика 3.3 б)).

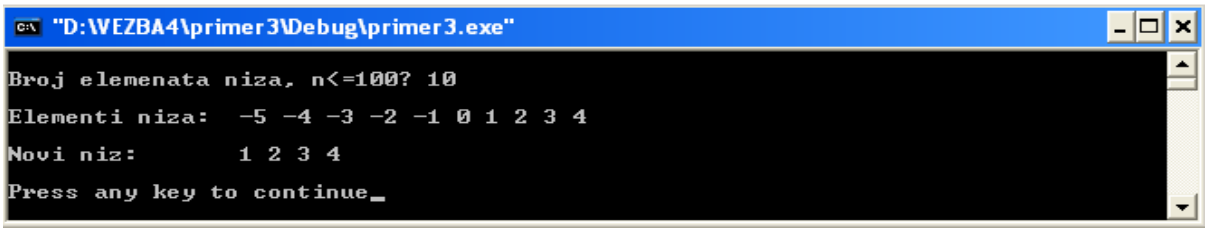
Функција *sazima_niz()* (дефиниција у линијама од 46. до 64.) користи показивач на показивач да би резултат функције *realloc()* доделила показивачу (адреса измењеног низа). За сажимање низа користи се већ познат алгоритам.

Функција *stampa_niz()* (дефиниција у линијама од 66. до 75.) користи као аргументе показивач на низ и дужину низа, приказује на екрану (линија 71.) претходно формирани низ (слика 3.3в)) и укида низ (*free()).*

Слика 3.3 а) Излаз програма **primer3_3** (први део)



Слика 3.3 б) Излаз програма **primer3_3** (други део)



Слика 3.3 в) Излаз програма **primer3_3** (трећи део)

Изворни кôд програма **primer3_3**

```
1:      /*primer3_4.c*/
2:      /*Spajanje do 10 redom zadatih nepraznih stringova*/
3:      /* (dinamicka dodela/oslobadjanje memorije za stringove) */
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #include <string.h>
8:      #define BROJ            10      /*dozvoljen broj stringova*/
9:      #define MAX             80      /*broj korisnih karaktera u stringu*/
10:     #define DODELA_S1       0       /*status prve poruke greske*/
11:     #define DODELA_S2       1       /*status druge poruke greske*/
12:     #define PROMENA_S1      2       /*status trece poruke greske*/
13:
14:     char *poruke_o_greskama[] =
15:     {       "\nNije uspela dinamicka dodela memorije za prvi zadati string!\n",
16:             "\nNije uspela dinamicka dodela memorije za sledeci string!\n",
17:             "\nNije uspelo prosirenje dinamicki dodeljene memorije!\n"          };
18:
19:     void greska(int status);
20:
21:     main()
22:     {       int i, l;
23:             char *string1, *string2;
24:
25:             if((string1 = malloc(MAX+1)) == NULL)
26:                     greska(DODELA_S1);
27:
28:             if((string2 = malloc(MAX+1)) == NULL)
```

```
29:                     greska(DODELA_S2);
30:
31:             puts("Prvi string:");
32:             gets(string1);
33:             l = strlen(string1);
34:
35:             if((string1 = realloc(string1, l+1)) == NULL)
36:                     greska(PROMENA_S1);
37:
38:             for(i=0; i<BROJ; i++)
39:             {       puts("\nSledeci string:");
40:                     gets(string2);
41:
42:                     if(strlen(string2)==0)
43:                             break;
44:
45:                     l += strlen(string2);
46:
47:                     if((string1 = realloc(string1,l+1)) == NULL)
48:                             greska(PROMENA_S1);
49:
50:                     strcat(string1, string2);
51:             }
52:             puts(string1);
53:
54:             free(string1);
55:             free(string2);
56:
57:             return 0;
58:     }
59:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
60:     void greska(int status)
61:     {       puts(poruke_o_greskama[status]);
62:             exit(1);
63:     }
```

Анализа програма **primer3_4**

Програм у овом примеру надовезује један на крај другог, до 10 непразних низова карактера, задатих преко тастатуре, користећи динамичку доделу меморије за први задати низ, проширење динамички додељене меморије за сваки следећи низ, као и ослобађање динамички додељене меморије. Због једноставности све ово је реализовано у главној функцији. Може се за вежбу урадити и друга верзија истог програма са кôдом распоређеним у више функција (помоћу показивача на показиваче као аргументима ових функција).

На почетку главне функције (линија 23.) написана је декларација показивача: *char *string1* и *char *string2* (који ће бити искоришћени као показивачи на прве карактере у низовима *string1* и *string2*, редом).

Програм даље (линије од 25. до 29.) тражи доделу меморије за низове *string1* и *string2* (у питању су низови карактера и због тога је у оба случаја

аргумент функције *malloc()* само дужина низа). Ако је додељена потребна меморија програм тражи да се унесе преко тастатуре први знаковни низ (слика 3.4 а)), смешта задати знаковни низ у *string1,* одређује његову дужину (линија 33.) и позива функцију *realloc()* (линија 35.) да би дужина стринга *string1* (први аргумент функције *realloc()*) била промењена на дужину задатог знаковног низа (други аргумент функције *realloc()*).

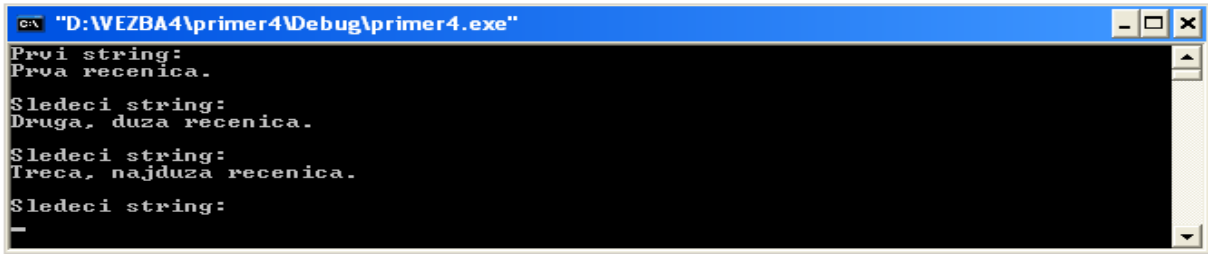
У петљи (линије од 38. до 51.) програм понавља описани поступак за сваки следећи задати знаковни низ: тражи да се зада преко тастатуре (слике од 3.4 б) до 3.4 г)), учита низ знакова и смести у *string2* (линија 40.), одреди његову дужину (линија 42.), динамички увећа дужину резултујућег *string1* за ову дужину (линије од 45. до 48.) и дода садржај *string2* у *string1* (линија 50.).

Програм на крају (линија 52.) приказује на екрану резултујући низ (слика 3.4 д)) и ослобађа динамички додељену меморију (линије 54. и 55.).

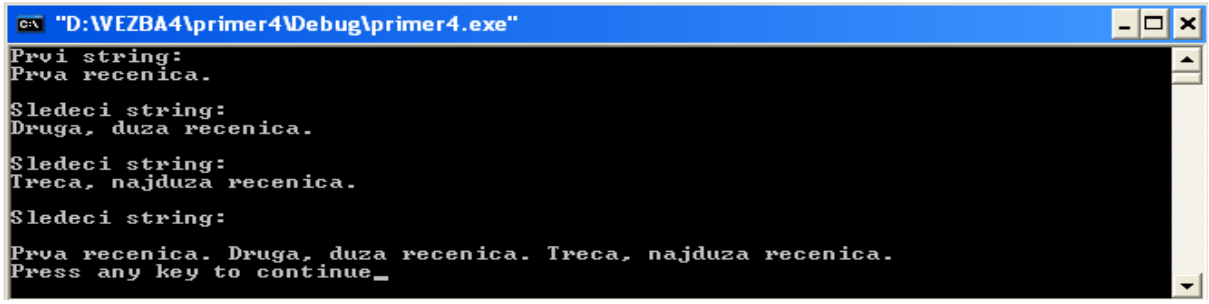
Слика 3.4 а) Излаз програма **primer3_4** (први део)

Слика 3.4 б) Излаз програма **primer3_4** (други део)

Слика 3.4 в) Излаз програма **primer3_4** (трећи део)



Слика 3.4 г) Излаз програма **primer3_4** (четврти део)



Слика 3.4 д) Излаз програма **primer3_4** (пети део)

Изворни кôд програма **primer3_5∗**

```
1:      /∗primer3_5.c∗/
2:      /∗Prikaz na ekranu onih nizova od zadatih m<=10 nizova (svaki∗/
3:      /∗duzine n<=10) ciji su svi elementi razliciti od nule (dinamicka∗/
4:       /∗dodela/oslobadjanje memorije za niz nizova celih brojeva)∗/
5:
6:      #include <stdio.h>
7:      #include <stdlib.h>
8:      #define MAX     10   /∗dozvoljeni broj imena∗/
9:
10:     void cita_niz(int niz[], int n);
11:     void stampa_niz(int niz[], int n);
12:     void greska(void);
13:
14:     main()
15:     {       int ∗niz[MAX], i, m, n;
16:
17:             do
18:             {       printf("\nBroj nizova: ");
19:                     scanf("%d", &m);
20:                     printf("\nBroj elemenata jednog niza: ");
21:                     scanf("%d", &n);
22:             }while(m<1 || m>MAX || n<1 || n>MAX);
23:
24:             for(i=0; i<m; i++)
25:             {       if((niz[i] = malloc(n ∗ sizeof(int))) == NULL)
26:                             greska();
27:             }
28:
29:             for(i=0; i<m; i++)
30:             {       printf("\n%d. niz: ", i+1);
31:                     cita_niz(niz[i], n);
32:             }
33:
34:             printf("\nNizovi bez broja 0:\n");
35:             for(i=0; i<m; i++)
36:                     stampa_niz(niz[i], n);
37:             printf("\n\n");
38:
39:             for(i=0; i<m; i++)
40:                     free(niz[i]);
41:             return 0;
42:     }
43:     /∗Funkcija koja cita sa tastature jednodimenzionalni niz∗/
44:     void cita_niz(int niz[], int n)
45:     {       int i;
46:             for(i=0; i<n; i++)
47:                     scanf("%d", &niz[i]);
```

```
48:     }
49:     /*Funkcija koja prikazuje na ekranu jednodimenzionalni niz*/
50:     void stampa_niz(int niz[], int n)
51:     {       int i, ima_nulu = 0;
52:
53:             for(i=0; i<n; i++)
54:                     if(niz[i]==0)
55:                             ima_nulu = 1;
56:             printf("\n");
57:             if(!ima_nulu)
58:                     for(i=0; i<n; i++)
59:                             printf("%d ", niz[i]);
60:     }
61:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
62:     void greska(void)
63:     {       printf("\nNema dovoljno memorije za nizove.\n");
64:             exit(1);
65:     }
```

Анализа програма **primer3_5***

Програм овог примера приказује на екрану оне низове из задатог низа целобројних низова који међу елементима не садрже вредност 0.

Поред главне функције у програму су предвиђене: функција *cita_niz(),* за читање са тастатуре једног низа, функција *stampa_niz(),* за приказ на екрану једног низа у низу низова и функција *greska()*. Прва поменута функција (дефиниција у линијама од 44. до 48.) користи показивач на низ типа *int* и дужину низа и помоћу стандардне функције *scanf()* смешта учитане бројеве редом у низ. Друга поменута функција (дефиниција у линијама од 50. до 60.) у петљи претражује низ и ако било који елемент има вредност 0, статусна променљива *ima_nulu* добија вредност 1 (линија 55.) и не дозвољава приказ на екрану низа (не извршавају се наредбе у линијама 58. и 59.).

У линији 15. главне функције, програм декларише низ од *MAX (10)* показивача на променљиву типа *int (int *niz[MAX])*.

Када прихвати са тастатуре број низова *m<=10* (слика 3.5 а)) и дужину сваког низа *n<=10* (слика 3.5 б)), програм у *for* петљи, за *m* једнодимензионалних низова (линије од 24. до 27.), позива функцију *malloc(),* да додели меморију за *n* елемената сваког низа типа *int* и да адресу сваког низа (ако је успела додела) чува показивач *niz[i]*. Дакле, адресе од *m* низова су:

- *niz[0]* је адреса низа*: niz[0] [0], niz[0] [1],..., niz[0] [n-1],*

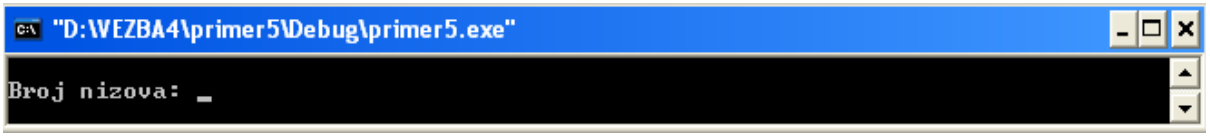
...

- *niz[m-1]* је адреса низа*: niz[m-1] [0], niz[m-1] [1],..., niz[m-1] [n-1].*

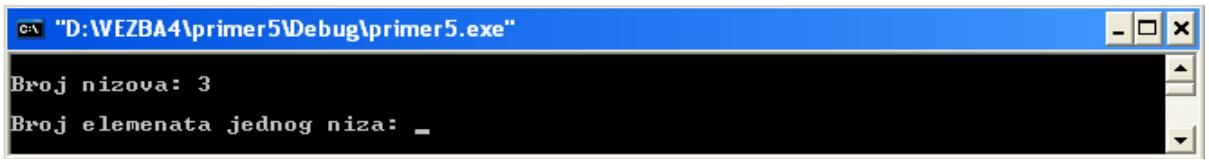
Ако додела меморије успе, програм у *for* петљи, за *m* једнодимензионалних низова (линије од 29. до 32.), позива функцију *cita_niz()* и у свакој итерацији петље предаје јој адресу једног једнодимензионалног низа *(niz[i])* и дужину низа *(n).* Помоћу ове функције програм тражи елементе сваког низа (слике 3.5 в), 3.5 г) и 3.5 д)) и смешта их од одговарајућих адреса.

Програм у *for* петљи, за *m* једнодимензионалних низова (линије 35. и 36.), позива функцију *stampa_niz(),* којој, у свакој итерацији петље, предаје адресу *(niz[i])* и дужину низа *(n))* и приказује низ ако нема нулу (слика 3.5 ђ)).

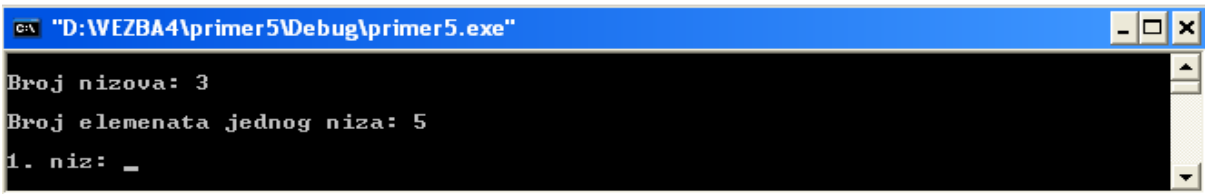
На крају главне функције позива се стандардна функција за ослобађање меморије, опет у *for* петљи, за сваки од *m* низова (линије 39. и 40.).



Слика 3.5 а) Излаз програма **primer3_5** (први део)

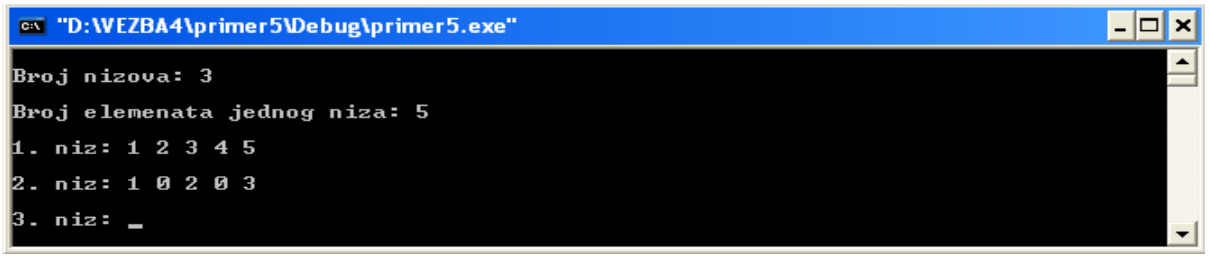


Слика 3.5 б) Излаз програма **primer3_5** (други део)

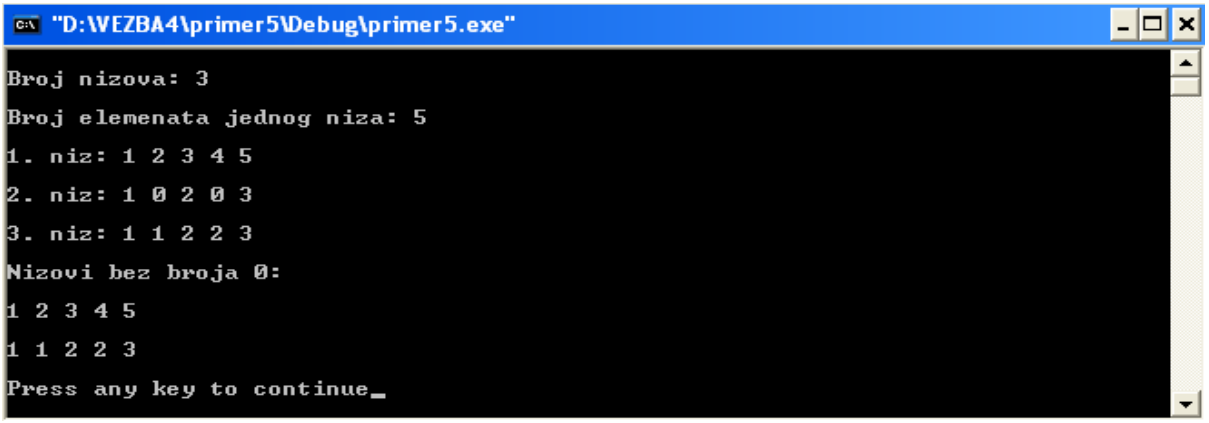


Слика 3.5 в) Излаз програма **primer3_5** (трећи део)

Слика 3.5 г) Излаз програма **primer3_5** (четврти део)



Слика 3.5 д) Излаз програма **primer3_5** (пети део)



Слика 3.5 ђ) Излаз програма **primer3_5** (шести део)

Изворни кôд програма **primer3_6∗**

```
1:      /∗primer3_6.c∗/
2:      /∗Sortiranje zadatog niza od n<=10 imena (duzina imena  m<=30) po∗/
3:      /∗abecednom redu. Za kraj unosa treba zadati prazan red (dinamicka∗/
4:      /∗dodela/oslobadjanje prostora u memorji za niz znakovnih nizova)∗/
5:
6:      #include <stdio.h>
7:      #include <stdlib.h>
8:      #include <string.h>
9:      #define BROJ                    10      /∗dozvoljeni broj imena∗/
10:     #define MAX                     30      /∗broj korisnih karaktera u stringu∗/
11:     #define GRESKA_DODELE           0       /∗status prve poruke greske∗/
12:     #define GRESKA_PROMENE          1       /∗status druge poruke greske∗/
13:
14:     char ∗poruke_o_greskama[] =
15:     {       "\nNije uspela dinamicka dodela memorije za niz!\n",
16:             "\nNije uspela promena dinamicki dodeljene memorije za niz!\n"     };
17:
18:     int cita_imena(char ∗imena[]);
19:     void sortira_imena(char ∗imena[], int n);
20:     void stampa_imena(char ∗imena[], int n);
21:     void greska(int status);
22:
23:     main( )
24:     {        int n;  char ∗∗imena;
25:
26:             if((imena = malloc(BROJ ∗ sizeof(char ∗)))==NULL)
27:                     greska(GRESKA_DODELE);
28:
29:             n = cita_imena(imena);
30:
31:             if((imena = realloc(imena, n ∗ sizeof(char ∗)))==NULL)
32:                     greska(GRESKA_PROMENE);
33:
34:             sortira_imena(imena, n);
35:             stampa_imena(imena,n);
36:
37:             free(imena);
38:
39:             return 0;
40:     }
41:     /∗Funkcija koja kreira spisak i popunjava ga imenima sa tastature∗/
42:     int cita_imena(char ∗imena[])
43:     {       int i, l;
44:
45:             puts("Unesite imena u posebnim redovima, prazan red za kraj");
46:
```

```
47:             for(i=0; i<BROJ; i++)
48:             {       if((imena[i] = malloc(MAX+1))==NULL)
49:                             greska(GRESKA_DODELE);
50:                     gets(imena[i]);
51:
52:                     if((l = strlen(imena[i])) == 0)
53:                     {       free(imena[i]);
54:                             break;
55:                     }
56:
57:                     if((imena[i] = realloc(imena[i], l+1)) == NULL)
58:                             greska(GRESKA_PROMENE);
59:             }
60:             return i;
61:     }
62:     /*Funkcija koja sortira spisak imena*/
63:     void sortira_imena(char *imena[], int n)
64:     {       int i, j; char *pom;
65:
66:             if((pom = malloc(MAX+1))==NULL)
67:                             greska(GRESKA_DODELE);
68:
69:             for(i=0; i<n-1; i++)
70:                     for(j=i+1; j<n; j++)
71:                             if(strcmp(imena[i], imena[j]) > 0)
72:                             {       strcpy(pom, imena[i]);
73:                                     strcpy(imena[i], imena[j]);
74:                                     strcpy(imena[j], pom);
75:                             }
76:             free(pom);
77:     }
78:     /*Funkcija koja prikazuje na ekranu i nakon toga ukida spisak imena*/
79:     void stampa_imena(char *imena[], int n)
80:     {       int i;
81:
82:             for(i=0; i<n; i++)
83:                     puts(imena[i]);
84:
85:             for(i=0; i<n; i++)
86:                     free(imena[i]);
87:     }
88:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
89:     void greska(int status)
90:     {       puts(poruke_o_greskama[status]);
91:             exit(1);
92:     }
```

Анализа програма **primer3_6***

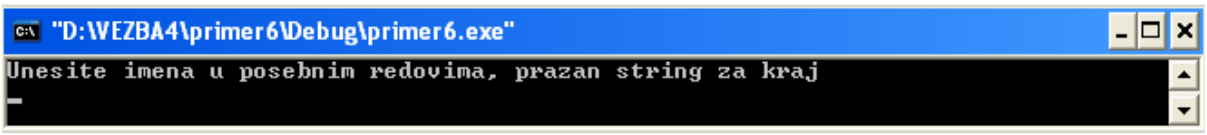
Програм у овом примеру чита низ имена са тастатуре, сортира га по абецедном реду и приказује на екрану помоћу одговарајућих функција. Динамичка додела и ослобађање меморије реализују се из главне функције.

Функција *cita_imena()* (линије од 42. до 61.) користи од остатка програма један аргумент: показивач на низ показивача. Када формални аргумент треба да буде показивач на низ показивача, он може бити написан на један од ова два начина: *char **imena* или *char *imena[]*. У овом случају ради се о низу карактера. У *for* петљи (линије од 47. до 59.) програм тражи доделу меморије за свако име у низу (адресу имена чува показивач: *imena[i]*, i=0,...,BROJ-1), чита име са тастатуре (линија 50.), одређује дужину задатог имена и повећава, ако је то могуће, величину низа додавањем дужине низа (линије 57. и 58.). Ова функција враћа број елемената формираног низа имена. Када се учита празан знаковни низ (линија 52.) онда то значи ослобађање динамички додељене меморије и крај читања имена (линије од 53. до 55.).

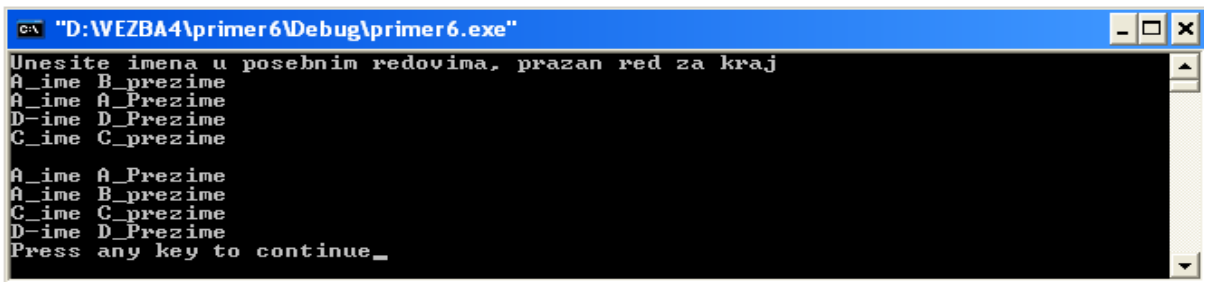
Функција *sortira_imena()* (линије од 63. до 77.) користи два аргумента: показивач на низ показивача и дужину низа показивача, тако да може сортирати елементе овог низа.

Функција *stampa_imena()* (линије од 79. до 87.) добија у аргументима: показивач на низ показивача и дужину низа показивача . У *for* петљи (линије 82. и 83.) приказује на екрану једно по једно име у низу (показивачи на ова имена су: *imena[0],..., imena[n-1]*) и ослобађа динамички додељену меморију за поменуте показиваче (линије 85. и 86.).

Програм у главној функцији позива функцију *malloc()* (линија 26.) да додели меморију за низ показивача *(imena)* на тип *char* и ако има довољно меморије величина резервисаног простора у меморији за низ показивача је: *BROJ*sizeof(char *)*. Програм онда чита са тастатуре имена помоћу функције *cita_imena()* (слика 3.6 а)), предаје јој претходно добијену адресу низа показивача и добија од ње број прочитаних имена (линија 29.). Онда евентуално мења величину додељене меморије (од адресе *imena*) на: *n*sizeof(char *)* (линије 31. и 32.). Следе позиви функција *sortira_imena()* и *stampa_imena()* (функцијама предаје адресу *imena* и дужину низа показивача (линије 34. и 35.), као и ослобађање динамички додељене меморије (линија 37.). Резултат програма, за задати низ имена, може се видети на слици 3.6 б).



Слика 3.6 а) Излаз програма **primer3_6** (први део)



Слика 3.6 б) Излаз програма **primer3_6** (други део)

Практични савети

- ✓ Динамичку доделу простора у меморији не користити код малих објеката у програму (променљивих основног типа и низова мале дужине).
- ✓ Величине динамички додељених простора у меморији треба да буду приближно исте за што више објеката у програму (због евентуалне појаве неискоришћеног простора након ослобађања меморије).
- ✓ У позивима стандардних функција за динамичку доделу и измену величине динамички додељеног меморијског простора за низ користити број елемената низа и резултат оператора *sizeof()* (уместо броја бајтова).

Евентуалне грешке

- ➢ Не заборавити проверу резултата позива стандардних функција за динамичку доделу и измену величине динамички додељеног меморијског простора, пре даљег коришћења овог меморијског простора.
- ➢ Када се простор у меморији динамички додели неопходно је и да се ослободи. Ако се меморија само додељује током извршавања програма, без ослобађања, програм ће вероватно пасти.

Задаци за припрему вежбе 3. у лабораторији

Задатак 1.

Написати програм на језику *C* који низ од *n (n≤100)* целих бројева, задатих преко тастатуре, сажима на низ бројева дељивих бројем *m* (број *m* се задаје преко тастатуре). Урадити програм помоћу динамичке доделе, промене и ослобађања меморијског простора за низ.

Задаци за вежбу 3. у лабораторији

Урадити следеће програме на језику *C* помоћу динамичке доделе, промене и ослобађања меморијског простора за низове. Издвојити подзадатке програма у посебне функције

Задатак 1.

Написати програм на језику *C* који сажима низ од *n (n≤100)* целих бројева, задат преко тастатуре, на следећи начин: избацује све елементе чије вредности нису у опсегу од 0 до 10,

Задатак 2.*

Написати програм на језику *C* који сажима низ од *n (n≤100)* целих бројева, задат преко тастатуре, на следећи начин: замењује групу узастопних нула елемената бројем нула у тој групи.

Задатак 3.

Написати програм на језику *C* који сажима знаковни низ дужине *n (n≤200),* задат преко тастатуре, на следећи начин: избацује мале заграде и знакове који се налазе између њих (претпоставка: унутар једних заграда нема других).

Задатак 4.

Написати програм на језику *C* који од једног низа редова (низа низова знакова), задатог преко тастатуре (број редова је *n (n≤10))*, формира други низ редова на следећи начин: избацује све редове који не почињу словом.

Задатак 5.*

Написати програм на језику *C* који сажима низ редова текста, задат преко тастатуре (број редова је *n (n≤10)*), на следећи начин: избацује све празне редове.

Вежба 4.

Структуре у програмима на језику *C*

Дефиниција типа структуре

Декларација примерака структуре

Приступ члановима структура

Структура као аргумент функције

Структура као повратна вредност од функције

Низови и листе структура

Примери

Изворни кôд програма **primer4_1**

```
1:      /*primer4_1.c*/
2:      /*Formiranje strukture podataka, zadatih preko tastature: */
3:      /*ime, broj indeksa i godina upisa (primena operatora tacka(.))*/
4:
5:      #include <stdio.h>
6:      #define MAX     30                      /*broj korisnih karaktera u stringu*/
7:
8:      struct student{                         /*definicija tipa strukture: student*/
9:              char ime[MAX+1];
10:             int br_indeksa, god_upisa;
11:             };
12:
13:     main()
14:     {       struct student student1;        /*deklaracija strukture: student1*/
15:
16:             printf("\nUnesite ime i prezime: ");
17:             gets(student1.ime);
18:
19:             printf("\nBroj indeksa: ");
20:             scanf("%d", &student1.br_indeksa);
21:
22:             printf("\nGodina upisa: ");
23:             scanf("%d", &student1.god_upisa);
24:
25:             printf("\n%s ", student1.ime);
26:             printf("%d %d\n\n", student1.br_indeksa, student1.god_upisa);
27:
28:             return 0;
29:     }
```

Анализа програма **primer4_1**

Структура је скуп елемената који су у неком односу у једном програму. За разлику од низова у структури је дозвољено да елементи буду различитог типа и ти елементи зову се чланови структуре.

Програм у овом примеру је један од најједноставнијих могућих са структуром података.

Да би програм користио неку структуру неопходно је претходно дефинисати тип структуре. Дефиниција типа структуре (линије од 8. до 11.) почиње службеном речи *struct* и ознаком типа структуре (*student*). Између великих заграда написане су дефиниције чланова структуре (у овом случају то су три члана: *ime* (низ који може да прихвати MAX знакова), *br_indeksa* и *god_upisa* (променљиве типа *int*). Дефиниција сваког члана има знак тачка-

зарез на крају. Дозвољено је, као што је овде случај, да више чланова истог типа буде дефинисано у истој линији (линија 10.). Дефиниција типа структуре завршава се затвореном великом заградом и знаком тачка-зарез (линија 11.). У модуларним програмима дефиниције типова структура издвајају се у посебне датотеке и због тога ће у свим програмима бити написане глобално (ван сваке функције).

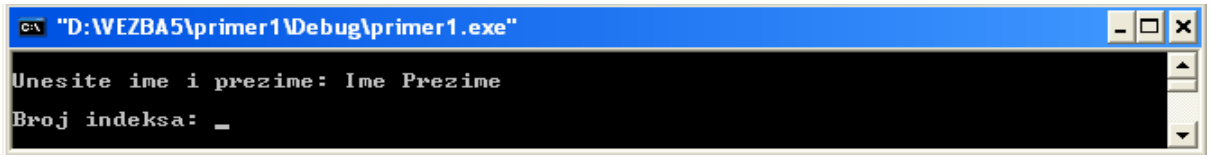
Описана дефиниција уводи тип структуре у овај програм (пружа информације о њеним члановима) али не резервише простор у меморији. Да би био резервисан овај простор неопходно је декларисати конкретну структуру дефинисаног типа. У овом случају то је структура *student1* (линија 14.).

Програм тражи да се преко тастатуре унесу подаци (слике 4.1 а), 4.1 б) и 4.1 в)) и унете податке додељује члановима структуре *student1* (*ime*, *br_indeksa* и *god_upisa*). Један од начина на који се може приступити члану структуре је оператор тачка (.). Са леве стране овог оператора пише се име структуре, а са десне име члана (*student1.ime*, *student1.br_indeksa* и *student1.god_upisa*).

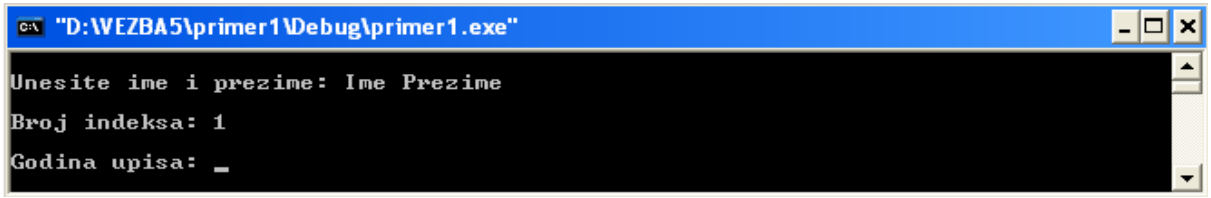
У линијама 17., 20. и 23. написане су наредбе помоћу којих се подаци задати преко тастатуре смештају у чланове структуре *student1*.

Функција *printf()* и овде се може користити (линије 25. и 26.) за приказ на екрану информација из чланова структуре *student1* (резултат програма може се видети на слици 4.1 г).

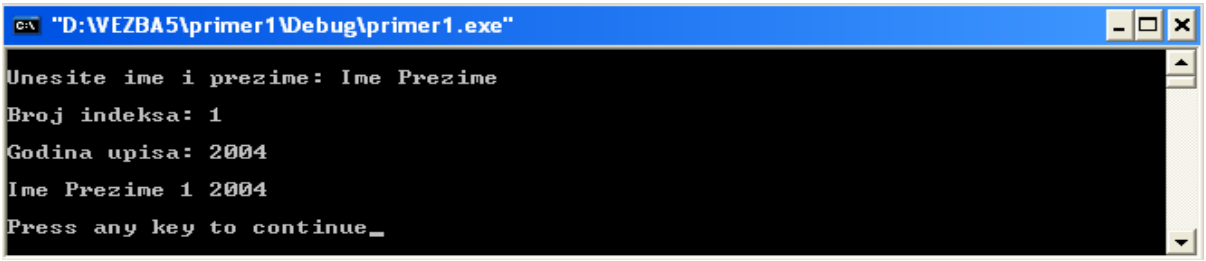
Слика 4.1 а) Излаз програма **primer4_1** (први део)



Слика 4.1 б) Излаз програма **primer4_1** (други део)



Слика 4.1 в) Излаз програма **primer4_1** (трећи део)



Слика 4.1 г) Излаз програма **primer4_1** (четврти део)

Изворни кôд програма **primer4_2**

```
1:    /*primer4_2.c*/
2:    /*Formiranje strukture od podataka zadatih preko tastature:*/
3:    /*ime ulice, broj ulaza i broj telefona (primena operatora ->)*/
4:
5:    #include <stdio.h>
6:    #define MAX1      30          /*broj korisnih karaktera u jednom stringu*/
7:    #define MAX2      15          /*broj korisnih karaktera u drugom stringu*/
8:
9:    struct adresa{
10:         char ulica[MAX1+1], telefon[MAX2+1];
11:         int br_ulaza;
12:         };
13:
14:   main()
15:   {     struct adresa adresa1;      /*deklaracija strukture tipa adresa*/
16:         struct adresa *ptr_a;       /*deklaracija pokazivaca na strukturu*/
17:
18:         ptr_a = &adresa1;           /*inicijalizacija pokazivaca*/
19:
20:         printf("\nUnesite ime ulice: ");
21:         gets(ptr_a->ulica);
22:
23:         printf("\nBroj ulaza: ");
24:         scanf("%d", &ptr_a->br_ulaza);getchar();
25:
26:         printf("\nBroj telefona: ");
27:         gets(ptr_a->telefon);
28:
29:         printf("\nul. %s %d, ", ptr_a->ulica, ptr_a->br_ulaza);
30:         printf("tel. %s\n\n", ptr_a->telefon);
31:
32:         return 0;
33:   }
```

Анализа програма **primer4_2**

У програму овог примера уведен је тип структуре *adresa* (линије од 9. до 12.), чији су чланови: два низа карактера *(ulica* и *telefon)* и целобројна променљива типа int *(br_ulaza)*.

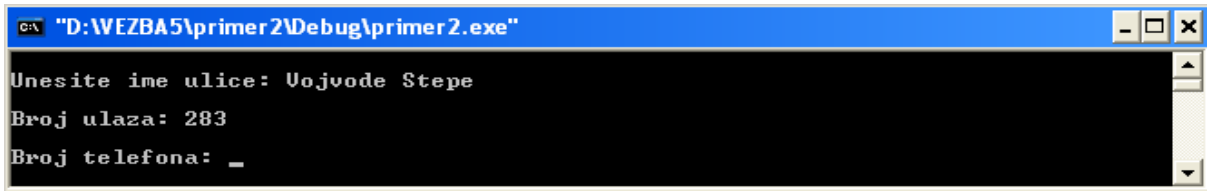
Поред декларисане структуре *adresa1* типа *adresa* (линија 15.), у главној функцији је декларисан и показивач *ptr_a* на структуру типа *adresa* (линија 16.) и иницијализован на адресу конкретне структуре *adresa1* (линија 18.).

Ако је у програму декларисан показивач на конкретну структуру (који садржи адресу првог бајта првог члана те структуре) члановима структуре се може приступити и помоћу тог показивача. Тај други начин приступа члановима структуре искоришћен је у овом примеру. Програм тражи (слике 4.2 а), 4.2 б) и 4.2 в)) да се преко тастатуре унесу подаци и смешта их у чланове структуре

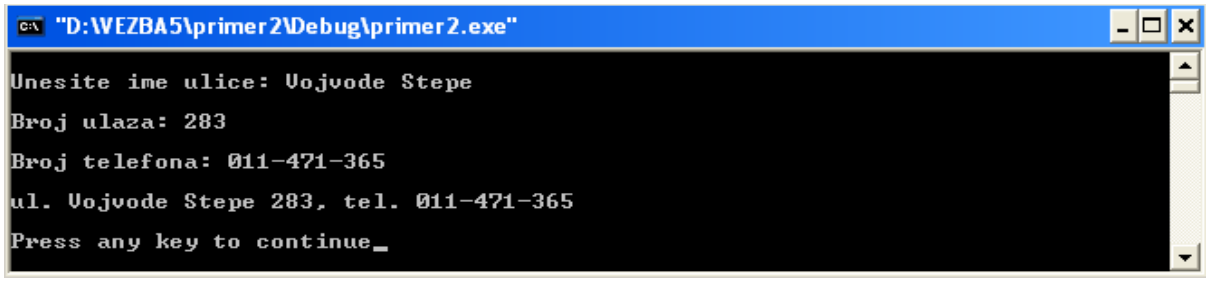
adresa1 (*ulica*, *br_ulaza* и *telefon*) помоћу оператора индиректног приступа члану структуре (->): *ptr_a->ulica, ptr_a->br_ulaza, ptr_a->telefon*. За читање података са тастатуре, смештање у одговарајуће чланове структуре (линије 21., 24. и 27.) и приказ вредности чланова структуре (линије 29. и 30.) коришћене су познате функције. Резултат програма може се видети на слици 4.2 г).

Слика 4.2 а) Излаз програма **primer4_2** (први део)

Слика 4.2 б) Излаз програма **primer4_2** (други део)



Слика 4.2 в) Излаз програма **primer4_2** (трећи део)



Слика 4.2 г) Излаз програма **primer4_2** (четврти део)

Изворни кôд програма **primer4_3**

```
1:      /*primer4_3.c*/
2:      /*Izracunavanje povrsine pravougaonika*/
3:      /*zadatih koordinata (kompleksna struktura)*/
4:
5:      #include <stdio.h>
6:
7:      struct koordinate{                      /*Definicija 1. tipa strukture*/
8:              int x, y;
9:      };
10:
11:     struct pravougaonik{                    /*Definicija 2. tipa strukture*/
12:             struct koordinate dole_levo, gore_desno;
13:             };
```

```
14:
15:     main( )
16:     {       int duzina, sirina;
17:             struct pravougaonik p1;             /*Deklaracija strukture 2. tipa*/
18:
19:             printf("\nKoordinate dole levo ugla: ");
20:             scanf("%d%d", &p1.dole_levo.x, &p1.dole_levo.y);
21:
22:             printf("\nKoordinate gore desno ugla: ");
23:             scanf("%d%d", &p1.gore_desno.x, &p1.gore_desno.y);
24:
25:             duzina = p1.gore_desno.x - p1.dole_levo.x;
26:             sirina = p1.gore_desno.y - p1.dole_levo.y;
27              printf("\nPovrsina pravougaonika: %ld \n\n", duzina*sirina);
28:
29:             return 0;
30:     }
```

Анализа програма **primer4_3**

Иако се структура сматра једним објектом у програму, она поред променљивих основног типа међу члановима може имати: показиваче, низове, друге структуре... .

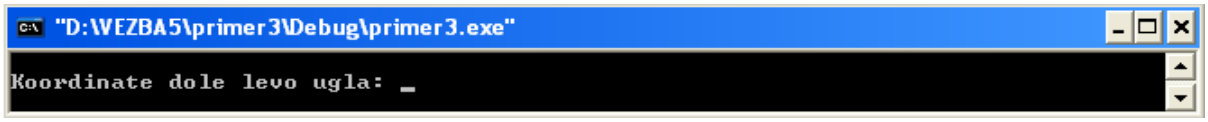
У програму овог примера који израчунава површину правоугаоника задатих координата дефинисана су два типа структуре (у линијама од 7. до 13.): *koordinate* (чланови су две променљиве типа *int*: *x, y*) и *pravougaonik* (чланови су две структуре типа *koordinate: dole_levo* и *gore_desno*).

У главној функцији је декларисана структура *p1* типа *pravougaonik*. Програм тражи да се унесу преко тастатуре по две координате доле лево и горе десно угла правоугаоника (слике 4.3 а) и 4.3 б)) и смешта их у одговарајуће чланове структуре *p1:*

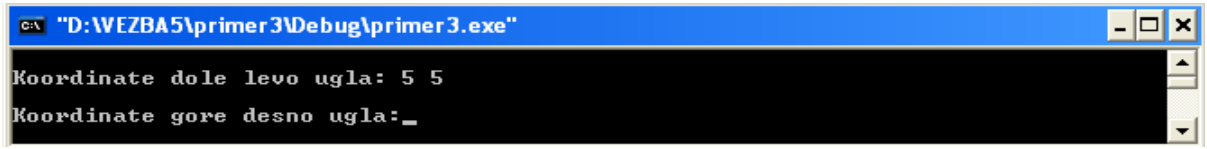
p1.dole_levo.x - први члан структуре *dole_levo*, унутар структуре *p1*,
p1.dole_levo.y - други члан структуре *dole_levo*, унутар структуре *p1*,
p1.gore_desno.x - први члан структуре *gore_desno*,унутар структуре *p1*,
p1.gore_desno.y -други члан структуре *gore_desno*,унутар структуре *p1*,

онда израчунава и приказује на екрану површину задатог правоугаоника (линије од 25. до 27.). На слици 3.3 в) може се видети резултат програма за један комплет задатих података.

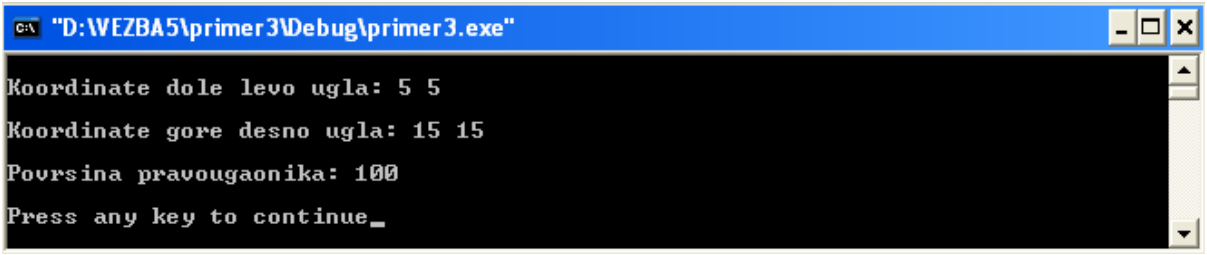
На овај начин се остварује приступ члановима комплексне структуре, која међу члановима има другу структуру.



Слика 4.3 а) Излаз програма **primer4_3** (први део)



Слика 4.3 б) Излаз програма **primer4_3** (други део)



Слика 4.3 в) Излаз програма **primer4_3** (трећи део)

Изворни кôд програма **primer4_4**

```
1:      /*primer4_4.c*/
2:      /*Formiranje strukture i prikaz na ekranu iz posebne funkcije*/
3:      /*podataka iz strukture (ime,smer,broj indeksa i godina upisa studenta*/
4:
5:      #include <stdio.h>
6:      #define MAX     30                  /*broj korisnih karaktera u stringu*/
7:
8:      struct student{
9:              char ime[MAX+1], smer[MAX+1];
10:             int br_indeksa, god_upisa;
11:             };
12:
13:     void stampa(struct student x);
14:
15:     main()
16:     {       struct student student1;
17:
18:             printf("\nIme i prezime studenta: ");
19:             gets(student1.ime);
20:
21:             printf("\nSmer na kome je student: ");
22:             gets(student1.smer);
23:
24:             printf("\nBroj indeksa i godina upisa (u obliku xx/xxxx): ");
25:             scanf("%d/%d", &student1.br_indeksa, &student1.god_upisa);
26:
27:             stampa(student1);
28:
29:             return 0;
30:     }
31:     /*Funkcija koja prikazuje na ekranu vrednosti clanova strukture*/
32:     void stampa(struct student x)
33:     {       printf("\nStudent:\t%s\nSmer:\t\t%s", x.ime, x.smer);
34:             printf("\nBroj indeksa:\t%d", x.br_indeksa);
35:             printf("\nGodina upisa:\t%d\n\n", x.god_upisa);
36:     }
```

Анализа програма **primer4_4**

Програм у овом примеру формира структуру од података унетих преко тастатуре и приказује на екрану садржај формиране структуре. Ново код овог програма, у односу на програме претходних примера, је издвајање дела програма који приказује вредности чланова структуре у посебну функцију.

На почетку програма, ван сваке функције, дефинисан је (линије од 8. до 11.) тип структуре *student* (чланови су два низа карактера: *ime* и *smer* и две променљиве типа *int*: *br_indeksa* и *god_upisa*) и декларисана је функција *stampa()* (линија 13.).

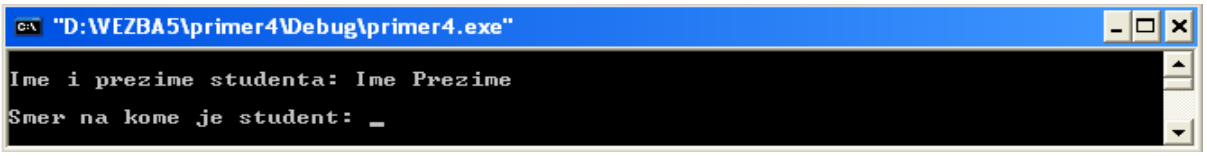
Из декларације функције може се видети да функција користи као аргумент структуру типа *student* (формално име структуре *x* овде није обавезно писати) и да не враћа вредност.

Функције примају структуру по вредности (због тога што је свака структура један објекат у програму) што значи да добијају само копије стварних чланова структуре и да на овај начин не могу променити садржај оригиналне структуре. Предавање копије целе структуре неће бити увек потребно.

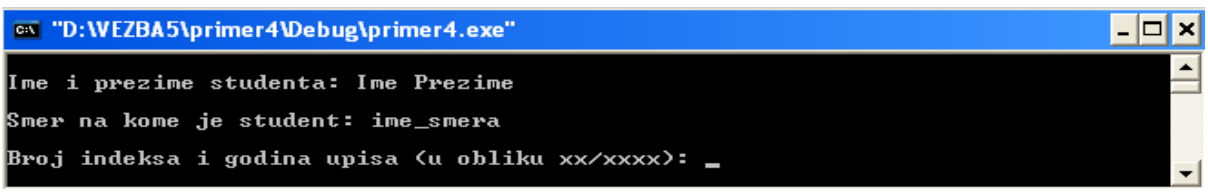
Из дефиниције функције *stampa()* (линије од 32. до 36.) може се видети да је приступ члановима структуре остварен помоћу оператора тачка. Са леве стране овог оператора пише се име структуре, формалног аргумента *(x)* а са десне стране име члана (линије од 33. до 35.). Функција користи стандардни начин за приказ на екрану вредности чланова структуре, формалног аргумента.

Главна функција програма има прво декларацију конкретне структуре типа *student,* која се зове *student1* (линија 16.). После тога тражи да се унесу преко тастатуре сви потребни подаци (слике 4.4 а), 4.4 б) и 4.4 в)), смешта ове податке у чланове структуре *student1* и на крају позива функцију *stampa()* да прикаже на екрану садржај структуре, чије је име написано на месту стварног аргумента ове функције (линија 27.). Резултат програма може се видети на слици 4.4 г).

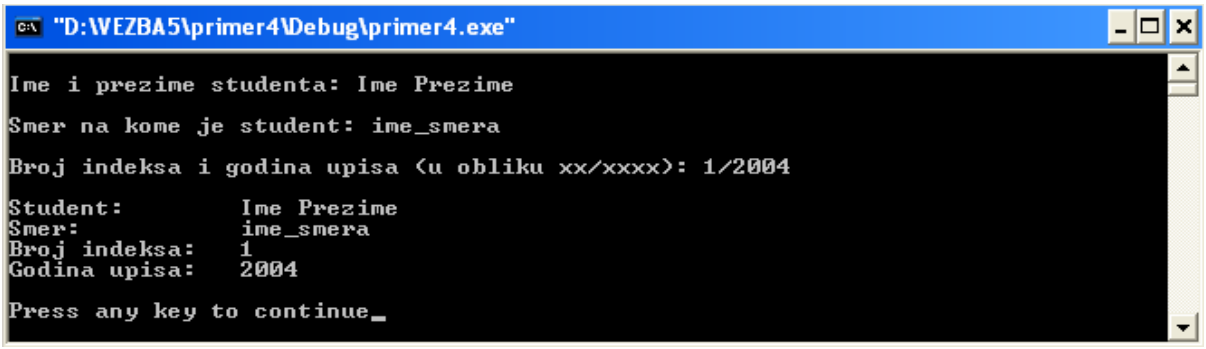
Слика 4.4 а) Излаз програма **primer4_4** (први део)



Слика 4.4 б) Излаз програма **primer4_4** (други део)



Слика 4.4 в) Излаз програма **primer4_4** (трећи део)



Слика 4.4 г) Излаз програма **primer4_4** (четврти део)

Изворни кôд програма **primer4_5**

```
1:    /*primer4_5.c*/
2:    /*Formiranje strukture zadatih podataka (o studentima i njihovim*/
3:    /*ocenama) i prikaz na ekranu spiska studenata koji su polozili ispit*/
4:
5:    #include <stdio.h>
6:    #include <stdlib.h>
7:    #include <string.h>
8:
9:    #define BROJ                  100    /*dozvoljeni broj studenata*/
10:   #define MAX                   30     /*broj korisnih karaktera u stringu*/
11:   #define GRESKA_DODELE         0      /*status prve poruke greske*/
12:   #define GRESKA_PROMENE        1      /*status druge poruke greske*/
13:
14:   char *poruke_o_greskama[] =
15:   {     "\nNije uspela dinamicka dodela memorije za niz!\n",
16:         "\nNije uspela promena dinamicki dodeljene memorije za niz!\n"};
17:
18:   struct student{
19:         char ime[MAX+1], smer[MAX+1];
20:         int br_indeksa, god_upisa, ocena;
21:         };
22:
23:   void cita_spisak(struct student niz[], int n);
24:   int sazima_spisak(struct student niz[], int n);
25:   void stampa_spisak(struct student niz[], int n);
26:   void greska(int status);
27:
28:   main( )
29:   {     int n,n2;
30:         struct student *spisak;     /*Deklaracija pokazivaca na niz struktura*/
31:
32:         do
33:         {     printf("\nBroj studenata: ");
34:               scanf("%d", &n);
35:         }while(n<1 || n>BROJ);
```

```
36:
37:             if((spisak = malloc(n *sizeof(struct student)))==NULL)
38:                     greska(GRESKA_DODELE);
39:
40:             cita_spisak(spisak, n);
41:             n2 = sazima_spisak(spisak, n);
42:
43:             if((spisak = realloc(spisak, n2 * sizeof(struct student)))==NULL)
44:                     greska(GRESKA_PROMENE);
45:
46:             stampa_spisak(spisak, n2);
47:
48:             free(spisak);
49:
50:             return 0;
51:     }
52:     /*Funkcija koja cita podatke sa tastature i formira niz struktura*/
53:     void cita_spisak(struct student niz[], int n)
54:     {       int i, tmp;
55:
56:             for (i = 0; i <n; i++)
57:             {       tmp = getchar();
58:                     printf("\nIme i prezime studenta: ");
59:                     gets(niz[i].ime);
60:
61:                     printf("Smer na kome je student: ");
62:                     gets(niz[i].smer);
63:
64:                     printf("Broj indeksa i godina upisa (u obliku xx/xxxx): ");
65:                     scanf("%d/%d", &niz[i].br_indeksa, &niz[i].god_upisa);
66:
67:                     do
68:                     {       printf("Ocena: ");
69:                             scanf("%d", &niz[i].ocena);
70:                     }while(niz[i].ocena<5 || niz[i].ocena>10);
71:             }
72:     }
73:     /*Funkcija koja sazima zadati spisak (niz struktura) */
74:     /*na spisak studenata koji su polozili ispit*/
75:     int sazima_spisak(struct student niz[], int n)
76:     {       struct student *ptr1, *ptr2;
77:
78:             for(ptr1=ptr2=niz; ptr1<niz+n; ptr1++)
79:                     if((*ptr1).ocena != 5)
80:                             *ptr2++ = *ptr1;
81:
82:             return (ptr2 - niz);
83:     }
84:
```

```
85:   /*Funkcija koja prikazuje na ekranu rezultujuci spisak*/
86:   void stampa_spisak(struct student niz[], int n)
87:   {       int i;
88:
89:           puts("\nSpisak studenata koji su polozili ispit:");
90:           for (i = 0; i <n; i++)
91:           {       printf("\n%s %s ", niz[i].ime, niz[i].smer);
92:                   printf("%d %d ", niz[i].br_indeksa, niz[i].god_upisa);
93:                   printf("%d", niz[i].ocena);
94:           }
95:           printf("\n\n");
96:   }
97:   /*Funkcija koja prikazuje na ekranu poruke o greskama*/
98:   void greska(int status)
99:   {       puts(poruke_o_greskama[status]);
100:          exit(1);
101:  }
```

Анализа програма **primer4_5**

Програм у овом примеру формира списак од података о студентима и њиховим оценама (задатих преко тастатуре) и приказује на екрану списак само оних студената који су положили испит. Списак података о студентима је реализован као низ структура. Помоћу низова структура у програмима се може руковати базама података и по томе се види који је њихов значај у програмима.

У овом програму дефинисан је тип структуре *student* и његови чланови: низови карактера *ime* и *smer*, као и променљиве типа *int: br_indeksa, god_upisa* и *ocena* (линије од 18. до 21.).

После дефиниције типа структуре написане су декларације свих функција које се позивају из главне (линије од 23. до 26.).

Функција *cita_spisak()* користи као аргументе показивач на низ структура типа *student* (то се може видети из заграда [] написаних после имена низа) и број *n* структура у том низу. Ова функција нема повратну вредност због тога што само смешта податке учитане са тастатуре у чланове одговарајуће структуре у низу. Функција (дефиниција је написана у линијама од 53. до 72.) у једној *for* петљи за *n* студената чита: име и презиме студента, име смера, број индекса, годину уписа и оцену од 5 до 10 и смешта ове податке у одговарајуће чланове:

- за првог студента у списку: *niz[0].ime, niz[0].smer, niz[0].br_indeksa, niz[0].god_upisa, niz[0].ocena*,

...

- за последњег студента у списку: *niz[n-1].ime, niz[n-1].smer, niz[n-1].br_indeksa, niz[n-1].god_upisa, niz[n-1].ocena.*

Функција *sazima_spisak()* користи као аргументе показивач на низ структура типа *student* и број *n* структура у том низу. Ова функција (дефиниција је написана у линијама од 75. до 83.) од низа структура (са подацима о студентима и њиховим оценама), чији је показивач и дужину добила као аргументе, формира низ само од оних структура за које је вредност члана *ocena* различита од 5 (за студенте који су положили испит).

Сажимање низа се реализује по већ описаном алгоритму. Оно што је ново у овом примеру је запис: у изразима саме *for* петље (линије од од 78. до 80.) написани су изрази иницијализације и промене показивача. Оба показивача на структуру типа *student* (*ptr1* и *ptr2*) на почетку имају адресу прве структуре у низу (*niz*). Све док је адреса коју чува *ptr1* мања од адресе краја низа (*niz+n*), ако вредност члана *ocena* оне структуре на коју показује *ptr1* није једнака 5, та структура остаје у резултујућем низу структура (тако што се садржај на који показује *ptr1* додели садржају на коју показује *ptr2* и показивач *ptr2* се инкрементира за следећу структуру у резултујућем низу). Показивач *ptr1* се инкрементира након сваке итерације и тако омогућава пролаз кроз оригинални задати низ структура. Као резултат сажимања задатог низа структура ова функција враћа цео број: дужину новог низа (*ptr2 – niz*).

Функција *stampa_spisak()*, као и претходне две, користи као аргументе показивач на низ структура типа *student* и број *n* структура у том низу и у једној *for* петљи (линије од 90. до 94.) приказује на екрану вредности свих чланова низа структура. Од ове функције нема повратне вредности.

Функција *greska()* приказује на екрану поруке о евентуалним грешкама при динамичкој додели меморије и промени динамички додељене меморије, на већ описан начин.

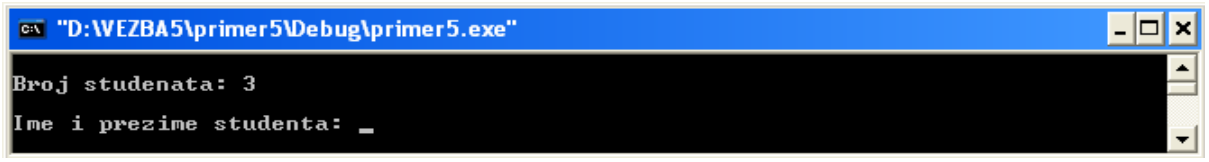
Главна функција на почетку има декларацију показивача на структуру. Уместо декларације низа структура (која би могла имати облик: *struct student spisak[MAX*]; за претходно дефинисани максимални број структура у низу *MAX*), да би што ефикасније била искоришћена меморија у програму је декларисан показивач (у линији 30.) на структуру типа *student,* који ће бити искоришћен као показивач на низ структура типа *student* (*struct student *spisak*;).

Када програм добије са тастатуре (слика 4.5 а)) број *n* студената у списку (то ће бити број структура у низу) може да тражи динамичку доделу меморије за низ од *n* структура типа *student* и ако има довољно меморије низ структура је смештен од адресе *spisak* (линије 37. и 38.).

Следе позиви функција *cita_spisak()* и *sazima_spisak()*. Свакој од ових функција као стварни аргументи предају се: адреса низа структура *spisak* и дужина низа структура *n* (линије 40. и 41.). Резултат друге наведене функције (дужина резултујућег низа добијеног после сажимања) додељује се променљивој *n2*. Због евентуално промењеног броја структура у низу нема потребе да до краја извршавања програма буде заузет сав простор резервисан на почетку и због тога следи (линије 43. и 44.) позив функције *realloc()* да се величина динамички додељене меморије од адресе *spisak* смањи на потребну (*n2 * sizeof(struct student)*). На крају главне функције је позив функције *stampa_spisak()* за стварне аргументе: адресу низа структура *spisak* и број структура у низу *n2*, као и ослобађање динамички додељене меморије (линије 46. и 48.).

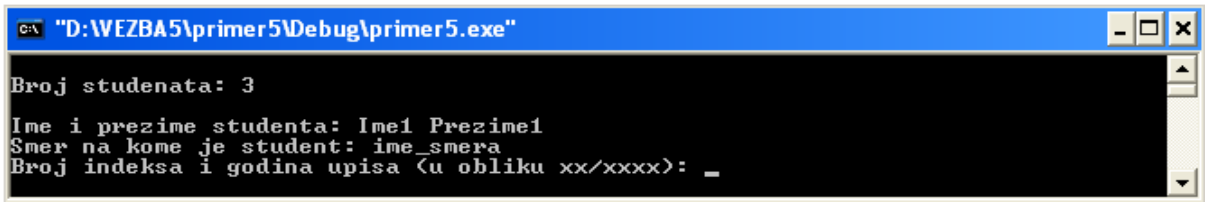
Резултат програма за задате комплете података (слике 4.5 б), 4.5 в), 4.5 г) и 4.5 д)) може се видети на слици 4.5 ђ).

Слика 4.5 а) Излаз програма **primer4_5** (први део)

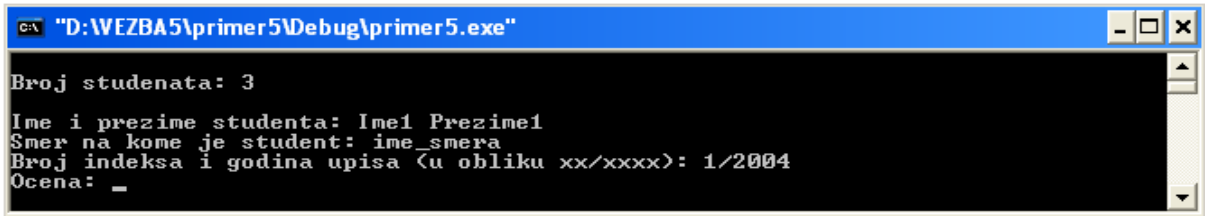


Слика 4.5 б) Излаз програма **primer4_5** (други део)

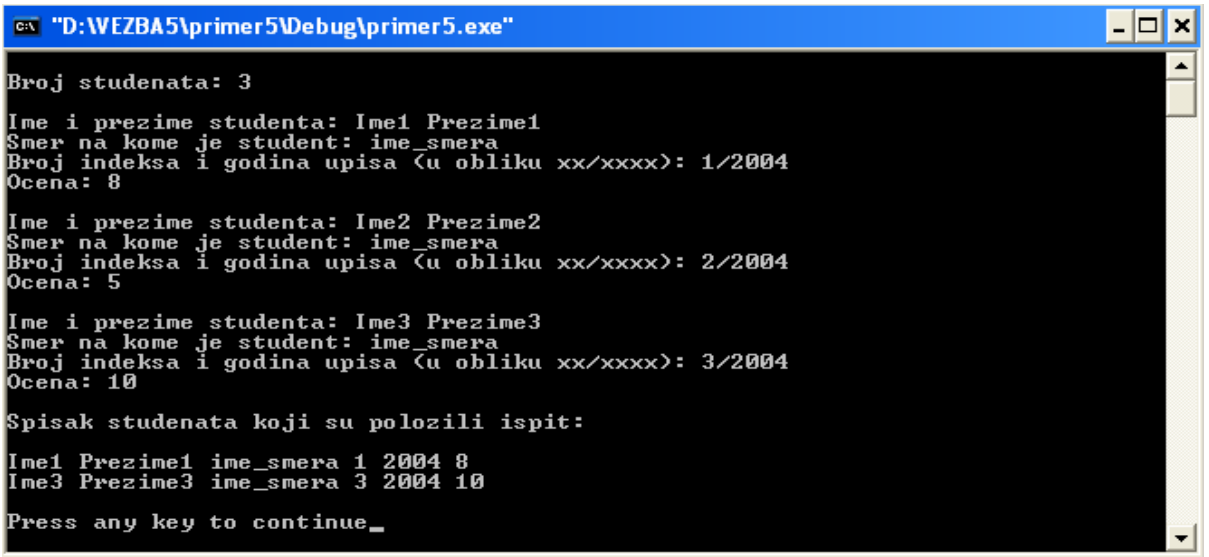
Слика 4.5 в) Излаз програма **primer4_5** (трећи део)



Слика 4.5 г) Излаз програма **primer4_5** (четврти део)



Слика 4.5 д) Излаз програма **primer4_5** (пети део)



Слика 4.5 ђ) Излаз програма **primer4_5** (шести део)

Изворни кôд програма **primer4_6∗**

```
1:      /∗primer4_6.c∗/
2:      /∗Kreiranje/pretrazivanje spiska zadatih adresa pomocu povezanih listi∗/
3:
4:      #include <stdio.h>
5:      #include <stdlib.h>
6:      #include <string.h>
7:      #define MAX     30                  /∗broj korisnih karaktera u stringu∗/
8:
9:      struct adresa{
10:             char prezime[MAX+1], ulica[MAX+1];
11:             int br_ulaza, br_stana;
12:             struct adresa ∗sledeca;
13:             };
14:
15:     void cita_adresu(struct adresa ∗ptr_adresa);
16:     void nalazi_adrese(struct adresa ∗ptr_adresa);
17:     void stampa_adresu(struct adresa ∗ptr_adresa);
18:
19:     main()
20:     {       char nastavak = 'D';
21:             struct adresa ∗prva, ∗tekuca;
22:
23:             prva = malloc(sizeof(struct adresa));
24:             tekuca = prva;
25:
26:             do
27:             {       if(tekuca == NULL)
28:                     {       fprintf(stderr, "Nije uspela dodela memorije!");
29:                             exit(1);
30:                     }
31:                     tekuca->sledeca = NULL;
32:
33:                     cita_adresu(tekuca);
34:
35:                     printf("\nIzabrati D za nastavak: ");
36:                     nastavak = getchar(); nastavak = getchar();
37:
38:                     if(nastavak == 'D')
39:                     {       tekuca->sledeca = malloc(sizeof(struct adresa));
40:                             tekuca = tekuca->sledeca;
41:                     }
42:             }while(nastavak == 'D');
43:
44:             nalazi_adrese(prva);
45:             free(prva);
46:             return 0;
47:     }
```

```
48:     void cita_adresu(struct adresa *ptr_adresa)
49:     {       puts("\nUneti prezime, ulicu, broj ulaza i broj stana:");
50:             scanf("%s%s%d%d",
51:                             ptr_adresa->prezime, ptr_adresa->ulica,
52:                             &ptr_adresa->br_ulaza, &ptr_adresa->br_stana);
53:     }
54:     void nalazi_adrese(struct adresa *ptr_adresa)
55:     {       int br_ulaza;
56:             char ulica[MAX+1];
57:
58:             printf("\nUneti ime ulice i broj ulaza: ");
59:             scanf("%s%d", ulica, &br_ulaza);
60:
61:             while(ptr_adresa != NULL)
62:             {       if(strcmp(      ptr_adresa->ulica, ulica)==0 &&
63:                                     ptr_adresa->br_ulaza == br_ulaza)
64:                             stampa_adresu(ptr_adresa);
65:
66:                     ptr_adresa = ptr_adresa->sledeca;
67:             }
68:     }
69:     void stampa_adresu(struct adresa *ptr_adresa)
70:     {       printf("%s %s %d %d\n",
71:                             ptr_adresa->prezime, ptr_adresa->ulica,
72:                             ptr_adresa->br_ulaza, ptr_adresa->br_stana);
73:     }
```

Анализа програма **primer4_6***

Када програм ради са низовима структура, дефинисање максималног броја структура (максималне дужине таквог низа) може бити проблем. Решење овог проблема је коришћење повезаних листи уместо низова структура.

Повезана листа је скуп структура у коме је свака структура истог типа и садржи међу члановима показивач на структуру тог истог типа. Овај показивач омогућава да се листа структура динамички мења, што значи да се структура по потреби: дода на почетак листе, дода на крај листе, уметне у листу између постојећих структура, избрише из листе... .

Овај програм од података са тастатуре, креира списак адреса (повезану листу структура), коју после претражује.

Програм у линијама од 4. до 6. укључује потребне библиотеке, а у линији 7. уводи потребну симболичку константу. У линијама од 9. до 13. програм садржи дефиницију структуре типа *adresa* (*prezime, ulica, br_ulaza, br_stana* и показивач **sledeca* на структуру истог типа), а у линијама од 15. до 17. декларацију функција: *cita_adresu(), nalazi_adrese()* и *stampa_adresu()* (свака од њих користи као аргумент показивач на структуру типа *adresa*).

Главна функција на почетку (линија 21.) декларише показиваче на структуру типа *adresa (*показивач **prva* биће искоришћен као показивач на прву структуру у листи, а показивач **tekuca* на текућу структуру у листи*)*. За сваку структуру у листи тражи се динамичка додела меморије. На почетку главна

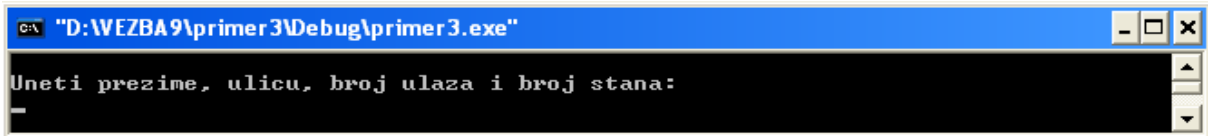
функција тражи динамичку доделу меморије за прву структуру у листи и проглашава је текућом у листи (линије 23. и 24.). Ако не успе додела меморије, следи прекид програма. У супротном, показивач који треба да је повезује са следећом у листи добија вредност *NULL* (нема следеће) (линије од 26. до 31.). Следи позив функције *cita_adresu()* за добијање са тастатуре вредности чланова текуће структуре у листи (функцији се предаје адреса текуће структуре). Ако се потврди наставак рада (линије од 38.) тражи се динамичка додела меморије за следећу структуру у листи и та структура се проглашава текућом у листи (линије 39. и 40.). Када се први пут не потврди наставак, програм излази из петље и позива функцију *nalazi_adrese()* (линија 44.) која налази и приказује на екрану садржаје структура са траженим вредностима.

Функција *cita_adresu()* користи као аргумент показивач *(*ptr_adresa*) на структуру типа *adresa.* Чита са тастатуре податке и додељује их члановима структуре: *ptr_adresa->prezime,*
ptr_adresa->ulica,
ptr_adresa->br_ulaza,
ptr_adresa->br_stana;

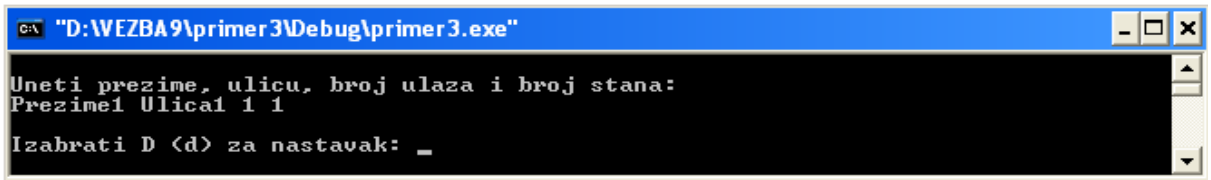
Функција *nalazi_adrese()* користи као аргумент показивач *(*ptr_adresa*). Ова функција тражи податке о имену улице и броју улаза и ако их нађе међу члановима структура у листи (претражује листу тако што вредност показивача *ptr_adresa* инкрементира на вредност *ptr_adresa->sledeca* да показује на следећу структуру), позива функцију *stampa_adresu()* (линије од 62. до 64.).

Функција *stampa_adresu()* приказује на екрану вредности свих чланова структуре чији је показивач (*ptr_adresa*) добила као аргумент.

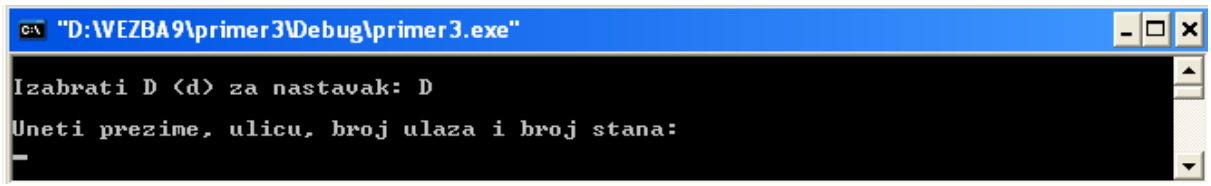
На сликама 4.6 а) до 4.6 ж) приказан је могући сценарио програма.



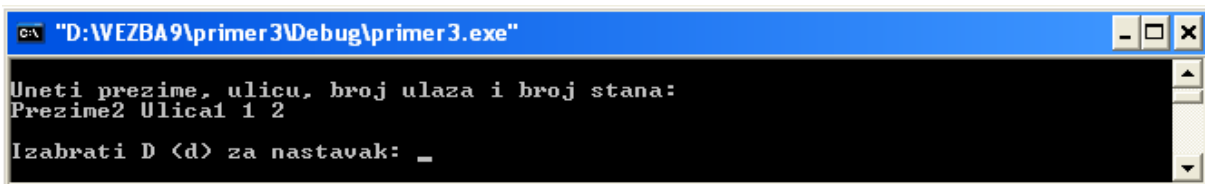
Слика 4.6 а) Излаз програма **primer4_6** (први део)



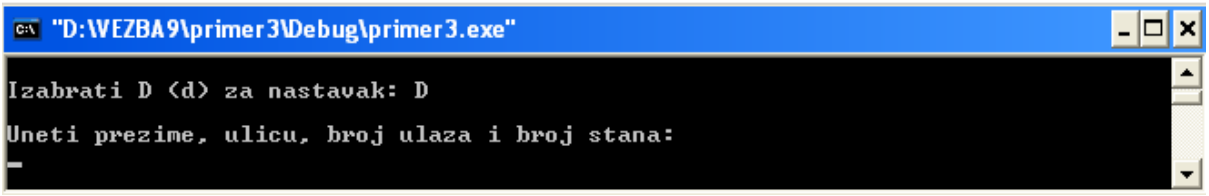
Слика 4.6 б) Излаз програма **primer4_6** (други део)



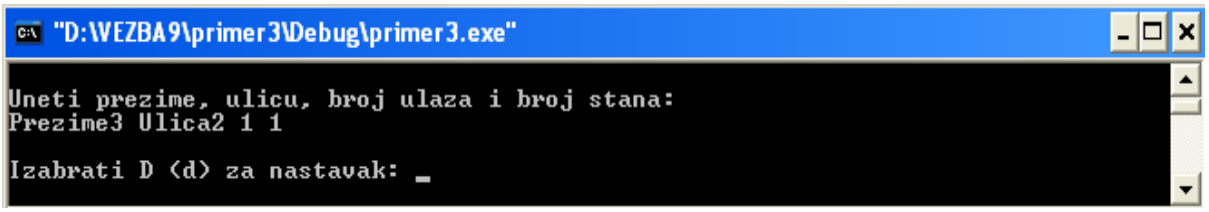
Слика 4.6 в) Излаз програма **primer4_6** (трећи део)



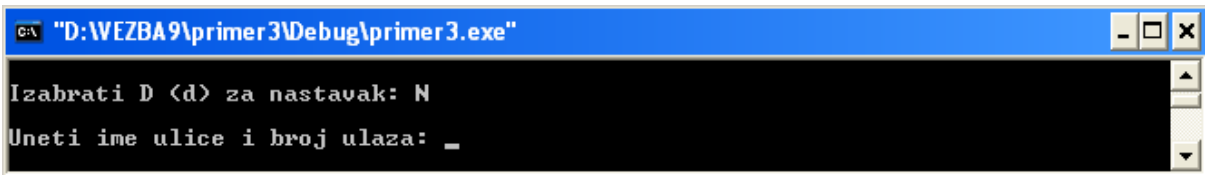
Слика 4.6 г) Излаз програма **primer4_6** (четврти део)



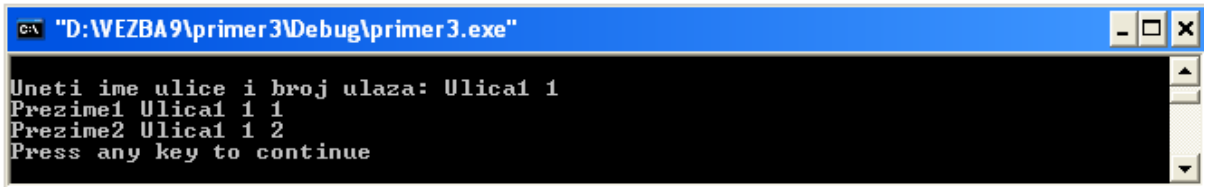
Слика 4.6 д) Излаз програма **primer4_6** (пети део)



Слика 4.6 ђ) Излаз програма **primer4_6** (шести део)



Слика 4.6 е) Излаз програма **primer4_6** (седми део)



Слика 4.6 ж) Излаз програма **primer4_6** (осми део)

Практични савети

- ✓ Дефиниције типова структура писати ван сваке функције због тога што се ове дефиниције издвајају у посебне датотеке модуларних програма.
- ✓ Службена реч *typedef* може бити корисна због краћег записа, али и смањити читљивост програма.
- ✓ У програмима који користе низове великог броја структура, оправдано је повезивање структура у листе, због ефикасног коришћења меморије.

Евентуалне грешке

- ➢ Структура, ма колико била сложена, сматра се једним објектом програма (за разлику од низа који је низ објеката).
- ➢ Структуре се, као и чланови структуре, прослеђују функцијама по вредности, а низови структура по референци.
- ➢ Код динамички повезаних листа не треба изоставити проверу резултата позива функције *malloc(),* као ни ослобађање динамички додељене меморије помоћу функције *free()*.

Задаци за припрему вежбе 4. у лабораторији

Задатак 1.

Написати програм на језику *C* који дефинише структуру типа *pravougaonik* (поменута структура треба да садржи структуру типа *tacka*) и на основу координата *x* и *y* доле-лево угла и страница (дужине и ширине) правоугаоника израчунава вредности осталих координата.

Задаци за вежбу 4. у лабораторији

Урадити следеће програме на језику *C* помоћу динамичке доделе, промене и ослобађања меморијског простора за структуре. Издвојити подзадатке програма у посебне функције.

Задатак 1.

Написати програм на језику *C* који креира списак смерова са следећим подацима: аутоматски генерисан redni_broj_smera и подаци учитани са тастатуре: *ime_smera* и *broj_studenata*. Списак може имати податке за *n* смерова *n* (*n≤10*). Програм даље треба да прихвати са тастатуре име смера и да прикаже на екрану број студената на том смеру.

Задатак 2.

Написати програм на језику *C* који креира списак са следећим подацима: *ime_studenta*, *ime_smera*, *br_indeksa* и *god_upisa.* Програм даље треба да прихвати са тастатуре податке за *n* студената (*n≤100*), а онда да прикаже на екрану све податке за студенте са постојећег списка, чија је година уписа до тражене године (задате преко тастатуре).

Задатак 3.

Написати програм на језику *C* који на основу података задатих преко тастатуре: имена (*ime_stanara)* за *n* (*n≤100*) станара, њихових адреса (за сваку адресу: *naziv_ulice*, *broj_ulaza*) креира списак и приказује на екрану имена свих станара у траженој улици (чији се назив задаје преко тастатуре). У случају да назив улице није на списку, програм треба да пријави на екрану одговарајућу поруку.

Задатак 4.*

Написати програм на језику *C* који, помоћу динамички повезане листе структура:

а) од података учитаних са тастатуре креира списак корисника разних мрежа мобилне телефије: *ime, adresa, poz_broj_mreze, broj_mob,*
б) приказује на екрану податке *ime* и *adresa* сваког корисника у креираном списку, чији *br_mob_telefona* почиње задатим brojem *poz_broj_mreze.*

Вежба 5.

Рад са датотекама у програмима на језику *C*

Текстуалне и бинарне датотеке

Отварање датотека

Позиционирање у датотекама

Размена података са датотекама

Затварање датотека

Примери

Изворни кôд програма **primer5_1**

```
1:      /*primer5_1.c*/
2:      /*Upis zadatog niza brojeva u tekstualnu datoteku*/
3:      /*i citanje sadrzaja datoteke iz iste funkcije*/
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #define MAX1        10          /*dozvoljena duzina niza*/
8:      #define MAX2        20          /*broj korisnih karaktera u stringu*/
9:
10:     typedef enum{GRESKA_DODELE, GRESKA_DAT}Greska;
11:
12:     char *poruke_o_greskama[] = {
13:             "\nGreska pri dinamickoj dodeli memorije!\n",
14:             "\nGreska pri otvaranju datoteke za upis!\n"      };
15:
16:     void greska(Greska status);
17:
18:     main()
19:     {       FILE *fptr;
20:             int i, n, *niz;
21:             char temp[MAX1], ime_dat[MAX2+1];
22:
23:             /*Formiranje niza*/
24:             do
25:             {       printf("\nDuzina niza (n <= %d): ", MAX1);
26:                     scanf("%d", &n);  fflush(stdin);
27:             }while(n<1 || n>MAX1);
28:
29:             if((niz = malloc(n*sizeof(int))) == NULL)
30:                     greska(GRESKA_DODELE);
31:
32:             printf("Vrednosti elemenata niza: ");
33:             for (i=0; i<n; i++)
34:                     scanf("%d", &niz[i]);
35:             fflush(stdin);
36:
37:             /*Otvaranje tekstualne datoteke za upis i citanje*/
38:             printf("Ime datoteke: ");
39:             gets(ime_dat);
40:
41:             if((fptr = fopen(ime_dat, "w+")) == NULL)
42:                     greska(GRESKA_DAT);
43:
```

```
44:             /*Upis u tekstualnu datoteku i prikaz na ekranu*/
45:             printf("\nUpisano u datoteku %s:", ime_dat);
46:             for (i=0; i<n; i++)
47:             {       fprintf(fptr, "\nniz[%d] = %d", i, niz[i]);
48:                     fprintf(stdout, "\nniz[%d] = %d", i, niz[i]);
49:             }
50:             rewind(fptr);           /*Povratak na pocetak datoteke*/
51:
52:             /*Citanje iz tekstualne datoteke i prikaz na ekranu*/
53:             printf("\n\nProcitano iz datoteke %s:", ime_dat);
54:              for(i=0; i<n; i++)
55:             {       fscanf(fptr, "%s%s%d", temp, temp, &niz[i]);
56:                     fprintf(stdout, "\nniz[%d] = %d", i, niz[i]);
57:             }
58:             printf("\n\n");
59:
60:             fclose(fptr);           /*Zatvaranje datoteke*/
61:             return 0;
62:     }
63:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
64:     void greska(Greska status)
65:     {       fprintf(stderr, poruke_o_greskama[status]);
66:             exit(1);
67:     }
```

Анализа програма **primer5_1**

За разлику од стандардних улазно/излазних токова за рад са тастатуром и екраном (*stdin, stdout, stderr*) за рад са сваком датотеком на диску потребно је отворити посебан улазно/излазни ток.

Датотека може бити отворена за упис, читање, или за обе ове операције и то у текстуалном или бинарном режиму. Све ово задаје се у време отварања улазно/излазног тока за размену података са датотеком (даље ће бити коришћен термин: отварање датотеке).

Функција за отварање датотеке *fopen()* дефинисана је у стандардној библиотеци *stdio.h* и враћа показивач. Када програм отвори датотеку повезује датотеку са структуром типа *FILE* (такође дефинисаном у библиотеци *stdio.h*) која садржи основне информације о датотеци. Функција *fopen()* враћа показивач на структуру типа *FILE*. Овај показивач се користи при раду са датотекама, без обзира на облик и режим датотеке.

Програм овог примера уписује задати низ бројева у једну текстуалну датотеку, чита њен садржај и приказује га на екрану.

На почетку главне функције (од линије 19.) декларисани су: показивач *fptr* на структуру типа *FILE* (*FILE *fptr*;), показивач на низ целих бројева (*int *niz*;) и низ карактера за чување имена датотеке (*char ime_dat[MAX2+1]*).

Програм из главне функције (линије од 24. до 35.) тражи дужину низа (слика 5.1 а)), динамички формира низ задате дужине, иницијализује низ на вредности задате преко тастатуре (слика 5.1 б)) и тражи име датотеке (слика 5.1 в)). После учитаног имена датотеке (линије 38. и 39.), програм из линије 41.

позива стандардну функцију *fopen()* за отварање текстуалне датотеке (подразумевано). Први аргумент ове функције је показивач на низ са именом датотеке (*ime_dat)*, а други показивач на низ са режимом датотеке за упис и читање после уписа ("w+").

Резултат функције *fopen()* додељује се показивачу *fptr* на структуру типа *FILE* придружену датотеци са задатим именом *(niz.txt)* и тестира се даље вредност овог показивача. Ако је то *NULL* значи да датотека није отворена у траженом режиму и у том случају предвиђен је позив функције *greska()* (линија 42.). У супротном приступа се истој датотеци за упис (од линије 45.).

Форматиран упис у датотеку реализује се помоћу функције *fprintf().* Једина разлика у односу на функцију *printf()* је у додатном аргументу, показивачу *fptr* (линија 47.). Функција *fprintf()* може да се искористи и за приказ података на екрану ако се уместо имена показивача придруженог датотеци (*fptr)* пише име главног стандардног излазног тока (*stdout*). Пример за ово је наредба у линији 48. Помоћу два описана позива функције *fprintf()* програм уписује у датотеку и приказује на екрану редом садржаје свих елемената задатог низа на следећи начин: име низа, индекс у заградама, оператор доделе и вредност која се додељује. За задате бројеве 1, 2 и 3, на пример, уписује:

niz[0] = 1
niz[1] = 2
niz[2] = 3

Читање садржаја датотеке од почетка, одмах после уписа, не би успело без наредбе *rewind(fptr).* Функција *rewind()* враћа текућу позицију, на коју показује *fptr,* на почетак датотеке (линија 50.)

Следи читање садржаја датотеке на коју показује *fptr* (линија 55.) и исписивање истог на екрану (линија 56.). Из сваког реда читају се по два низа карактера (сваки се смешта у променљиву *temp*) и цео број. (На приме: из првог реда датотеке у низ *temp* смешта се низ знакова *niz[0]*, а онда знак =, из другог реда датотеке низ знакова *niz[1]*, а онда знак =…).

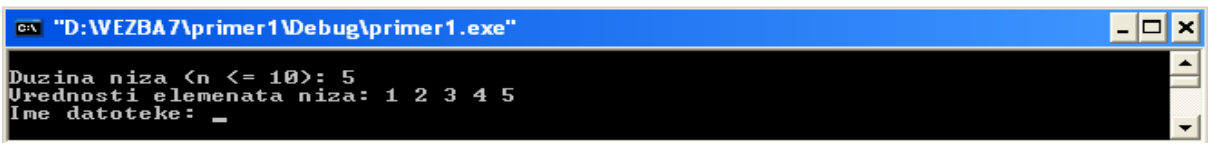
Функција *fprintf()* пружа велики избор контрола за форматирање (истих као функција *printf()*) и може се користити за упис табела и графика у датотеке.

Затварање улазно/излазног тока за датотеку којој је придружен показивач *fptr* остварује се помоћу позива функције *fclose(fptr)* (линија 60.).

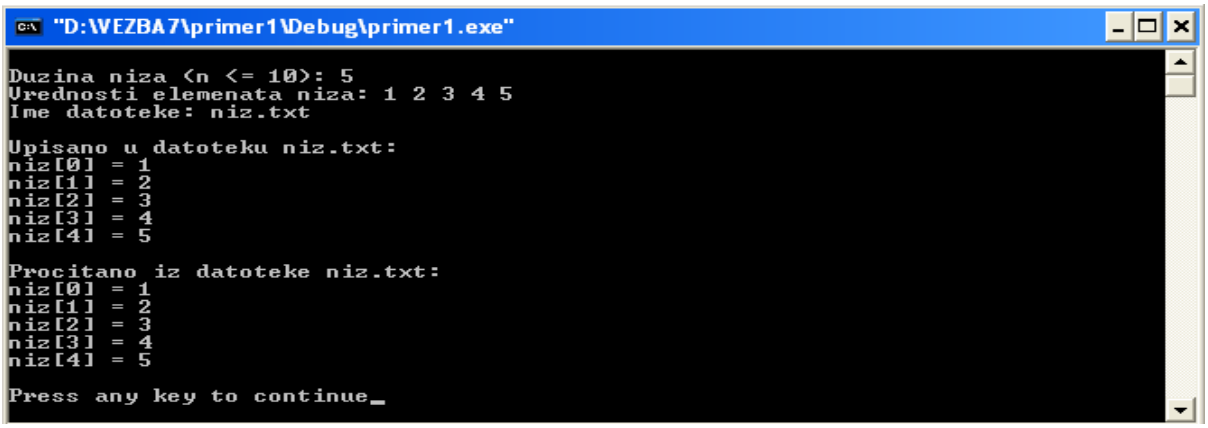
На слици 5.1 г) приказан је излаз програма на екрану, а на слици 5.1 д) преглед садржаја текст датотеке *niz.txt* (отворене на текућем директоријуму).

Слика 5.1 а) Излаз програма **primer5_1** (први део)

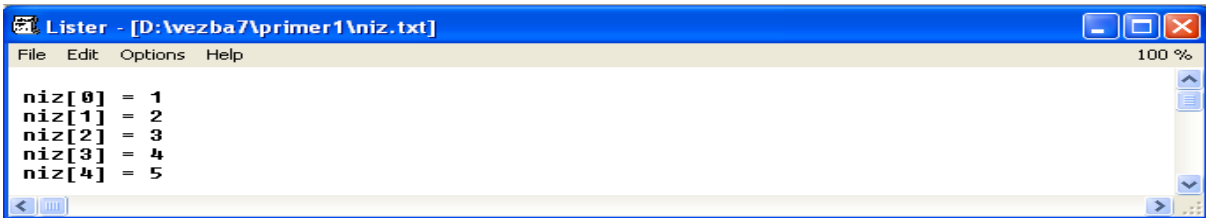
Слика 5.1 б) Излаз програма **primer5_1** (други део)



Слика 5.1 в) Излаз програма **primer5_1** (трећи део)



Слика 5.1 г) Излаз програма **primer5_1** (четврти део)



Слика 5.1 д) Излаз програма **primer5_1** (пети део)

Изворни кôд програма **primer5_2**

```
1:      /*primer5_2.c*/
2:      /*Upis zadatog znakovnog niza u tekstualnu datoteku, */
3:      /*prikaz na ekranu pojedinih delova ove datoteke, kao i*/
4:      /*vrednosti indikatora pozicije posle svakog citanja sadrzaja*/
5:
6:      #include <stdio.h>
7:      #include <stdlib.h>
8:      #define MAX     5               /*broj korisnih karaktera u stringu*/
9:
10:     typedef enum {GRESKA1_DAT,GRESKA2_DAT}Greska;
11:
12:     char *poruke_o_greskama[] = {
13:             "\nGreska pri otvaranju datoteke za upis!\n",
14:             "\nGreska pri otvaranju datoteke za citanje!\n"   };
15:
16:     void greska(Greska status);
17:
18:     main()
19:     {       FILE *fptr;
20:             char bafer[MAX+1];
21:             char poruka[] = "abcdefghijklmnopqrstuvwxyz";
22:
23:             /*Otvaranje datoteke, upis u datoteku i zatvaranje datoteke*/
24:             if ( (fptr = fopen("primer.txt", "w")) == NULL)
25:                     greska(GRESKA1_DAT);
26:
```

```
27:             fputs(poruka, fptr);
28:             fclose(fptr);
29:
30:             /*Otvaranje iste datoteke za citanje*/
31:             if ((fptr = fopen("primer.txt", "r")) == NULL)
32:                     greska(GRESKA2_DAT);
33:
34:             printf("\nPo otvaranju datoteke detektuje se pozicija: %ld", ftell(fptr));
35:
36:             /*Citanje prvih 5 karaktera iz datoteke*/
37:             fgets(bafer, MAX+1, fptr);
38:             printf("\n\nProcitan sadrzaj: %s, ", bafer);
39:             printf("detektovana pozicija u datoteci: %ld", ftell(fptr));
40:
41:             /*Citanje sledecih 5 karaktera iz datoteke*/
42:             fgets(bafer, MAX+1, fptr);
43:             printf("\n\nProcitan sadrzaj: %s, ", bafer);
44:             printf("detektovana pozicija u datoteci: %ld", ftell(fptr));
45:
46:             /*Povratak na pocetak datoteke*/
47:             rewind(fptr);
48:             printf("\n\nNakon povratka na pocetak, ");
49:             printf("detektovana pozicija u datoteci: %ld", ftell(fptr));
50:
51:             /*Citanje prvih 5 karaktera iz datoteke i zatvaranje datoteke*/
52:             fgets(bafer, MAX+1, fptr);
53:             printf("\n\nProcitan sadrzaj od pocetka datoteke: %s\n\n", bafer);
54:             fclose(fptr);
55:
56:             return 0;
57:     }
58:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
59:     void greska(Greska status)
60:     {       fprintf(stderr, poruke_o_greskama[status]);
61:             exit(1);
62:     }
```

Анализа програма **primer5_2**

Програм у овом примеру уписује један знаковни низ у текстуалну датотеку, а онда чита поједине делове исте датотеке и приказује их на екрану.

Нова функција у овом примеру је *ftell()*, која враћа текућу позицију у датотеци, на коју показује аргумент *fptr*. Повратна вредност поменуте функције је број бајтова од почетка датотеке.

На почетку главне функције (линије 19., 20. и 21) написане су потребне декларације: показивача *fptr* на поменуту структуру типа *FILE*, низа карактера *bafer* и низа карактера *poruka*. Последњи низ карактера је и иницијализован унутар декларације (слова од a до z).

Први део овог програма (линије од 24. до 28.):

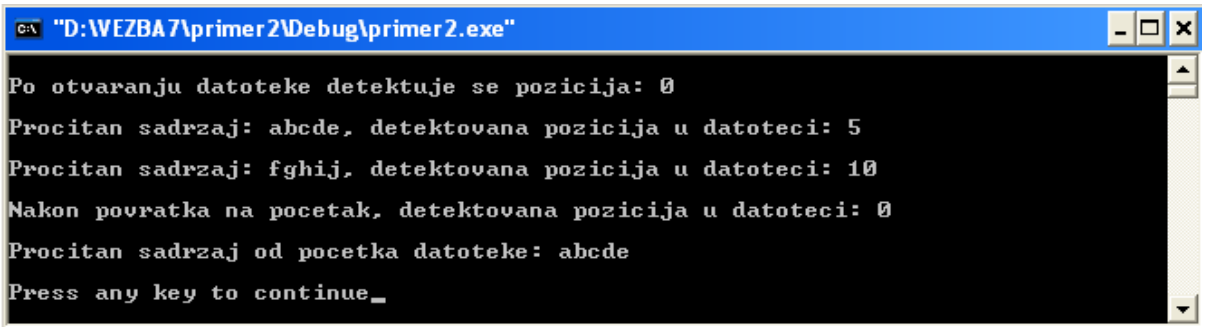
- отвара датотеку (*"primer.txt")* за упис *("w")* (линија 24.),

- уписује цео знаковни низ (слова од *a* до *z*) у отворену датотеку. За ово се користи функција *fputs()* која има додатни аргумент (показивач *fptr* на датотеку у коју се уписује) у односу на функцију *puts()* (линија 27.),
- затвара датотеку којој је придружен показивач *fptr* помоћу позива функције *fclose(fptr)* (линија 28.).

Други део овог програма (линије од 31. до 54.):

- отвара датотеку (*"primer.txt"*) за читање (*"r"*) (линија 31.),
- приказује на екрану текућу позицију у отвореној датотеци помоћу функције *ftell()* (линија 34.) и то је број 0 (0 бајтова од почетка),
- чита низ знакова од почетка датотеке помоћу функције *fgets()* која има два додатна аргумента (показивач *fptr* на датотеку из које се чита и максимални број знакова MAX) у односу на функцију *gets()* и смешта учитане знакове у низ карактера *bafer* (линија 37) а онда приказује на екрану садржај низа (линија 38.),
- приказује на екрану текућу позицију у отвореној датотеци помоћу функције *ftell()* (линија 39.) и то је број 5 (5 бајтова од почетка),
- чита следећи низ знакова из отворене датотеке помоћу функције *fgets()* и смешта учитане знакове у низ карактера *bafer* (линије 42.), а онда приказује на екрану садржај овог низа (линија 43.),
- приказује на екрану текућу позицију у отвореној датотеци помоћу функције *ftell()* (линија 44.) и то је број 10 (10 бајтова од почетка),
- враћа текућу позицију у отвореној датотеци на почетак датотеке помоћу функције *rewind()* (линија 47.),
- приказује на екрану текућу позицију у отвореној датотеци помоћу функције *ftell()* (линије 48. и 49.) и то је број 0 (0 бајтова од почетка, због тога што је текућа позиција враћена на почетак),
- чита низ знакова, опет од почетка датотеке, помоћу функције *fgets()* и смешта учитане знакове у низ каратера *bafer* (линија 52.), а онда приказује на екрану садржај овог низа (линија 53.),
- затвара отворену датотеку (линија 54.).

Све наведене детектоване позиције у датотеци могу се видети на слици 5.2 а), а садржај формиране текстуалне датотеке на слици 5.2 б).



Слика 5.2 а) Излаз програма **primer5_2** (први део)

Слика 5.2 б) Излаз програма **primer5_2** (други део)

Изворни кôд програма **primer5_3**

```
1:      /*primer5_3.c*/
2:      /*Kopiranje sadrzaja jedne tekstualne datoteke u drugu, */
3:      /*izmena imena datoteke i brisanje datoteke*/
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #define MAX  20              /*broj korisnih karaktera u stringu*/
8:
9:      typedef enum{
10:       GRESKA_KOPIRANJA, GRESKA_IMENA, GRESKA_BRISANJA}Greska;
11:
12:     char *poruke_o_greskama[] = {
13:             "\nGreska pri kopiranju datoteke!\n",
14:             "\nGreska pri promeni imena datoteke!\n",
15:             "\nGreska pri brisanju datoteke!\n"         };
16:
17:     int kopira_dat(char *ime1, char *ime2);
18:     void greska(Greska status);
19:
20:     main()
21:     {       char dat1[MAX+1], dat2[MAX+1];
22:
23:             /*Kopiranje sadrzaja iz jedne u drugu datoteku*/
24:             printf("\nIme originalne datoteke: ");
25:             gets(dat1);
26:             printf("Ime datoteke kopije: ");
27:             gets(dat2);
28:
29:             if(kopira_dat(dat1, dat2) == 0)
30:                     printf("Uspesno izvrseno kopiranje datoteke.\n");
31:             else
32:                     greska(GRESKA_KOPIRANJA);
33:
34:             /*Izmena imena datoteke*/
35:             printf("\nUnesite staro ime datoteke: ");
36:             gets(dat1);
37:             printf("Unesite novo ime datoteke: ");
38:             gets(dat2);
39:
40:             if (rename(dat1, dat2) == 0)
41:                     printf("Ime datoteke %s je promenjeno u ime %s.\n", dat1, dat2);
42:             else
43:                     greska(GRESKA_IMENA);
44:
45:             /*Brisanje datoteke*/
46:             printf("\nUnesite ime datoteke koju treba izbrisati: ");
47:             gets(dat1);
```

```
48:
49:             if (remove(dat1) == 0)
50:                     printf("Datoteka %s je izbrisana.\n\n", dat1);
51:             else
52:                     greska(GRESKA_BRISANJA);
53:
54:             return 0;
55:     }
56:     /*Funkcija koja kopira sadrzaj jedne tekst datoteke u drugu*/
57:     int kopira_dat( char *ime1, char *ime2)
58:     {       FILE *fptr1, *fptr2;
59:             int c;
60:
61:             if ( (fptr1 = fopen(ime1, "r") ) == NULL )
62:                     return -1;
63:             if ( (fptr2 = fopen(ime2, "w") ) == NULL  )
64:                     return -1;
65:
66:             while ((c=fgetc(fptr1)) !=EOF )
67:                     fputc(c, fptr2);
68:
69:             fclose(fptr2);
70:             fclose(fptr1);
71:
72:             return 0;
73:     }
74:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
75:     void greska(Greska status)
76:     {       fprintf(stderr, poruke_o_greskama[status]);
77:             exit(1);
78:     }
```

Анализа програма **primer5_3**

Програм у овом примеру обавља неке основне операције над датотекама: копира садржај једне датотеке у другу, мења име датотеке и брише датотеку. Имена потребних датотека програм тражи са тастатуре.

За копирање садржаја датотеке не постоји стандардна функција и због тога је у линији 17. декларисана функција *kopira_dat()*. Из декларације се може видети да функција користи два аргумента: показивач на низ карактера који чува име оригиналне датотеке и показивач на низ карактера који чува име датотеке копије. Функција враћа целобројну вредност остатку програма.

У линијама од 57. до 73. написана је дефиниција функције *kopira_dat()*. На почетку (линија 58.) декларисана су два показивача на структуру типа *FILE* (због тога што ће функција радити са две датотеке). Функција онда покушава да отвори две датотеке, једну за читање (“*r*”) и другу за упис (“*w*”) (линије од 61. до 64.). Ако не успе у отварању било које од датотека враћа вредност -1. У супротном иде даље и чита из прве датотеке један по један знак, помоћу функције *fgetc().* Сваки учитани знак, до знака за крај датотеке (*EOF*), функција *fputc()* ће приказати на екрану (линије 66. и 67.). Свака од претходно поменутих

функција има један додатни аргумент у односу на функције које размењују знак по знак са тастатуром и екраном (*getchar()* и *putchar()*) а то је показивач *fptr1* на датотеку. Следи затварање отворених датотека (линије 69. и 70.) и враћање вредности 0 остатку програма (знак да је функција извршила свој задатак).

На почетку главне функције (линија 21.) декларисани су низови карактера *dat1* и *dat2* за чување имена датотека. Главна функција тражи да се преко тастатуре унесу имена датотека (слике 5.3 а) и 5.3 б)) и смешта учитана имена у низове *dat1* и *dat2.* Програм предаје функцији *kopira_dat()* показиваче на низове са именима датотека и у зависности од резултата позване функције приказује одговарајућу поруку на екрану (слика 5.3 в)).

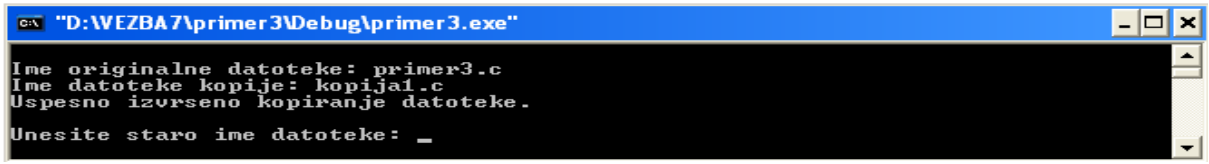
За остале предвиђене операције, измену имена и брисање датотеке, постоје стандардне функције.

Функција *rename()* мења име датотеке (линија 40.). Први аргумент ове функције је показивач на низ карактера који чува старо име датотеке, а други аргумент показивач на низ карактера који чува ново име датотеке (то могу бити и сама имена датотека написана између наводника). Ако функција врати вредност 0, то значи да је замена имена успела (слика 5.3 г)).

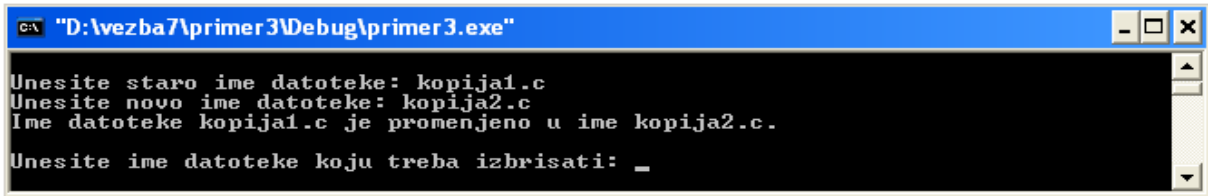
Функција *remove()* брише датотеку (линија 49.). Аргумент ове функције је показивач на низ карактера који чува име (то може бити и само име датотеке, написано између наводника). Ако функција врати вредност 0, брисање датотеке је успело (слика 5.3 д)).

Слика 5.3 а) Излаз програма **primer5_3** (први део)

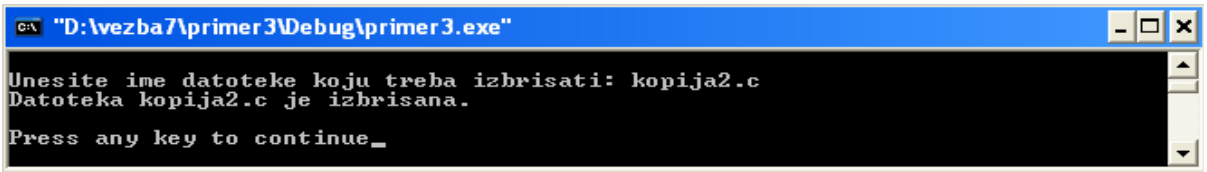
Слика 5.3 б) Излаз програма **primer5_3** (други део)



Слика 5.3 в) Излаз програма **primer5_3** (трећи део)



Слика 5.3 г) Излаз програма **primer5_3** (четврти део)



Слика 5.3 д) Излаз програма **primer5_3** (пети део)

Изворни кôд програма **primer5_4**

```
1:      /*primer5_4.c*/
2:      /*Upis zadatog niza brojeva u binarnu datoteku*/
3:      /*i citanje sadrzaja iste datoteke*/
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #define MAX     5               /*dozvoljena duzina niza*/
8:
9:      typedef enum{GRESKA1_DAT,GRESKA2_DAT}Greska;
10:
11:     char *poruke_o_greskama[] = {
12:             "\nGreska pri otvaranju datoteke za upis!\n",
13:             "\nGreska pri otvaranju datoteke za citanje!\n"   };
14:
15:     void greska(Greska status);
16:
17:     main()
18:     {       FILE *fptr;
19:             int i, niz1[MAX], niz2[MAX];
20:
21:             /*Inicijalizacija niza1*/
22:             for (i=0; i<MAX; i++)
23:                     niz1[i] = 2*i;
24:
25:             /*Otvaranje binarne datoteke za upis*/
26:             if((fptr = fopen("primer.bin", "wb")) == NULL)
27:                     greska(GRESKA1_DAT);
28:
29:             /*Upis vrednosti niza1 u datoteku i zatvaranje datoteke*/
30:             fwrite(niz1, sizeof(int), MAX, fptr);
31:             fclose(fptr);
32:
33:             /*Otvaranje iste datoteke za citanje*/
34:             if ( (fptr = fopen("primer.bin", "rb")) == NULL)
35:                     greska(GRESKA2_DAT);
36:
37:             /*Citanje podataka iz datoteke u niz2 i zatvaranje datoteke*/
38:             fread(niz2, sizeof(int), MAX, fptr);
39:             fclose(fptr);
40:
41:             /*Prikaz vrednosti elemenata upisanog i procitanog niza*/
42:             printf("\nUpis\tCitanje\n\n");
43:             for (i=0; i<MAX; i++)
44:                     printf("%d\t%d\n", niz1[i], niz2[i]);
45:             printf("\n");
46:
47:             return 0;
```

```
48:     }
49:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
50:     void greska(Greska status)
51:     {       fprintf(stderr, poruke_o_greskama[status]);
52:             exit(1);
53:     }
```

Анализа програма **primer5_4**

Програм у овом примеру размењује податке (низ целих бројева) са бинарном датотеком.

У линији 18. декларисан је показивач *fptr* на структуру типа *FILE* (исто као и код текстуалне датотеке).

Следи декларација два низа (линија 19.) и иницијализација једног од њих (линије 22. и 23.).

За отварање бинарне датотеке за упис (линија 26.) користи се иста функција као код текстуалне датотеке *fopen().* Први аргумент ове функције је показивач на низ са именом датотеке (само име датотеке, написано између наводника), а други аргумент је показивач на низ са режимом уписа *("wb"*).

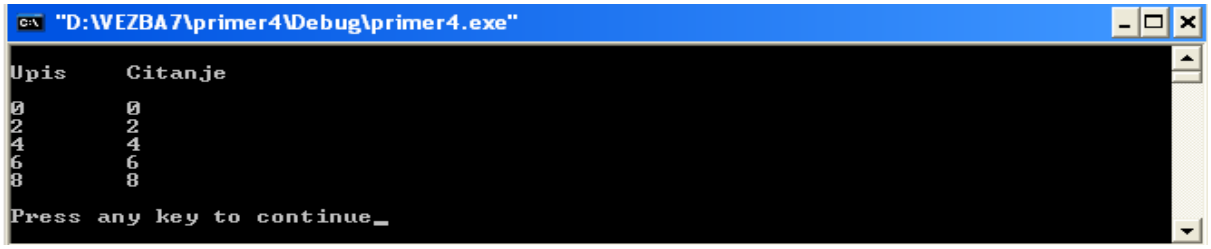
За упис података у бинарну датотеку користи се стандардна функција *fwrite()* (линија 30.). Ова функција уписује у бинарну датотеку тражени број меморијских блокова (сваки тражене величине) од тражене адресе у оперативној меморији. Адреса и величина једног меморијског блока, број блокова и показивач на датотеку, редом су аргументи функције *fwrite()*. У овом случају, *MAX* елемената од адресе *niz1* сваки величине *sizeof(int)* уписује се у датотеку на коју показује *fptr*.

За затварање датотеке, после обављене операције уписа, користи се функција *fclose()* (линија 31.), а за отварање исте датотеке у режиму за читање, функција *fopen()* (линија 34.).

За читање података из бинарне датотеке користи се стандардна функција *fread()* (линија 38.). Ова функција чита из бинарне датотеке тражени број меморијских блокова од тражене адресе у меморији. Адреса и величина једног меморијског блока, број блокова и показивач на датотеку редом су аргументи функције *fread()*. У овом случају, MAX елемената сваки величине *sizeof(int),* чита се из датотеке на коју показује *fptr* и уписује од адресе *niz2* у оперативној меморији.

За затварање датотеке, после обављене операције читања, користи се функција *fclose()* (линија 39.).

Из линија 43. и 44. програм приказује на екрану: низ целих бројева уписан у бинарну датотеку *(niz1)* и исти низ целих бројева, прочитан из бинарне датотеке *(niz2)* (слика 5.4).



Слика 5.4 Излаз програма **primer5_4**

Изворни кôд програма **primer5_5**

```
1:      /*primer5_5.c*/
2:      /*Citanje sadrzaja odredjenog dela binarne datoteke*/
3:      /*u koju je prethodno upisan zadati niz*/
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #define MAX     5               /*dozvoljena duzina niza*/
8:
9:      typedef enum{GRESKA1_DAT, GRESKA2_DAT, GRESKA_POZ} Greska;
10:
11:     char *poruke_o_greskama[] = {
12:             "\nGreska pri otvaranju datoteke za upis!\n",
13:             "\nGreska pri otvaranju datoteke za citanje!\n",
14:             "\nGreska pri pozicioniranju u datoteci\n"          };
15:
16:     void greska(Greska status);
17:
18:     main()
19:      {      FILE *fptr;
20:             int i, broj, rbroj, niz[MAX];
21:
22:             for (i=0; i<MAX; i++)
23:                     niz[i] = 10*i;
24:
25:             /*Otvaranje binarne datoteke za upis*/
26:             if ((fptr = fopen("primer.bin", "wb")) == NULL)
27:                     greska(GRESKA1_DAT);
28:
29:             /*Upis vrednosti niza u datoteku i zatvaranje datoteke*/
30:             fwrite(niz, sizeof(int), MAX, fptr);
31:             fclose(fptr);
32:
33:             printf("\nUpisano u datoteku primer.bin:\n");
34:             for (i=0; i<MAX; i++)
35:                     printf("\n%d. element: %d", i+1, niz[i]);
36:
37:             /*Otvaranje binarne datoteke za citanje*/
38:             if ((fptr = fopen("primer.bin", "rb")) == NULL)
39:                     greska(GRESKA2_DAT);
40:
41:             do
42:             {   printf("\n\nBirati redni broj elementa(od 1 do %d): ",MAX);
43:                 scanf("%d", &rbroj);
44:             }while(rbroj < 1 || rbroj > MAX);
45:
46:             /*Pomeranje indikatora pozicije na odredjeni element*/
47:              if ((fseek(fptr, (rbroj-1)*sizeof(int), SEEK_SET) ) != 0)
```

```
48:                     greska(GRESKA_POZ);
49:
50:             /*Citanje odredjenog elementa niza iz datoteke*/
51:             fread(&broj, sizeof(int), 1, fptr);
52:             printf("\n%d. element ima vrednost %d.\n\n", rbroj, broj);
53:
54:             fclose(fptr);
55:             return 0;
56:     }
57:     /*Funkcija koja prikazuje na ekranu poruke o greskama*/
58:     void greska(Greska status)
59:     {       fprintf(stderr, poruke_o_greskama[status]);
60:             exit(1);
61:     }
```

Анализа програма **primer5_5**

Програм у овом примеру уписује задати низ целих бројева у бинарну датотеку а онда чита из исте датотеке вредност елемента низа са траженим редним бројем.

У линији 19. декларисан је показивач *fptr* на структуру типа *FILE*, а у линији 20. низ целих бројева. Следи иницијализација низа (линије 22. и 23.).

У свом првом делу овај програм:

- отвара бинарну датотеку *("primer.bin")* за упис *(“wb”)* и приказује на екрану одговарајућу поруку ако није успело отварање датотеке (линије 26. и 27.),
- уписује претходно иницијализовани низ у отворену датотеку (линија 30),
- затвара датотеку (линија 31.),
- приказује на екрану (линије од 33. до 35.) редне бројеве и вредности елемената низа уписаног у датотеку (слика 7.5 а)).

У свом другом делу овај програм:

- отвара исту бинарну датотеку *("primer.bin")* за читање *(“rb”)* и приказује на екрану одговарајућу поруку ако није успело отварање датотеке (линије 38. и 39.),
- тражи да се унесе преко тастатуре редни број једног елемента низа (линије од 41. до 44.),
- помера помоћу функције *fseek()* текућу позицију у датотеци на почетак елемента са траженим редним бројем (линија 47.) и приказује на екрану одговарајућу поруку ако ова функција није извршила свој задатак (линија 48.),
- чита вредност елемента низа са траженим редним бројем (линија 51.) и приказује га на екрану (линија 52.),
- затвара датотеку (линија 54.).

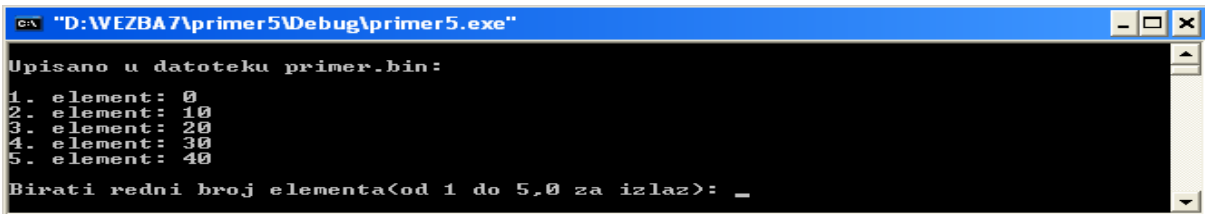
Функција *fseek(),* као и претходно описане функције *ftell()* и *rewind(),* припада групи стандардних функција за позиционирање у датотеци.

У претходним примерима биле су заступљене: функција *rewind()*, која премешта текућу позицију у датотеци на почетак датотеке и функција *ftell()*, која

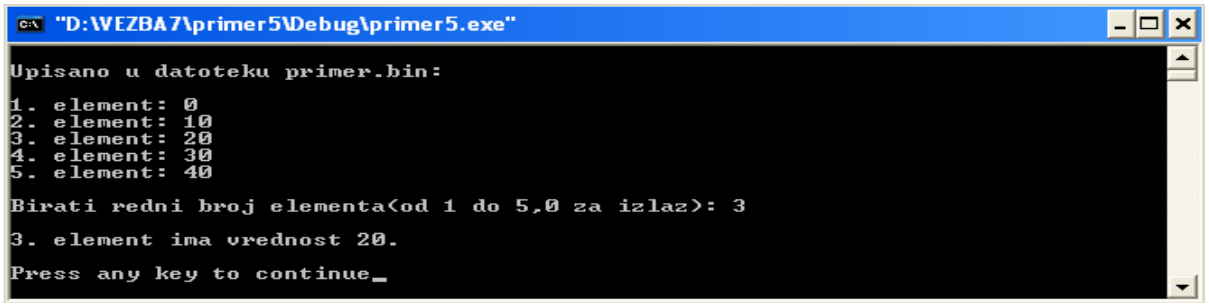
даје као резултат текућу позицију у датотеци (њен резултат углавном користи функција *fseek()*).

Функција *fseek()* позива се да промени текућу позицију у датотеци за одређени број бајтова. Постоје три унапред дефинисане константе које могу бити коришћене уместо бројева помераја: *SEEK_SET* (померај у односу на почетак датотеке), *SEEK_CUR* (померај у односу на текућу позицију) и *SEEK_END* (померај у односу на крај датотеке).

У овом примеру (линија 47.) функција *fseek()* добија задатак да промени текућу позицију у датотеци на коју показује *fptr*, од почетка датотеке (*SEEK_SET*) на почетак записа у датотеци који је вредност елемента са траженим редним бројем (*((rbroj-1)*sizeof(int)*). Приказ на екрану поруке од програма за задавање редног броја елемента и извештај о вредности траженог елемента могу се видети на сликама 5.5 а) и 5.5 б).



Слика 5.5 а) Излаз програма **primer5_5** (први део)



Слика 5.5 б) Излаз програма **primer5_5** (други део)

Изворни кôд програма **primer5_6∗**

```
1:      /∗primer5_6.c∗/
2:      /∗Formiranje i azuriranje sadrzaja binarne datoteke koja cuva zapise∗/
3:      /∗fiksne duzine: ime, prezime, smer, reg. broj i broj polozenih ispita∗/
4:
5:      #include <stdio.h>
6:      #include <stdlib.h>
7:      #include <string.h>
8:      #define MAX     20              /∗broj korisnih karaktera u stringu∗/
9:
10:     typedef enum{GRESKA1_DAT, GRESKA2_DAT, GRESKA_POZ} Greska;
11:
12:     char ∗poruke_o_greskama[] = {
13:             "\nGreska pri otvaranju datoteke za upis!\n",
14:             "\nGreska pri otvaranju datoteke za citanje!\n",
15:             "\nGreska pri pozicioniranju u datoteci\n"          };
```

```
16:
17:     FILE *fptr;
18:
19:     typedef struct student{
20:             char ime[MAX+1], prezime[MAX+1], smer[MAX+1], reg_broj[MAX+1];
21:             int br_pol_ispita;
22:             }Student;
23:     Student s1;
24:
25:     int broj_struktura = 0;
26:
27:     void unos(void);
28:     void izmene(void);
29:     void prikaz(void);
30:     void greska(Greska status);
31:
32:     main()
33:     {       int opcija, izlaz = 0;
34:
35:             unos();
36:
37:             do
38:             {       printf("\n\n1. Izmena\n2. Prikaz\n3. Izlaz");
39:                     printf("\n\nIzabrati opciju: ");
40:                     scanf("%d", &opcija);
41:
42:                     switch(opcija)
43:                     {       case 1:
44:                                             izmene();
45:                                             break;
46:                             case 2:
47:                                             prikaz();
48:                                             break;
49:                             default:
50:                                             izlaz = 1;
51:                                             break;
52:                     }
53:             }while(!izlaz);
54:
55:             return 0;
56:     }
57:     /*Funkcija koja upisuje sadrzaj u binarnu datoteku*/
58:     void unos(void)
59:     {       /*Otvaranje datoteke*/
60:             if((fptr = fopen("student.bin", "wb")) == NULL)
61:                     greska(GRESKA1_DAT);
62:
63:             /*Prihvatanje vrednosti clanova strukture i upis u datoteku*/
64:             printf("\nUneti za svakog studenta u posebnom redu ");
65:             printf("(do reci Kraj u posebnom redu):\nime i prezime, ");
```

```
66:             printf("ime smera, registarski broj i broj polozenih ispita\n\n");
67:
68:             while(1)
69:             {       scanf("%s", s1.ime);
70:                     if(strcmp(s1.ime, "Kraj")==0)
71:                             break;
72:
73:                     scanf("%s%s",s1.prezime, s1.smer);
74:                     scanf("%s%d",s1.reg_broj, &s1.br_pol_ispita);
75:
76:                     fwrite(&s1, sizeof(Student), 1, fptr);
77:
78:                     broj_struktura++;
79:             }
80:
81:             /*Zatvaranje datoteke*/
82:             fclose(fptr);
83:     }
84:     /*Funkcija koja azurira sadrzaj binarne datoteke*/
85:     void izmene(void)
86:     {       int rbroj, promena;
87:
88:             /*Otvaranje datoteke*/
89:             if ( (fptr = fopen("student.bin", "rb+")) == NULL)
90:                     greska(GRESKA2_DAT);
91:
92:             /*Izmena podataka u datoteci*/
93:             printf("\nUnesite redni broj studenta ");
94:             printf("i dodatni broj polozenih ispita: ");
95:
96:             do
97:             {       scanf("%d", &rbroj);
98:                     scanf("%d", &promena);
99:             }
100:            while(rbroj < 1 || rbroj > broj_struktura);
101:
102:
103:            if((fseek(fptr, (rbroj-1) *sizeof(Student),SEEK_SET))!=0)
104:                    greska(GRESKA_POZ);
105:
106:            fread(&s1, sizeof(Student), 1, fptr);
107:
108:            s1.br_pol_ispita += promena;
109:
110:            if((fseek(fptr, -sizeof(Student), SEEK_CUR)) != 0)
111:                    greska(GRESKA_POZ);
112:
113:            fwrite(&s1, sizeof(Student), 1, fptr);
114:
115:            /*Zatvaranje datoteke*/
```

```
116:            fclose(fptr);
117:    }
118:    /*Funkcija koja prikazuje sadrzaj binarne datoteke*/
119:    void prikaz(void)
120:    {       int i;
121:
122:            /*Otvaranje datoteke*/
123:            if((fptr = fopen("student.bin", "rb")) == NULL)
124:                    greska(GRESKA2_DAT);
125:
126:            /*Prikaz na ekranu sadrzaja datoteke*/
127:            for(i=0; i<broj_struktura;i++)
128:            {   fread(&s1, sizeof(Student), 1, fptr);
129:                printf("\n%d. %s %s %s ", i+1, s1.ime, s1.prezime, s1.smer);
130:                printf("%s %d ispita", s1.reg_broj, s1.br_pol_ispita);
131:            }
132:
133:            /*Zatvaranje datoteke*/
134:            fclose(fptr);
135:    }
136:    /*Funkcija koja prikazuje na ekranu poruke o greskama*/
137:    void greska(Greska status)
139:    {       fprintf(stderr, poruke_o_greskama[status]);
140:            exit(1);
141:    }
```

Анализа програма **primer5_6***

Програм у овом примеру формира и ажурира са тастатуре записе у бинарној датотеци који садрже податке о студентима. Сваки запис садржи име и презиме студента, име смера, регистарски број и број положених испита.

Ван сваке функције дефинисан је тип структуре *Student* који међу члановима има низове карактера: *ime, prezime, smer, reg_broj*, као и целобројну променљиву: *br_pol_ispita*. Декларисана је глобално и структура *s1* овог типа (линије од 19. до 23.).

Целобројна променљива *broj_struktura* глобално је декларисана и иницијализована на вредност 0 (линија 25.). Глобална декларација је написана због тога што ће већина функција програма користити ову променљиву.

У линијама од 27. до 29. написане су декларације функција: *unos(), izmene()* и *prikaz().* Свака од ових функција не користи аргументе и не враћа вредност остатку програма.

Функција *unos()* (дефиниција је написана од линије 58.):

- отвара бинарну датотеку *("student.bin"*) за упис *("wb")*, придружује јој показивач *fptr* и приказује на екрану одговарајућу поруку ако није успело отварање датотеке (линије 60. и 61.),
- тражи да се унесу преко тастатуре вредности чланова структуре (линије од 64.),
- у једној *while(1)* петљи (од линије 68.) прихвата са тастатуре податке и смешта их у одговарајуће чланове структуре *s1* (*s1.ime,*

s1.prezime, s1.smer, s1.reg_broj, s1.br_pol_ispita). Када члан структуре *s1.ime* добије реч *"Kraj"*, то значи крај извршавања петље. Све док није задата ова реч, у петљи се реализује упис садржаја структуре *s1* (од адресе *s1*, величине *sizeof(Student)*) у бинарну датотеку на коју показује показивач *fptr*. (линија 76.) и инкрементира се *broj_struktura* (линија 78.),

- затвара отворену датотеку (линија 82.).

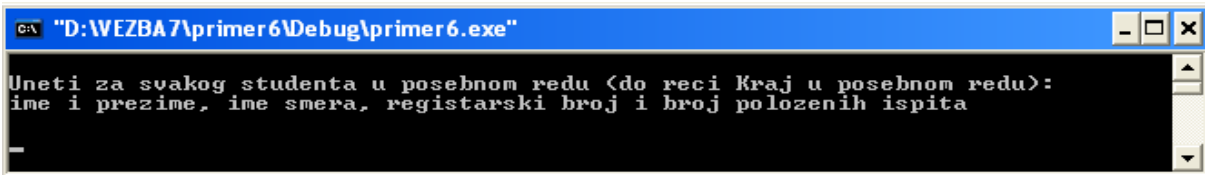
Функција *izmene()* (дефиниција је написана од линије 85.):

- отвара бинарну датотеку *("student.bin"*) за читање и упис *("rb+")*, придружује јој показивач *fptr* и приказује на екрану одговарајућу поруку ако није успело отварање датотеке (линије 89. и 90.),
- тражи да се унесу преко тастатуре редни број студента и додатни број положених испита (линије од 93. до 100.),
- помера помоћу функције *fseek()* текућу позицију у датотеци од почетка датотеке на почетак структуре у датотеци са траженим редним бројем (линија 103.) и приказује на екрану одговарајућу поруку ако ова функција није извршила свој задатак (линија 104.),
- чита садржај структуре са траженим редним бројем (линија 106.),
- прочитаној вредности *s1.br_pol_ispita* додаје нови број положених испита (линија 108.),
- помера помоћу функције *fseek()* текућу позицију у датотеци, за једну структуру уназад да би то опет био почетак структуре са траженим редним бројем (линија 110.) и приказује на екрану одговарајућу поруку ако ова функција није извршила свој задатак (линија 111.),
- уписује измењени садржај структуре у датотеку (линија 113.),
- затвара отворену датотеку (линија 116.).

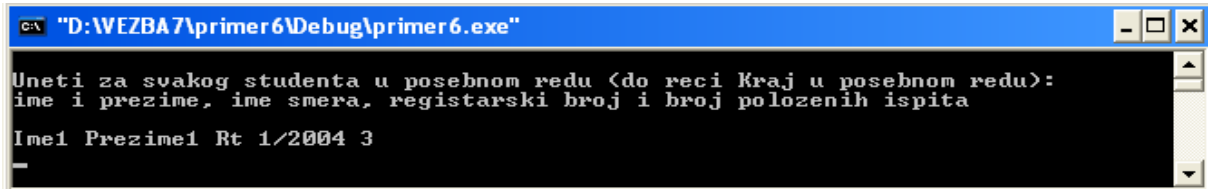
Функција *prikaz()* (дефиниција је написана од линије 119.):

- отвара бинарну датотеку *("student.bin"*) за читање *("rb")*, придружује јој показивач *fptr* и приказује на екрану одговарајућу поруку ако није успело отварање датотеке (линије 123. и 124.),
- у једној *for* петљи чита један по један запис (за сваку структуру у датотеци) и приказује редни број и садржај структуре на екрану (линије од 127. до 131.),
- затвара отворену датотеку (линија 134.).

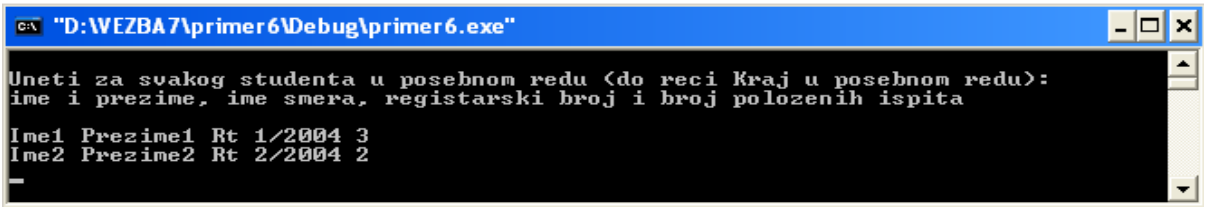
У главној функцији (од линије 32.) програм прво позива функцију *unos()* (линија 35.), да би били креирани записи у датотеци. У једној *do.. while* петљи (линије од 37. до 53.) приказује на екрану опције: *1. Izmena, 2. Prikaz, 3. Izlaz*. За изабрану опцију 1. позива функцију *izmene()*, која ажурира постојеће записе у датотеци. За изабрану опцију 2. позива функцију *prikaz(),* за преглед садржаја записа датотеке на екрану. За изабрану опцију 3. прекида извршавање текуће петље и то значи крај извршавања програма. На сликама 5.6 а) до 5.6 з) може се видети један могући сценарио извршавања програма.



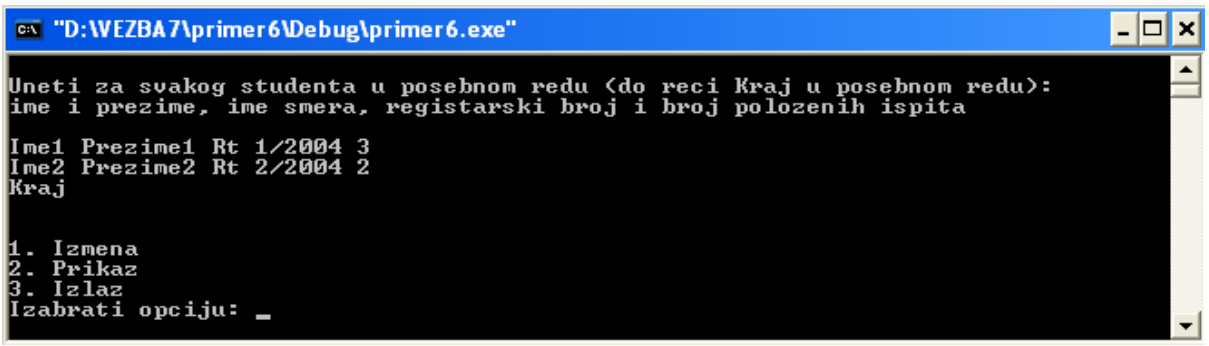
Слика 5.6 а) Излаз програма **primer5_6** (први део)



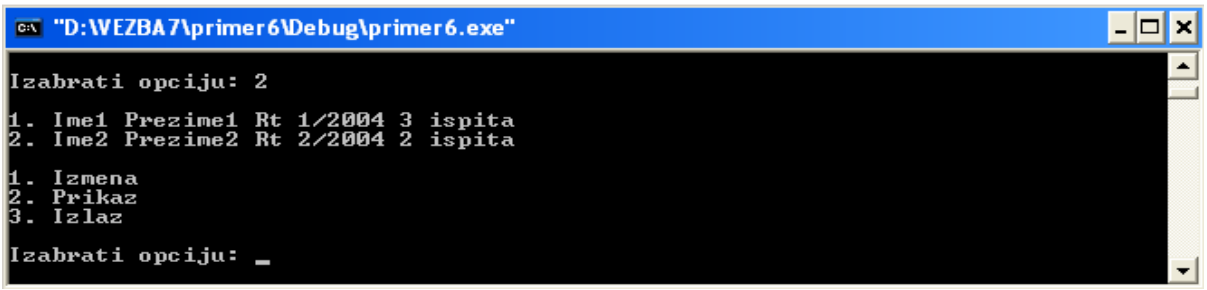
Слика 5.6 б) Излаз програма **primer5_6** (други део)



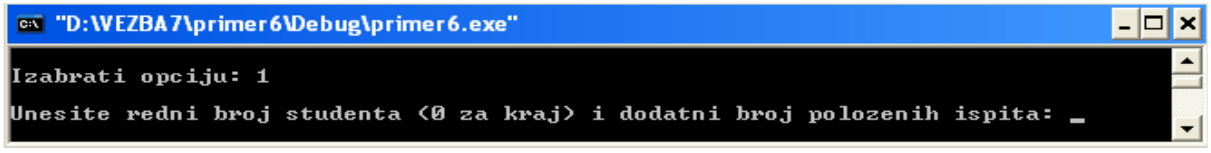
Слика 5.6 в) Излаз програма **primer5_6** (трећи део)



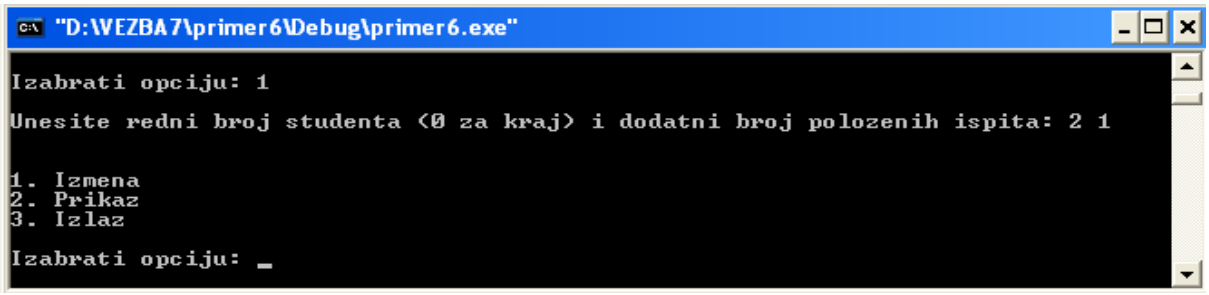
Слика 5.6 г) Излаз програма **primer5_6** (четврти део)



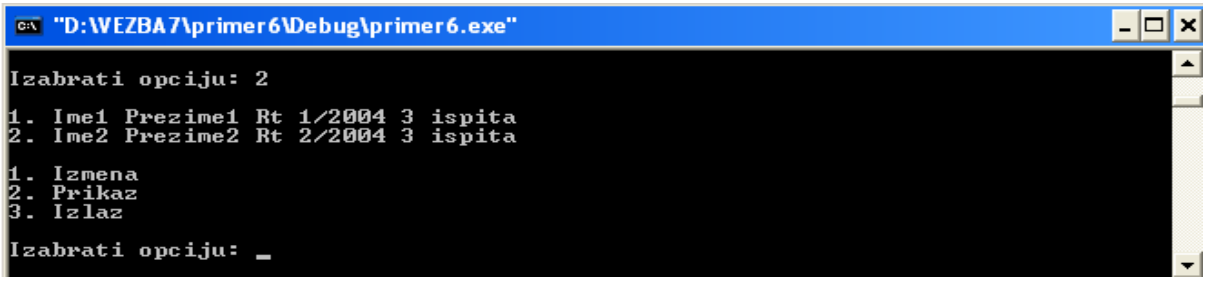
Слика 5.6 д) Излаз програма **primer5_6** (пети део)



Слика 5.6 ђ) Излаз програма **primer5_6** (шести део)



Слика 5.6 е) Излаз програма **primer5_6** (седми део)



Слика 5.6 ж) Излаз програма **primer5_6** (осми део)

Слика 5.6 з) Излаз програма **primer5_6** (девети део)

Практични савети

- ✓ За чување велике количине података предвидети бинарне датотеке (за исту количину података краћи је запис у овим датотекама у односу на запис у текстуалним).
- ✓ Само ако је потребно обезбедити преглед садржаја датотеке предвидети текстуалне датотеке. (Најчешће се користи комбинација бинарних датотека, које чувају све потребне податке и текстуалних, за онај део података који треба да буде прегледан).

Евентуалне грешке

- ➢ Не треба изоставити проверу резултата функције која се користи за отварање улазно/излазног тока за размену података са датотеком *(fopen())*. Ако се датотека отвара за упис, може се десити грешка ако није форматиран диск (на коме је предвиђен упис), или ако тражена путања за датотеку не постоји. Ако се датотека отвара за читање, може се десити грешка ако датотека са траженим именом или путањом не постоји. Поменута функција ће вратити *NULL* показивач ако није успела да изврши свој задатак.
- ➢ Не изоставити затварање сваког отвореног улазно/излазног тока за размену података са датотеком да последње измене у датотекама не би евентуално биле изгубљене.

Задаци за припрему вежбе 5. у лабораторији

Задатак 1.

Написати програм на језику *C* који чита садржај једне текстуалне датотеке и копира га у другу на следећи начин: сва велика слова текста конвертује у иста мала слова. Предвидети да имена датотека програм чита са тастатуре.

Задаци за вежбу 5. у лабораторији

Предвидети у следећим програмима да се имена свих потребних датотека прихватају са тастатуре.

Задатак 1.

Написати програм на језику *C* који чита садржај текстуалне датотеке и приказује на екрану извештај о дужини најдужег реда у овој датотеци.

Задатак 2.

Написати програм на језику *C* који чита садржај једне текстуалне датотеке и уписује овај садржај у другу текстуалну датотеку на следећи начин: сваку реч у посебном реду.

Задатак 3.

Написати програм на језику *C* који из датотеке са изворним кодом овог програма чита текст и уписује га у другу датотеку на следећи начин: на почетак сваког реда додаје редни број линије и знак двотачка (:).

Задатак 4.

Написати програм на језику *C* који

а) од задатих (преко тастатуре) *m (m≤10)* низова целих бројева, сваки исте дужине *n (n≤10),* формира *m* записа у бинарној датотеци,
б) разврстава записе ове датотеке у две бинарне датотеке тако да у једној буду низови чија је средња вредност већа од 0, а у другој сви остали низови,
в) приказује на екрану садржаје свих креираних датотека.

Задатак 5.*

Написати програм на језику *C* који

а) од задатих (преко тастатуре) података: *ime, prezime, br_indeksa, god_upisa, ocena*, за *n (n≤10)* студената формира *n* записа у бинарној датотеци,
б) формира другу бинарну датотеку од записа прве датотеке за оне студенте који су положили испит,
в) чита из друге датотеке и приказује на екрану оцену за задате податке: *br_indeksa* и *god_upisa.*

Вежба 6.

Модули и претпроцесорске директиве у програмима на језику *C*

Претпроцесорске директиве за контролу превођења изворног кода

Распоређивање изворног кода програма по модулима

Претпроцесорске директиве за израду макроа

Примери

Изворни кôд програма **primer6_1**

Копирање садржаја једне текст датотеке у другу, без речи *“greska”*

```
1:      /*direkt.h*/
2:      /*Datoteka_zaglavlje sa direktivama*/
3:
4:      #if !defined _direkt_h_
5:      #define _direkt_h_
6:
7:      #include <stdio.h>
8:      #include <stdlib.h>
9:      #include <string.h>
10:     #define MAX     20          /*broj korisnih karaktera u stringu*/
11:
12:     #endif
```

```
1:      /*global.h*/
2:      /*Datoteka_zaglavlje sa definicijama*/
3:
4:      #if !defined _global_h_
5:      #define _global_h_
6:
7:      #include "direkt.h"
8:
9:      typedef enum{NEMA_ARG, GRESKA1_DAT, GRESKA2_DAT}Greska;
10:     char *poruke_o_greskama[100] = {
11:             "\nZa pokretanje programa potrebno je zadati imena "
12:             "\ndatoteka u komandnoj liniji, posle imena programa!",
13:             "\nGreska pri otvaranju datoteke za citanje!\n",
14:             "\nGreska pri otvaranju datoteke za upis!\n"              };
15:     void greska(Greska);
16:
17:     #endif
```

```
1:      /*extern.h*/
2:      /*Datoteka_zaglavlje sa eksternim  definicijama*/
3:
4:      #if !defined _extern_h_
5:      #define _extern_h_
6:
7:      #include "direkt.h"
8:
9:      extern enum{NEMA_ARG, GRESKA1_DAT, GRESKA2_DAT}Greska;
10:     extern char *poruke_o_greskama[100];
11:     extern void greska(Greska);
12:
13:     #endif
```

```
1:      /*glavni.c*/
2:      /*Glavni modul: kopiranje sadrzaja jedne tekst datoteke u drugu*/
3:      /*bez reci "greska" (imena datoteka prihvataju se iz komandne linije)*/
4:
5:      #include "global.h"
6:
7:      main(int argc, char *argv[])
8:      {       char rec[MAX+1];
9:              FILE *fptr1, *fptr2;
10:
11:             if(argc < 3)
12:                     greska(NEMA_ARG);
13:
14:             if((fptr1 = fopen(argv[1], "r")) == NULL)
15:                     greska(GRESKA1_DAT);
16:
17:             if((fptr2 = fopen(argv[2], "w")) == NULL)
18:                     greska(GRESKA2_DAT);
19:
20:             while(1)
21:             {       fscanf(fptr1, "%s", rec);
22:
23:                     if(feof(fptr1))
24:                             break;
25:
26:                     if(strcmp(rec, "greska") != 0)
27:                             fprintf(fptr2, "%s ", rec);
28:             }
29:
30:             fclose(fptr2);
31:             fclose(fptr1);
32:
33:             return 0;
34:     }
```

```
1:      /*greska.c*/
2:      /*Modul sa funkcijom koja prikazuje na ekranu poruke  o greskama*/
3:
4:      #include "extern.h"
5:
6:      void greska(Greska)
7:      {       fprintf(stderr, poruke_o_greskama[Greska]);
8:              exit(1);
9:      }
```

Анализа програма **primer6_1**

Обиман изворни кôд програма распоређује се обично у више датотека (ово поједностављује писање, тестирање и одржавање програма). Све датотеке једног програма са екстензијом *.c* зову се модули програма *(source*

files). Модули једног програма укључују, помоћу директиве *#include,* датотеке са екстензијом *.h* које се зову датотеке заглавља *(header files).*

Овај програм има задатак да копира текст из једне текстуалне датотеке у другу, без задате речи *“greska”.*

Садржаји свих датотека заглавља могу се видети на почетку изворног кôда програма:

- *direkt.h* је датотека која садржи директиве *#include* за све библиотеке потребне програму и директиву *#define* за симболичке константе програма (поједине врсте директива, ако их има много, могу бити издвојене у посебне датотеке заглавља);
- *global.h* је датотека са глобалним дефиницијама типова (набројаних симбола) променљивих (низа показивача на поруке о грешкама) и функција *(greska())*;
- *extern.h* је датотека која има све дефиниције као и датотека *global.h,* разлика је у томе што су дефиниције написане као екстерне (са службеном речи *extern* испред дефиниције). Ово је због тога што укључивање глобалних дефиниција (датотеке *global.h*) у сваки модул преводилац не би дозволио (јер то значи понављање дефиниција). Овако датотеку *global.h,* која једина садржи дефиниције, укључује само један модул програма (у овом случају то је *glavni.c*), а датотеку *extern.h*, која садржи позиве на постојеће дефиниције, укључују сви остали модули програма.

Изворни кôд у свакој од поменутих датотека заглавља (погледати на пример датотеку *direkt_h*) почиње претпроцесорским директивама:

#if !defined

#define

и завршава се директивом:

#endif

Претпроцесорске директиве (као ове) пишу се тако да уоквирују кôд датотеке и елиминишу могућност вишеструког превођења истог дела кôда програма. Директиве написане на почетку, у случају да датотека заглавље *direкt.h* није већ укључена у модул програма *(#if !defined _direkt_h_*), омогућују да се укључи и да се преведе унутар модула део кôда у линијама од 7. до 10. *(#define _direkt_h_*). У супротном то неће бити могуће. Директива на крају *(#endif*) је обавезна код коришћења поменутих директива.

Изворни кôд програма распоређен је у два модула, *glavni.c* и *greska.c* (сами називи модула говоре о функцијама које садржи сваки од њих):

- *glavni.c* је модул који садржи главну функцију. Главна функција има аргументе и тражи из командне линије имена датотека (слика 6.1 а)). У главној функцији су прво декларисани показивачи на структуру типа *FILE* (*fptr1* и *fptr2*). Након провере да ли су у командној линији поред имена програма задата имена за две датотеке (линије 11. и 12.) позива се функција *fopen()* за отварање датотека. Прва, оригинална датотека (показивач на име је *argv[1]),* отвара се за читање (линије 14. и 15.), а друга, копија (показивач на име је *argv[2]),* за упис (линије 17. и 18.). Само копирање се реализује у једној *while(1)* петљи (линије од 20. до 28.). Садржај оригиналне датотеке чита се реч по реч (изузев речи *“greska”)* помоћу функције *fscanf()* и уписује реч по реч помоћу функције *fprintf().* Прекид петље обезбеђен је помоћу стандардне функције

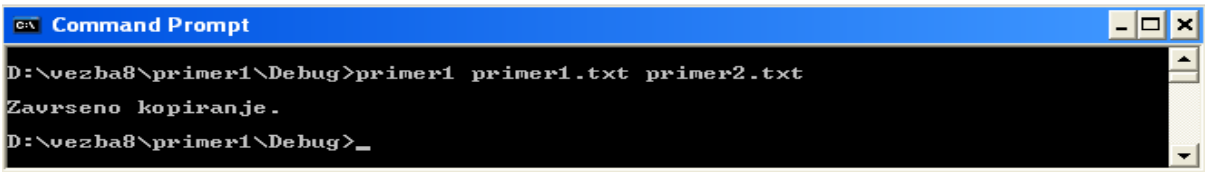
feof() (линија 23.). Ова функција користи аргумент показивач на датотеку и враћа вредност 0, све док текућа позиција у датотеци није на крају датотеке. Када региструје крај датотеке, враћа вредност 1 и то значи крај петље (линија 24.). На крају, главна функција затвара све отворене датотеке (линије 30. и 31.);

– *greska.c* је модул који нема пуно кода али је ту као пример модула у коме су све функције сличног типа (поруке о грешкама и осталим предвиђеним ситуацијама у програму). Функција је урађена на исти начин као и у примерима пре овог и неће бити даље анализирана.

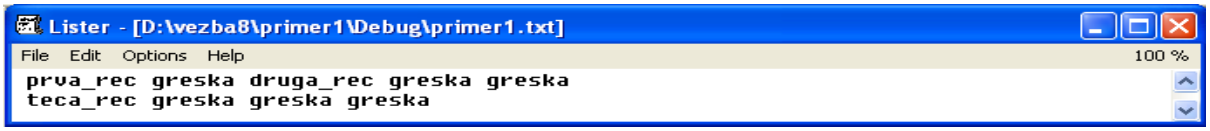
Један пар задатих имена за датотеке (из командне линије) и извештај о успешно завршеном копирању могу се видети на слици 6.1 б).

Садржај оригиналне датотеке, која је креирана пре извршавања програма, на директоријуму на коме се пушта програм, може се видети на слици 6.1 в), а садржај датотеке копије (све речи из прве датотеке копиране без речи *"greska")* на слици 6.1 г).

Слика 6.1 а) Излаз програма **primer6_1** (први део)



Слика 6.1 б) Излаз програма **primer6_1** (други део)



Слика 6.1 в) Излаз програма **primer6_1** (трећи део)

Слика 6.1 г) Излаз програма **primer6_1** (четврти део)

Изворни кôд програма **primer6_2**

Читање са тастатуре,ажурирање и приказ на екрану списка адреса

```
1:      /*direkt.h*/
2:      /*Datoteka_zaglavlje sa direktivama*/
3:
4:      #if !defined _direkt_h_
5:      #define _direkt_h_
6:
7:      #include <stdio.h>
8:      #include <stdlib.h>
9:      #include <string.h>
```

```
10:   #define BROJ        100    /*dozvoljeni broj adresa u spisku*/
11:   #define MAX         30     /*broj korisnih karaktera u stringu*/
12:
13:   #endif
```

```
1:    /*global.h*/
2:    /*Datoteka_zaglavlje sa definicijama*/
3:
4:    #if !defined _global_h_
5:    #define _global_h_
6:
7:    #include "direkt.h"
8:
9:    typedef enum{
10:         GRESKA_DODELE, GRESKA_PROMENE, OPCIJA_KRAJ}Greska;
11:
12:   char *poruke_o_greskama[] = {
13:         "\nNije uspela dinamicka dodela memorije za niz!\n",
14:         "\nNije uspela promena dinamicki dodeljene memorije za niz!\n",
15:         "\nIzabrana je opcija za kraj!\n"                              };
16:
17:   struct adresa{
18:         char prezime[MAX+1], ulica[MAX+1];
19:         int br_ulaza, br_stana;
20:         } *spisak;
21:
22:   int broj_adresa;
23:
24:   void cita_adresu(void);
25:   void dodaje_adresu(struct adresa x);
26:   void menja_adresu(void);
27:   void stampa_adrese(void);
28:   void greska(Greska);
29:
30:   #endif
```

```
1:    /*extern.h*/
2:    /*Datoteka_zaglavlje sa eksternim definicijama*/
3:
4:    #if !defined _extern_h_
5:    #define _extern_h_
6:
7:    #include "direkt.h"
8:
9:    extern enum{
10:         GRESKA_DODELE, GRESKA_PROMENE, OPCIJA_KRAJ}Greska;
11:
12:   extern char *poruke_o_greskama[BROJ];
13:
14:   extern struct adresa{
```

```
15:             char prezime[MAX+1], ulica[MAX+1];
16:             int br_ulaza, br_stana;
17:             } *spisak;
18:
19:     extern int broj_adresa;
20:
21:     extern void cita_adresu(void);
22:     extern void dodaje_adresu(struct adresa x);
23:     extern void menja_adresu(void);
24:     extern void stampa_adrese(void);
25:     extern void greska(Greska);
26:
27:     #endif
```

```
1:      /*glavni.c*/
2:      /*Glavni modul: glavna funkcija i pozivi ostalih funkcija*/
3:
4:      #include "global.h"
5:
6:      main( )
7:      {       int opcija;
8:
9:              broj_adresa = 0;
10:
11:             while(1)
12:             {       printf("\n1. Nova adresa");
13:                     printf("\n2. Izmena adrese");
14:                     printf("\n3. Prikaz");
15:                     printf("\n4. Izlaz");
16:
17:                     printf("\n\nIzabrati opciju: ");
18:                     scanf("%d", &opcija);
19:                     fflush(stdin);
20:
21:                     switch(opcija)
22:                     {       case 1:
23:                                     if(broj_adresa < BROJ)
24:                                             cita_adresu();
25:                                     break;
26:                             case 2:
27:                                     menja_adresu();
28:                                     break;
29:                             case 3:
30:                                     stampa_adrese();
31:                                     break;
32:                             case 4:
33:                                     greska(OPCIJA_KRAJ);
34:                                     break;
35:                             default:
36:                                     break;
```

```
37:                     }
38:             }
39:             if(broj_adresa > 0)
40:                     free(spisak);
41:
42:             return 0;
43:     }
```

```
1:      /*cita_adresu.c*/
2:      /*Modul sa funkcijom koja cita podatke o adresi sa tastature*/
3:      /*i  funkcijom koja dodaje adresu spisku*/
4:
5:      #include "extern.h"
6:
7:      /*Funkcija koja cita podatke o adresi sa tastature*/
8:      void cita_adresu(void)
9:      {       struct adresa adr1;
10:             int i, ista_adresa = 0;
11:
12:             printf("\nUneti prezime stanara, ime ulice, broj ulaza i broj stana:\n");
13:             scanf("%s%s", adr1.prezime, adr1.ulica);
14:             scanf("%d%d", &adr1.br_ulaza, &adr1.br_stana);
15:             fflush(stdin);
16:
17:             if(broj_adresa==0)
18:                     dodaje_adresu(adr1);
19:             else
20:             {       for(i=0; i<broj_adresa; i++)
21:                             if(strcmp(spisak[i].prezime, adr1.prezime)==0 &&
22:                               strcmp(spisak[i].ulica, adr1.ulica)==0 &&
23:                               spisak[i].br_ulaza == adr1.br_ulaza &&
24:                               spisak[i].br_stana == adr1.br_stana)
25:                                     ista_adresa = 1;
26:                     if(!ista_adresa)
27:                             dodaje_adresu(adr1);
28:             }
29:     }
30:     /*Funkcija koja dodaje adresu spisku*/
31:     void dodaje_adresu(struct adresa x)
32:     {       if(broj_adresa==0)
33:             {       if((spisak = malloc(sizeof(struct adresa))) == NULL)
34:                             greska(GRESKA_DODELE);
35:             }
36:             else if((spisak = realloc(     spisak, (broj_adresa+1) *
37:                                            sizeof(struct adresa))) == NULL)
38:                             greska(GRESKA_PROMENE);
39:
40:             spisak[broj_adresa] = x;
41:             broj_adresa++;
42:     }
```

```
1:      /*menja_adresu.c*/
2:      /*Modul sa funkcijom koja azurira podatke o adresi u spisku*/
3:
4:      #include "extern.h"
5:
6:      void menja_adresu(void)
7:      {       char ulica[MAX+1];
8:              int i, br_ulaza, br_stana;
9:
10:             printf("\nStara adresa (ulica, broj ulaza i broj stana):\n");
11:             scanf("%s%d%d", ulica, &br_ulaza, &br_stana);
12:
13:             for(i=0; i<broj_adresa; i++)
14:                     if(strcmp(spisak[i].ulica, ulica)==0 &&
15:                             spisak[i].br_ulaza==br_ulaza &&
16:                             spisak[i].br_stana==br_stana)
17:                     {       printf("\nNova adresa (ulica, broj ulaza i broj stana):\n");
18:                             scanf("%s%d%d", spisak[i].ulica,
19:                                     &spisak[i].br_ulaza,
20:                                     &spisak[i].br_stana);
21:                             fflush(stdin);
22:                     }
23:     }
```

```
1:      /*stampa_adrese.c*/
2:      /*Modul sa funkcijom koja prikazuje na ekranu spisak adresa*/
3:
4:      #include "extern.h"
5:
6:      void stampa_adrese(void)
7:      {       char ulica[MAX+1];
8:              int i, br_ulaza;
9:
10:             printf("\nIme ulice i broj ulaza:\n");
11:             scanf("%s%d", ulica, &br_ulaza);
12:             fflush(stdin);
13:
14:             printf("\nSpisak stanara na adresi %s %d:\n", ulica, br_ulaza);
15:             for(i=0; i<broj_adresa; i++)
16:                     if(strcmp(spisak[i].ulica,ulica)==0
17:                             && spisak[i].br_ulaza==br_ulaza)
18:                     {       printf("%s %s ", spisak[i].prezime,spisak[i].ulica);
19:                             printf("%d %d\n", spisak[i].br_ulaza, spisak[i].br_stana);
20:                     }
21:     }
```

```
1:      /*greska.c*/
2:      /*modul sa funkcijom koja prikazuje na ekranu poruke  o greskama*/
3:
4:      #include "extern.h"
```

```
5:
6:      void greska(Greska)
7:      {       fprintf(stderr, poruke_o_greskama[Greska]);
8:              exit(1);
9:      }
10:
```

Анализа програма **primer6_2**

Овај програм има задатак да чита са тастатуре, ажурира и приказује на екрану списак адреса (садржај низа структура, формиран преко тастатуре).

Садржаји свих датотека заглавља могу се видети на почетку изворног кôда програма:

- *direkt.h*, је датотека која садржи директиве *#include* за све библиотеке потребне програму и директиве *#define* за симболичке константе програма;
- *global.h* је датотека са глобалним дефиницијама, и она је укључена само у модул *glavni.c*. Ова датотека поред дефиниције типова (набројаних симбола), низа показивача на поруке о грешкама и декларације функције *greska(),* садржи дефиницију структуре типа *adresa* (*prezime, ulica, br_ulaza, br_stana*) и показивача на структуру истог типа *(*spisak),* дефиницију глобалне променљиве *broj_adresa*, као и декларацију функција: *cita_adresu(), dodaje_adresu(), menja_adresu(), stampa_adrese()* (једина функција од поменуте четири која користи аргумент је функција *dodaje_adresu()* и ни једна од функција не враћа вредност остатку програма);
- *extern.h* је датотека коју укључују сви модули сем главног. Ова датотека има све дефиниције као и датотека *global.h* (са службеном речи *extern* испред дефиниције).

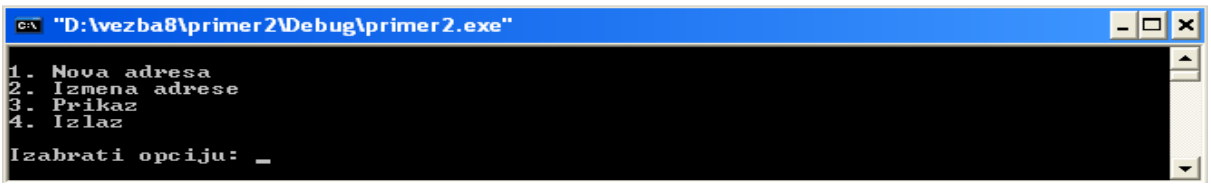
Изворни кôд програма распоређен је по модулима *glavni.c, cita_adresu.c, menja_adresu.c, stampa_adrese.c* и *greska.c* (сами називи модула говоре о функцијама које садржи сваки од њих):

- *glavni.c* је модул са главном функцијом која иницијализује broj_adresa на вредност 0 (линија 9.) и у *while(1)* петљи (линије од 11. до 38.) приказује на екрану опције: *1. Nova adresa, 2. Izmena adrese, 3. Prikaz, 4. Izlaz*. За изабрану опцију 1. и ако је вредност променљиве *broj_adresa* мања од највећег дозвољеног броја *(BROJ)* позива функцију *cita_adresu(),* за изабрану опцију 2. позива функцију *menja_adresu()* и за изабрану опцију 3. функцију *stampa_adrese().* За изабрану опцију 4. програм излази из петље и ако је у међувремену било динамичке доделе меморије ослобађа ту меморију (линије 39. и 40.);
- *cita_adresu.c* је модул који садржи функције *cita_adresu()* и *dodaje_adresu().* Прва поменута функција тражи да се унесу подаци о адреси и смешта ове податке у чланове структуре *adr1* типа *adresa*, формиране у овој функцији (линија од 9. до 15.). Ако нема адреса на списку (линија 17.) позива се функција *dodaje_adresu()* која креира списак и додаје му садржај структуре *adr1,* иницијализоване са тастатуре (линија 18.). У супротном

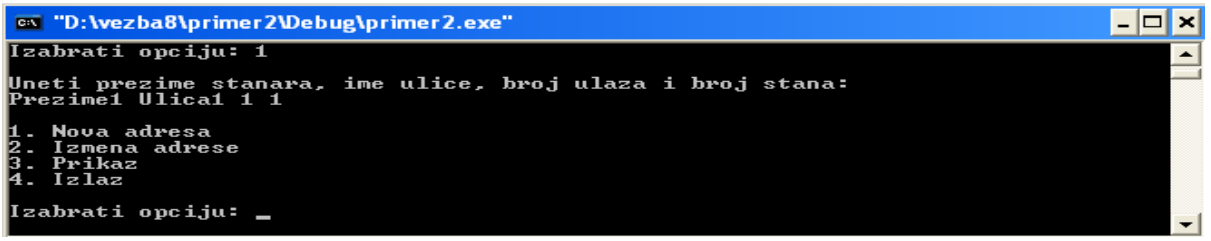
(линија 19.), ако исти подаци већ не постоје у некој од структура списка позива се функција *dodaje_adresu()* која додаје списку садржај структуре *adr1* (линије од 20. до 28.). Друга функција овог модула *dodaje_adresu()* добија од остатка програма садржај структуре типа *adresa*. Ако није било адреса на списку (линија 32.) тражи динамичку доделу за прву структуру типа *adresa* (линије 33. и 34.), у супротном тражи увећање динамички додељеног простора за још једну структуру типа *adresa* (линије од 36. до 38.). Следећа структура у списку добија садржај структуре, аргумента функције и инкрементира се број адреса у списку (линије 40. и 41);

- *menja_adresu.c* је модул који садржи само једну функцију *menja_adresu()*. Ова функција тражи да се унесу преко тастатуре подаци о постојећој адреси у списку (линије 10. и 11.) и ако нађе задату адресу у списку (постојећем низу структура) тражи са тастатуре податке о новој адреси (линије од 13. до 22.);
- *stampa_adrese.c* је модул који садржи само једну функцију *stampa_adrese()*. Ова функција чита са тастатуре следеће податке: име улице и број улаза (линије 10. и 11.), претражује списак адреса (постојећи низ структура) и приказује на екрану све адресе (садржаје свих структура) које имају тражене податке (линије од 14. до 20.);
- *greska.c* је модул који садржи познату функцију за приказ на екрану порука о грешкама.

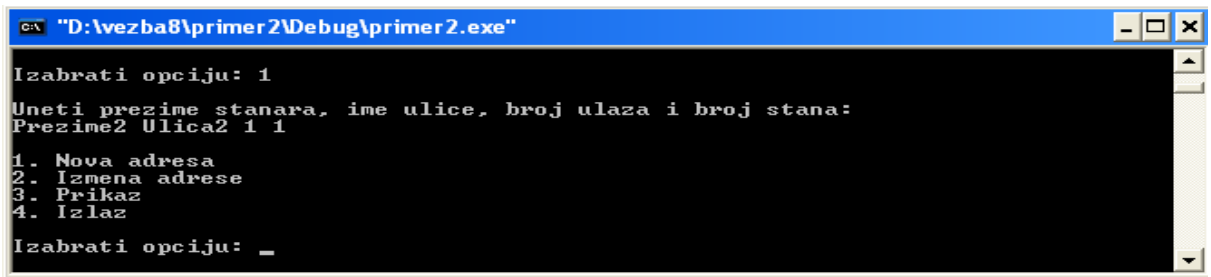
На сликама 6.2 а) до 6.2 ж) може се видети један могући сценарио извршавања програма.



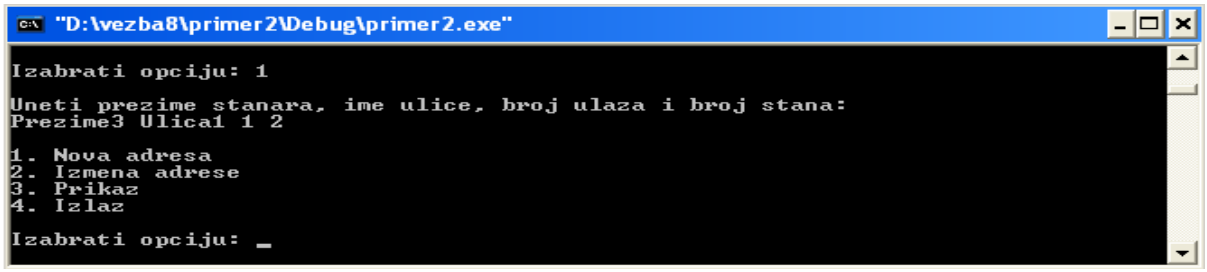
Слика 6.2 а) Излаз програма **primer6_2** (први део)



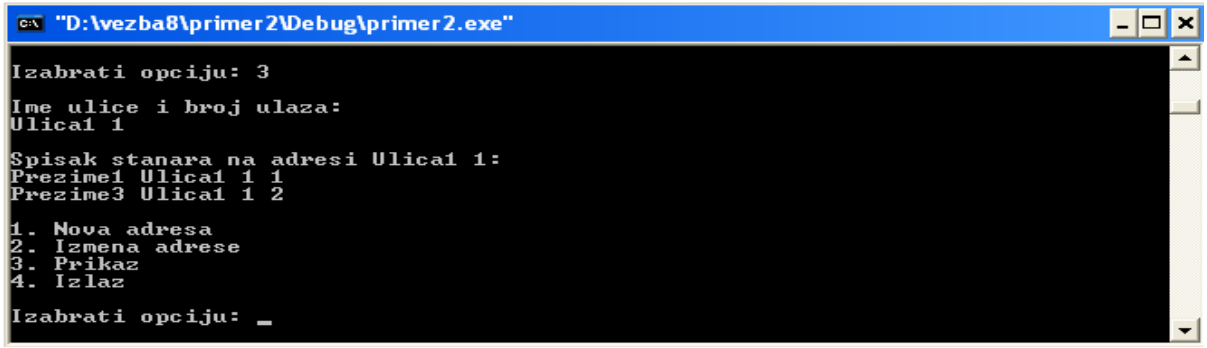
Слика 6.2 б) Излаз програма **primer6_2** (други део)



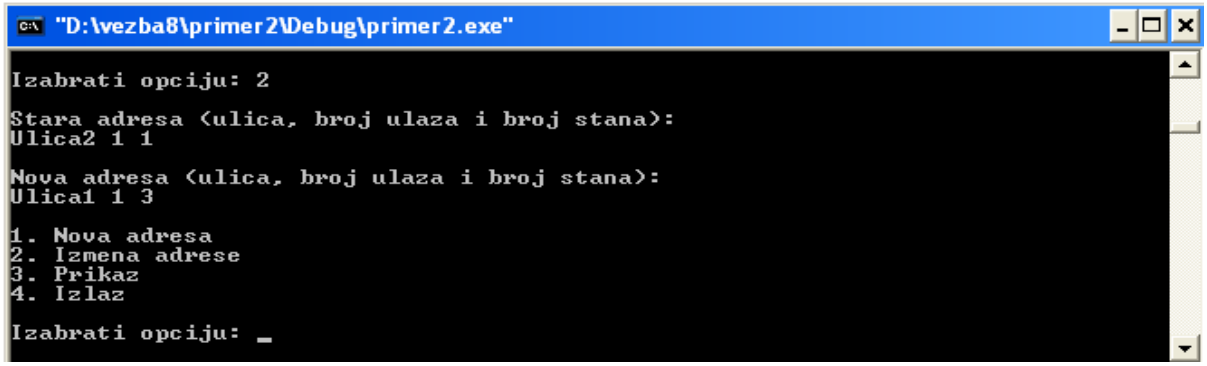
Слика 6.2 в) Излаз програма **primer6_2** (трећи део)



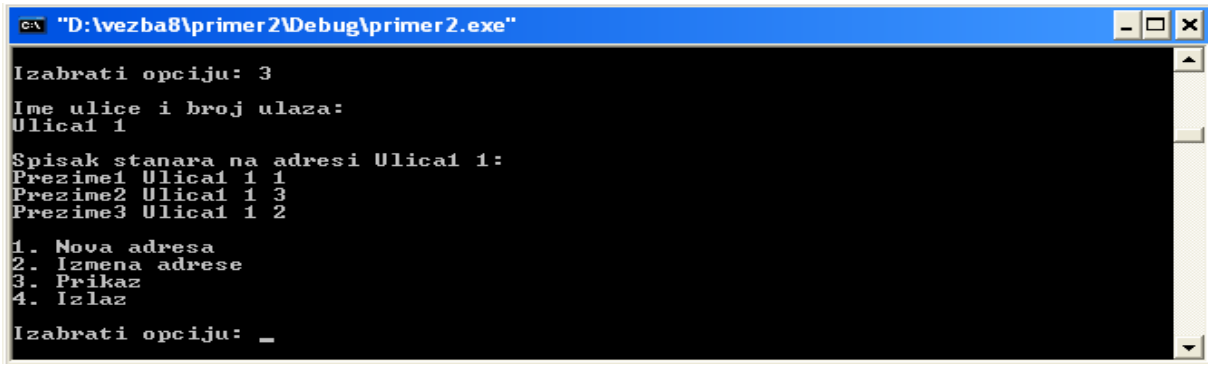
Слика 6.2 г) Излаз програма **primer6_2** (четврти део)



Слика 6.2 д) Излаз програма **primer6_2** (пети део)



Слика 6.2 ђ) Излаз програма **primer6_2** (шести део)



Слика 6.2 е) Излаз програма **primer6_2** (седми део)

Слика 6.2 ж) Излаз програма **primer6_2** (осми део)

Изворни кôд програма **primer6_3∗**

Читање са тастатуре података о испиту, упис у датотеку, ажурирање и приказ на екрану садржаја датотеке

```
1:      /∗direkt.h∗/
2:      /∗Datoteka zaglavlje sa direktivama∗/
3:
4:      #if !defined _direkt_h_
5:      #define _direkt_h_
6:
7:      #include <stdio.h>
8:      #include <stdlib.h>
9:      #define MAX         30          /∗broj korisnih karaktera u stringu∗/
10:     #define BROJ        100         /∗dozvoljeni broj struktura u nizu∗/
11:
12:     #endif
```

```
1:      /∗global.h∗/
2:      /∗Datoteka zaglavlje sa definicijama∗/
3:
4:      #if !defined _global_h_
5:      #define _global_h_
6:
7:      #include "direkt.h"
8:
9:      typedef enum{GRESKA1_DAT, GRESKA2_DAT}Greska;
10:
11:     char ∗poruke_o_greskama[] = {
12:             "\nGreska pri otvaranju datoteke za citanje!\n",
13:             "\nGreska pri otvaranju datoteke za upis!\n"      };
14:
15:     struct student{
16:             char ime[MAX+1];
17:             int reg_broj, god_upisa;
18:             };
19:     struct pol_ispiti{
20:             struct student student1;
21:             int broj_ispita;
22:             }polozio;
23:
24:     int broj_zapisa;
25:
26:     FILE ∗fptr;
27:
28:     void dod_dat(void);
29:     void azur_dat(void);
30:     void prikaz_dat(void);
31:     void greska(Greska);
32:
33:     #endif
```

```
1:      /*extern.h*/
2:      /*Datoteka zaglavlje sa eksternim  definicijama*/
3:
4:      #if !defined _extern_h_
5:      #define _extern_h_
6:      #include "direkt.h"
7:
8:      extern enum{GRESKA1_DAT, GRESKA2_DAT}Greska;
9:      extern char *poruke_o_greskama[BROJ];
10:
11:     extern struct student{
12:                     char ime[MAX+1];
13:                     int reg_broj, god_upisa;
14:                     };
15:     extern struct pol_ispiti{
16:                     struct student student1;
17:                     int broj_ispita;
18:                     }polozio;
19:     extern int broj_zapisa;
20:
21:     extern FILE *fptr;
22:
23:     extern void dod_dat(void);
24:     extern void azur_dat(void);
25:     extern void prikaz_dat(void);
26:     extern void greska(Greska);
27:
28:     #endif
```

```
1:      /*glavni.c*/
2:      /*Glavni modul: prikaz na ekranu menu-a i pozivi ostalih funkcija*/
3:
4:      #include "global.h"
5:
6:      main()
7:      {       int opcija, izlaz = 0;
8:              do
9:              {       printf("\n1. Dodavanje");
10:                     printf("\n2. Azuriranje");
11:                     printf("\n3. Prikaz");
12:                     printf("\n4. Izlaz");
13:
14:                     printf("\n\nIzabrati opciju: ");
15:                     scanf("%d", &opcija);
16:                     fflush(stdin);
17:
18:
19:                     switch(opcija)
20:                     {       case 1:
```

```
21:                                     dod_dat();
22:                                     break;
23:                             case 2:
24:                                     azur_dat();
25:                                     break;
26:                             case 3:
27:                                     prikaz_dat();
28:                                     break;
29:                             case 4:
30:                                     izlaz = 1;
31:                                     break;
32:                             default:
33:                                     break;
34:                     }
35:             }while(!izlaz);
36:
37:             return 0;
38:     }
```

```
1:      /*dod_dat.c*/
2:      /*Modul sa funkcijom koja datoteci dodaje zapise o studentima*/
3:
4:      #include "extern.h"
5:
6:      void dod_dat(void)
7:      {       int rbr;  struct pol_ispiti prazno = {0};
8:
9:              /*Datoteka sa zadatim imenom ne postoji*/
10:             if((fptr = fopen("ispiti.bin", "rb+")) == NULL)
11:             {       /*Datoteka sa zadatim imenom se kreira*/
12:                     if((fptr = fopen("ispiti.bin", "wb+")) == NULL)
13:                             greska(GRESKA1_DAT);
14:
15:                     broj_zapisa = 0;
16:                     fwrite(&broj_zapisa, sizeof(broj_zapisa), 1, fptr);
17:             }
18:             /*Datoteka postoji, citanje broja zapisa*/
19:             else
20:             {       fread(&broj_zapisa, sizeof(broj_zapisa), 1, fptr);
21:
22:                     /*Dodavanje zapisa*/
23:                     while(1)
24:                     {       printf("\nRedni broj studenta: ");
25:                             scanf("%d", &rbr);
26:                             fflush(stdin);
27:
28:                             if(rbr<1 || rbr>BROJ)  break;
29:
30:                             printf("Ime studenta: ");
31:                             gets(polozio.student1.ime);
```

```
32:
33:                             printf("Registarski broj i godina upisa: ");
34:                             scanf("%d%d",&polozio.student1.reg_broj,
35:                                             &polozio.student1.god_upisa);
36:                             fflush(stdin);
37:
38:                             polozio.broj_ispita = 0;
39:
40:                             /*Nov zapis iza kraja datoteke*/
41:                             if(rbr>broj_zapisa)
42:                             {       fseek(fptr, 0, SEEK_END);
43:
44:                                     while(++broj_zapisa<rbr)
45:                                             fwrite(&prazno, sizeof(prazno), 1, fptr);
46:
47:                                     broj_zapisa = rbr;
48:                             }
49:
50:                             /*Nov zapis pre kraja datoteke*/
51:                             else
52:                                     fseek(fptr,sizeof(broj_zapisa)+
53:                                             (rbr-1)*sizeof(polozio), SEEK_SET);
54:
55:                             /*Dodavanje zapisa*/
56:                             fwrite(&polozio, sizeof(polozio), 1, fptr);
57:
58:                             /*Promena broja zapisa*/
59:                             rewind(fptr);
60:                             fwrite(&broj_zapisa, sizeof broj_zapisa, 1, fptr);
61:                     }
62:             }
63:             fclose(fptr);
64:     }
```

```
1:      /*azur_dat.c*/
2:      /*Modul sa funkcijom koja azurira sadrzaj datoteke*/
3:
4:      #include "extern.h"
5:      void azur_dat(void)
6:      {       int rbr,promena;
7:
8:              /*Datoteka sa zadatim imenom ne postoji*/
9:              if((fptr = fopen("ispiti.bin", "rb+")) == NULL)
10:                     greska(GRESKA2_DAT);
11:
12:             /*Datoteka postoji*/
13:             fread(&broj_zapisa, sizeof(broj_zapisa), 1, fptr);
14:
15:             /*Azuriranje zapisa*/
16:             while(1)
```

```
17:             {       /*Citanje rednog broja studenta sa tastature*/
18:                     printf("\nRedni broj studenta: ");
19:                     scanf("%d", &rbr);
20:                     fflush(stdin);
21:
22:                     if(rbr<1 || rbr>BROJ)
23:                             break;
24:
25:                     if(rbr>broj_zapisa)
26:                     {       printf("\nPogresno zadat redni broj studenta!\n");
27:                             continue;
28:                     }
29:
30:                     /*Citanje podataka iz datoteke*/
31:                     fseek(fptr, sizeof(broj_zapisa) +
32:                                     (rbr-1)*sizeof(polozio), SEEK_SET);
33:                     fread(&polozio, sizeof(polozio), 1, fptr);
34:
35:                     /*Prikaz ucitanih podataka na ekranu*/
36:                     if(polozio.student1.ime[0])
37:                     {       printf(" %-20s %-5d %-5d %-3d\n",
38:                                     polozio.student1.ime,polozio.student1.reg_broj,
39:                                     polozio.student1.god_upisa, polozio.broj_ispita);
40:
41:                             /*Citanje novog podatka sa tastature*/
42:                             printf("\nBroj dodatno polozenih ispita: ");
43:                             scanf("%d", &promena);
44:                             fflush(stdin);
45:
46:                             polozio.broj_ispita += promena;
47:
48:                             /*Upis novih podataka u datoteku*/
49:                             fseek(fptr, -sizeof(polozio), SEEK_CUR);
50:                             fwrite(&polozio, sizeof(polozio), 1, fptr);
51:                     }
52:                     else
53:                             printf(" Ne postoji student sa ovim rednim brojem.\n");
54:             }
55:             fclose(fptr);
56:     }
```

```
1:      /*prikaz_dat.c*/
2:      /*Modul sa funkcijom koja prikazuje na ekranu sadrzaj datoteke*/
3:
4:      #include "extern.h"
5:
6:      void prikaz_dat(void)
7:      {       int rbr;
8:
9:              /*Otvaranje datoteke sa zadatim imenom*/
```

```
10:             fptr = fopen("ispiti.bin", "rb+");
11:
12:             /*Datoteka ne postoji*/
13:             if(fptr==NULL)
14:                     greska(GRESKA2_DAT);
15:
16:             /*Datoteka postoji*/
17:             fread(&broj_zapisa, sizeof(broj_zapisa), 1, fptr);
18:
19:             for(rbr=1; rbr<=broj_zapisa; rbr++)
20:             {       printf("%3d", rbr);
21:                     fread(&polozio, sizeof(polozio), 1, fptr);
22:
23:                     /*Prikaz podataka na ekranu*/
24:                     if(polozio.student1.ime[0] != '\0')
25:                     {       printf(" %-20s %-5d %-5d %-3d\n",
26:                                     polozio.student1.ime,polozio.student1.reg_broj,
27:                                     polozio.student1.god_upisa, polozio.broj_ispita);
28:                     }
29:                     else
30:                             printf(" Ne postoji student sa zadatim rednim brojem.\n");
31:             }
32:             fclose(fptr);
33:     }
```

```
1:      /*greska.c*/
2:      /*Modul sa funkcijom koja prikazuje na ekranu poruke o greskama*/
3:
4:      #include "extern.h"
5:
6:      void greska(Greska)
7:      {       fprintf(stderr, poruke_o_greskama[Greska]);
8:              exit(1);
9:      }
```

Анализа програма **primer6_3***

Овај програм има задатак да чита са тастатуре податке о испиту, уписује их у датотеку и ажурира садржај датотеке.

Датотеке заглавља могу се видети на почетку изворног кôда програма:

- *direkt.h* је датотека која садржи директиве *#include* за све библиотеке потребне програму и директиве *#define* за симболичке константе програма;
- *global.h* је датотека са глобалним дефиницијама и она је укључена само у модул *glavni.c*. Ова датотека поред дефиниције типова (набројаних симбола), низа показивача на поруке о грешкама и декларације функције *greska(),* садржи дефиницију структуре типа *student (ime, reg_broj, god_upisa)*, дефиницију друге структуре типа *pol_ispiti (*која међу члановима има и структуру типа *student* и целобројну променљиву *broj_ispita*), глобалне декларације

променљиве *broj_zapisa*, показивача *fptr* на структуру типа *FILE*, као и свих потребних функција: *dod_dat(), azur_dat()* и *prikaz_dat().* Свака од последње три наведене функције не добија аргументе и не враћа резултујућу вредност остатку програма;

- *extern.h* је датотека коју укључују сви модули сем главног. Ова датотека има све дефиниције као и датотека *global.h* (са службеном речи *extern* испред дефиниције).

Изворни кôд програма распоређен је по модулима *glavni.c, dod_dat.c, azur_dat.c, prikaz_dat.c* и *greska.c*:

- *glavni.c* је модул са главном функцијом која у једној *do.. while* петљи приказује опције: *1.Dodavanje, 2.Azuriranje, 3.Prikaz, 4.Izlaz*, све док се не изабере опција 4. За изабрану опцију 1. позива функцију *dod_dat(),* за изабрану опцију 2. функцију *azur_dat(),* а за изабрану опцију 3. функцију *prikaz_dat();*
- *dod_dat.c* је модул који садржи функцију *dod_dat()*. Ова функција проверава (линија 10.) да ли бинарна датотека *("ispiti.bin")* постоји на текућем директоријуму, тако што пита да ли може бити отворена за читање и упис. Ако датотека не постоји, отвара се бинарна датотека са истим именом за упис и читање *("wb+")* (линије 12. и 13.), поставља *broj_zapisa* (структура) на вредност 0 и уписује број записа на почетак датотеке (линије 15. и 16.). У супротном, ако датотека са задатим именом постоји (линија 19.) функција чита број записа са почетка датотеке (линија 20.) и у једној *while(1)* петљи (линије од 23. до 61.):
  - тражи са тастатуре редни број студента, а онда и све податке о студенту и испиту и смешта учитане податке у чланове структуре *polozio.student1*:
    polozio.student1.ime,
    polozio.student1.reg_broj,
    polozio.student1.god_upisa,
  - променљивој *polozio.broj_ispita* за почетак додељује вредност 0 (линија 38.),
  - ако је задати редни број већи од постојећег броја записа, поставља текућу позицију на крај датотеке (линија 42.), број записа се инкрементира и све док је мањи од траженог редног броја одвајају се места за празне записе (линије 44. и 45.), да би било могуће додавање записа иза краја постојећих у датотеци,
  - у супротном, ако је задати редни број мањи од постојећег броја записа, помера се текућа позиција на почетак записа (структуре) са задатим редним бројем (линије 52. и 53.), да би било могуће додавање записа пре краја постојећих записа.
  - додаје запис (садржај структуре) датотеци (линија 56.)
  - враћа текућу позицију на почетак датотеке и уписује нови број записа на почетак датотеке (линије 59. и 60.);
- *azur_dat.c* је модул који садржи функцију *azur_dat()*. Ова функција прво проверава да ли датотека са именом *"ispiti.bin"* постоји на текућем директоријуму тако што испитује да ли се може отворити за читање и упис (линије 9. и 10.). Ако датотека постоји, следи

читање броја записа са њеног почетка (линија 13.). Онда, у једној *while(1)* петљи (линије од 16. до 54.):

– тражи да се унесе преко тастатуре редни број студента (записа у датотеци) и чита овај број (линије од 18. до 28.),

– помера текућу позицију на почетак записа (структуре) са задатим редним бројем (линије 31. и 32.),

– чита податке о студенту са редним бројем задатим преко тастатуре (линија 33.),

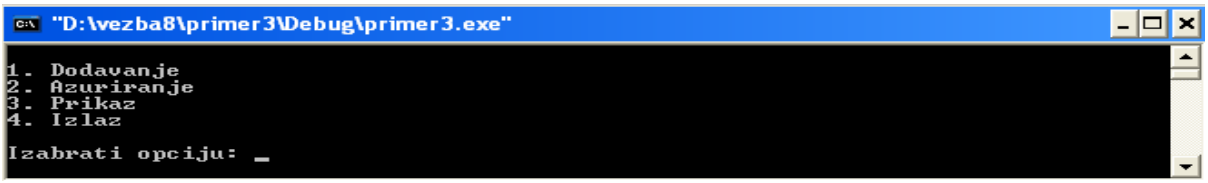
– ако запис са задатим редним бројем није празан (линија 36.) приказује на екрану податке прочитане из датотеке (*polozio.student1.ime,polozio.student1.reg_broj,polozio.student1.god_upisa, polozio.broj_ispita*), тражи са тастатуре број додатно положених испита, чита са тастатуре овај број и инкрементира број положених испита (линија 46.). Онда враћа текућу позицију на почетак записа са задатим редним бројем и уписује нов садржај структуре у датотеку (линије 49. и 50.),

– у супротном, ако је запис са задатим редним бројем празан, приказује на екрану одговарајућу поруку. Када се зада редни број записа ван дозвољеног опсега прекида се петља и затвара отворена датотека (линија 55.);

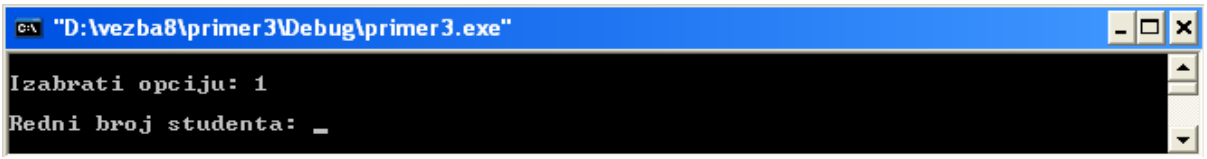
– *prikaz_dat.c* је модул који садржи функцију *prikaz_dat()*. Ова функција на почетку проверава да ли датотека *"ispiti.bin"* постоји на текућем директоријуму тако што покушава да отвори ову датотеку за читање и упис (линије од 10. до 14.). Ако датотека постоји функција даље чита са почетка датотеке број записа (линија 17.), а онда у једној *for* петљи чита сваки запис (садржај структуре) и ако није празан, приказује на екрану садржај (линије од 19. до 31.). Функција на крају затвара отворену датотеку (линија 32.);

– *greska.c* је модул који садржи познату функцију за приказ на екрану порука о грешкама.

На сликама 6.3 а) до 6.3 љ) може се видети један могући сценарио извршавања програма.

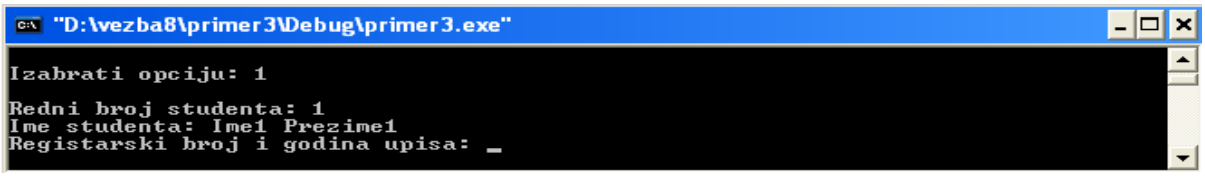


Слика 6.3 а) Излаз програма **primer6_3** (први део)

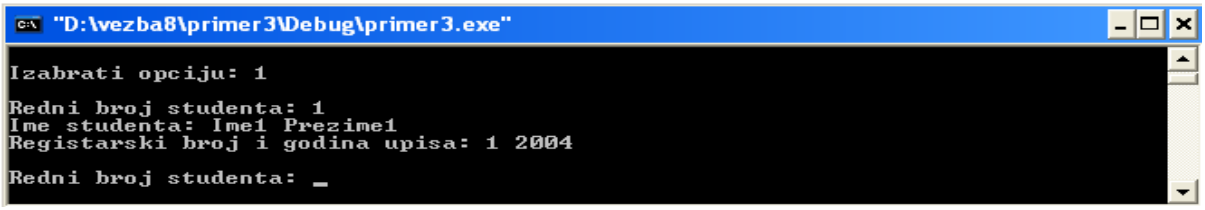


Слика 6.3 б) Излаз програма **primer6_3** (други део)

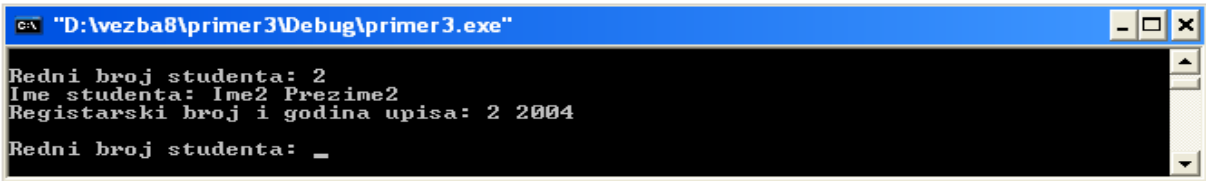
Слика 6.3 в) Излаз програма **primer6_3** (трећи део)



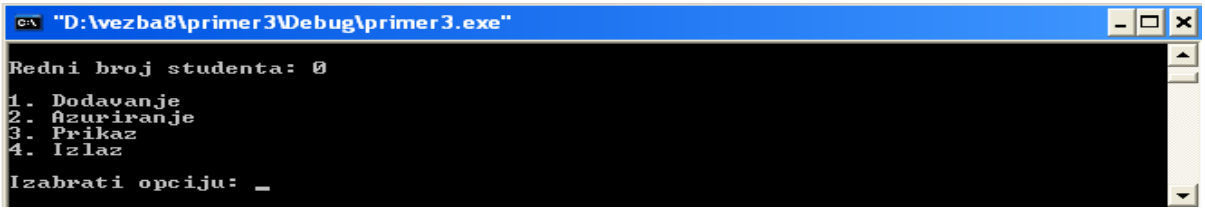
Слика 6.3 г) Излаз програма **primer6_3** (четврти део)



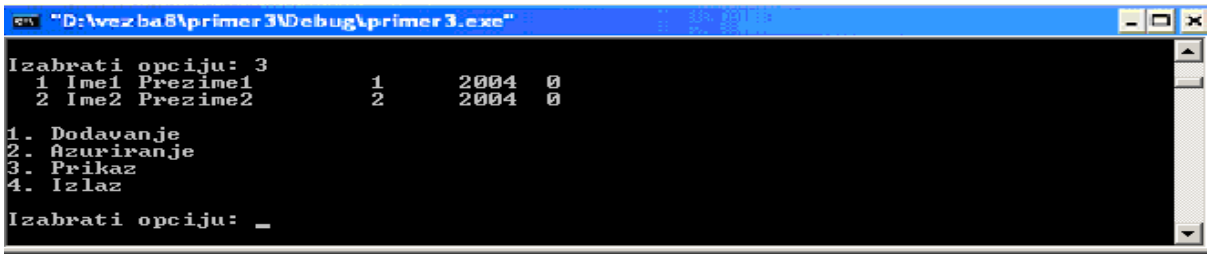
Слика 6.3 д) Излаз програма **primer6_3** (пети део)



Слика 6.3 ђ) Излаз програма **primer6_3** (шести део)



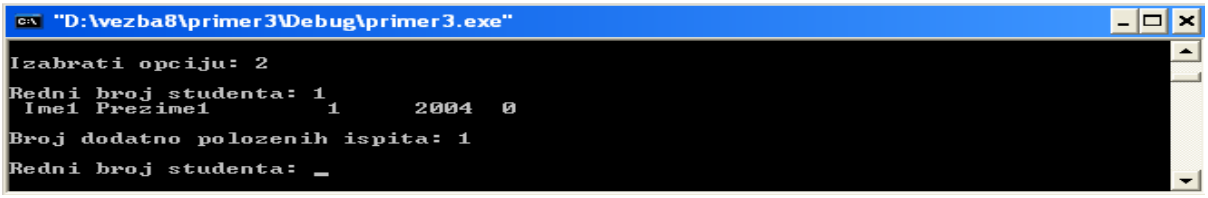
Слика 6.3 е) Излаз програма **primer6_3** (седми део)



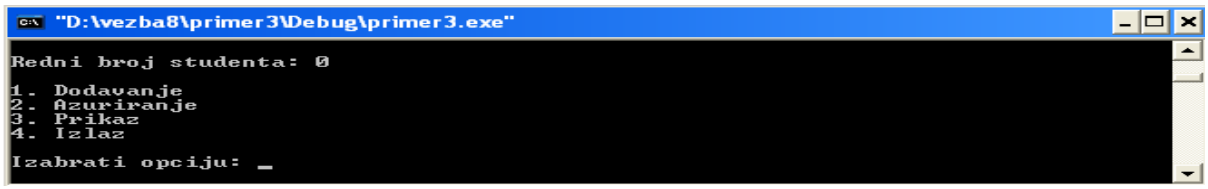
Слика 6.3 ж) Излаз програма **primer6_3** (осми део)

Слика 6.3 з) Излаз програма **primer6_3** (девети део)

Слика 6.3 и) Излаз програма **primer6_3** (десети део)



Слика 6.3 ј) Излаз програма **primer6_3** (једанаести део)



Слика 6.3 к) Излаз програма **primer6_3** (дванаести део)

Слика 6.3 л) Излаз програма **primer6_3** (тринаести део)

Слика 6.3 љ) Излаз програма **primer6_3** (четрнаести део)

Изворни кôд програма **primer6_4**

```
1:      /*primer6_4.c*/
2:      /*Odredjivanje veceg od dva zadata cela broja, pomocu makroa*/
3:
4:      #include <stdio.h>
5:
6:      /*Definicije Makroa*/
7:      #define UCITAJ_CELE(x,y)        scanf("%d%d",&x,&y)
8:      #define ISPISI_TEKST(x)         printf(#x)
9:      #define ISPISI_CEO(x)           printf("%d\n\n",(x))
10:     #define MAX(x,y)                ((x)>(y))?(x):(y)
11:
12:     main()
13:     {       int a, b;
14:
15:             /*Pozivi Makroa*/
16:             ISPISI_TEKST(\nZadati dva cela broja:\t\t);
17:             UCITAJ_CELE(a,b);
18:             ISPISI_TEKST(Veci od dva zadata broja je:\t);
19:             ISPISI_CEO(MAX(a,b));
20:
21:             return 0;
22:     }
```

Анализа програма **primer6_4**

Претпроцесорска директива #define може се користити и за дефиницију макроа

Програм у овом примеру одређује већи од два задата цела броја, помоћу одговарајућих макроа.

Дефиниција макроа почиње исто као и дефиниција симболичке константе, директивом *#define* и исто се, у фази припреме кôда за превођење у програму, симбол макроа замењује својом дефиницијом.

Разлика је у томе што уместо простог израза (само једне вредности константе), дефиниција макроа може садржати веома сложен израз и што макро може имати и аргументе (као и функција).

Сви макрои написани у овом програму имају аргументе. Из дефиниција макроа (линије од 7. до 10.) може се видети да ће претпроцесор у фази припреме кôда за превођење заменити:

- макро *UCITAJ_CELE(x,y)* изразом *scanf("%d%d",&x,&y),*
- макро *ISPISI_TEKST(x)* изразом *printf("x")* (пише се *printf(#x)),*
- макро *ISPISI_CEO(x)* изразом *printf("%d\n\n",(x)),*
- макро *MAX(x,y)* изразом *((x)>(y))?(x):(y).*

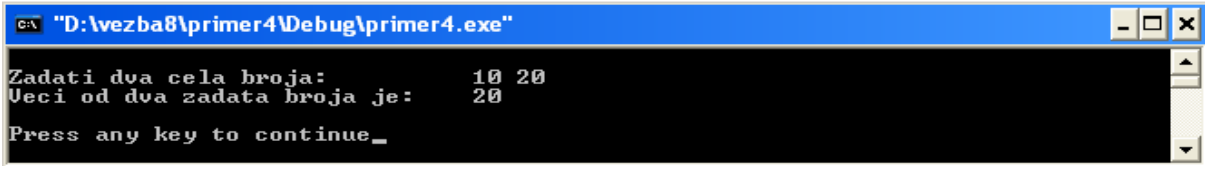
У изразу макроа сваки и најпростији израз, као што је име променљиве, треба написати између малих заграда (погледати макрое *ISPISI_CEO* и *MAX*). Макро *ISPISI_TEKST* је специфичан због тога што користи знак #. Овај знак се може користити када функција има само један аргумент.

Претпроцесор тражи у програму знаковне низове *UCITAJ_CELE, ISPISI_TEKST, ISPISI_CEO* и *MAX* (позиве макроа) и замењује свако њихово појављивање одговарајућим изразом. У позивима макроа (линије од 16. до 19.) треба написати стварне аргументе (као и код функција) и знак тачка-зарез на крају позива да би се од израза који се уграђује на том месту у кôд, формирала наредба програма. На сликама 6.4 а) и 6.4 б) приказан је излаз овог програма.

Макро се за разлику од функција, у фази претпроцесирања уграђује у кôд на сваком месту свог позива, што значи већи број линија кода, али не троши време на предају аргумената и повратак вредности, као што је то случај код функција.

Недостаци макроа су недостатак провере броја и типова аргумената при позиву (што код функција није случај) и немогућност дефинисања показивача на макро (показивачи на функције се користе). Макрои се због тога користе уместо функција само у оним деловима једног програма где је критична брзина његовог извршавања.

Слика 6.4 а) Излаз програма **primer6_4** (први део)



Слика 6.4 б) Излаз програма **primer6_4** (други део)

Практични савети

- ✓ У датотеке заглавља издвојити: директиве, глобалне дефиниције објеката у програму и глобалне дефиниције функција (ако их има пуно, распоредити их у више оваквих датотека).
- ✓ Изворни кôд распоредити по модулима тако да у главном модулу, поред позива функција из осталих модула буде минимална количина кôда.
- ✓ Сваки модул програма, различит од главног, може имати једну функцију или више функција које чине логичку целину у програму.
- ✓ Програм написати са функцијама а онда разматрати замену појединих функција са макроима (ако је то неопходно).

Евентуалне грешке

- ➢ Датотека заглавље са глобалним дефиницијама не може бити укључена у више програмских модула. Обично ову датотеку укључује главни модул, а датотеку са екстерним дефиницијама (која се формира од оригиналне датотеке са дефиницијама) укључују сви остали програмски модули.
- ➢ У датотеци са екстерним дефиницијама нису могуће иницијализације због тога што се она само позива на датотеку заглавље са оригиналним дефиницијама.

Задаци за припрему вежбе 6. у лабораторији

Задатак 1.

Написати програм на језику *C* који копира текст из једне датотеке у другу на следећи начин: сваку реченицу издваја у посебан ред (реченица се завршава знаком тачка). Распоредити изворни кôд програма по датотекама: *direkt.h, global.h, extern.h, glavni.c, kopira.c, greska.c*.

Задаци за вежбу 6. у лабораторији

Распоредити изворне кôдове следећих програма по одговарајућим модулима. Предвидети да програми прихватају имена потребних датотека:

а) из командне линије, при позиву програма,
б) током извршавања програма.

Задатак 1.

Написати програм на језику *C* који копира текст из једне датотеке у другу тако што сваки празан знак замењује знаком доња црта (_).

Задатак 2.

Написати програм на језику *C* који приказује на екрану бројеве појављивања директива: #include и #define, у свакој датотеци заглавље овог програма.

Задатак 3.

Написати програм на језику *C* који уписује у бинарну датотеку задати списак од *n (n≤100)* студената, са следећим подацима за сваког студента: ime, broj_indeksa, adresa и br_telefona и приказује на екрану све постојеће податке за студента са траженим бројем индекса (из претходно формиране датотеке).

Задатак 4.*

Написати програм на језику *C* који кодира текст из једне датотеке, копира га у другу тако што дуплира знак доња црта (_), а онда декодира текст из друге датотеке, копира га у трећу, тако што два знака доња црта замењује једним истим знаком.

Задатак 5.*

Написати програм на језику *C* који уписује у бинарну датотеку *m (m≤10)* низова целих бројева (задатих преко тастатуре), од којих је сваки исте дужине *n (n≤10),* а онда чита из исте датотеке и приказује на екрану низове који су формирани само од позитивних вредности.

Вежба 7.

Улаз / излаз података у програмима на језику *C++*

Базне стандардне класе за улаз / излаз

Стандардне функције и оператори за улаз / излаз

Форматиран улаз / излаз

Наредбе за контролу тока и низови

Примери

Изворни кôд програма **primer7_1**

```
1:      //primer7_1.cpp
2:      //Realizacija ulaza/izlaza (tastatura/ekran) za razlicite vrste podataka
3:      #include <iostream>                  //potrebna biblioteka ulaza/izlaza
4:      #define MAX1        80               //broj korisnih karaktera u stringu
5:      #define MAX2        10               //broj karaktera stringa, koji se citaju
6:      using namespace std;                 //potreban imenski prostor za biblioteke
7:
8:      int main( )                          //tip rezultata glavne funkcije neophodan
9:      {       int znak, broj_i;    float  broj_f;    char   odgovor, string[MAX1+1];
10:
11:             cout << "\nUlaz/Izlaz podataka u C++ programu\n";
12:             cout << "Zelite li da nastavite (D/N)? ";
13:             cin  >> odgovor;
14:             cin.ignore();                        //prihvata znak za prelaz u novi red
15:
16:             if(odgovor == 'D' || odgovor == ‘d’)
17:             {       cout << "\nCelobrojna vrednost: ";
18:                     cin  >>  broj_i;
19:                     cout << "Zadata celobrojna vrednost je: " << broj_i << endl;
20:
21:                     cout << "\nRealna vrednost: ";
22:                     cin  >>  broj_f;
23:                     cout << "Zadata realna vrednost je: " << broj_f << endl;
24:
25:                     //Citanje do prvog belog znaka od zadatog reda znakova
26:                     cout << "\nRed teksta:\n";
27:                     cin  >> string;
28:                     cout << "Prva rec je: " << string << endl;
29:
30:                     //Citanje ostatka od zadatog reda znakova
31:                     cin.get(string, MAX1+1);    //prihvata ostatak do kraja reda
32:                     cin.ignore();               //prihvata znak za prelaz u novi red
33:
34:                     //Citanje znak po znak do znaka za prelaz u novi red
35:                     cout << "\nRed teksta:\n";
36:                     while((znak = cin.get()) != '\n')
37:                             cout.put((char)znak);        //moze i: cout << (char)znak;
38:
39:                     //Citanje zadatog broja znakova ili do znaka za prelaz u novi red
40:                     cout << "\n\nRed teksta:\n";
41:                     cin.get(string, MAX2+1);
42:                     cout << "Prvih " << MAX2 << " znakova: " << string << endl;
43:             }
44:             return 0;                                                    }
```

Анализа програма **primer7_1**

Код пројектовања програма на језику *C++* користи се објектно оријентисана метода. То значи да се за потребе програма пројектују групе објеката које могу бити међусобно повезане. Објекти у овим програмима настају као конкретни примерци класа. Класа је скуп података (чланова) и операција (метода или функција чланица) које се примењују над тим подацима, при чему је све то повезано у целину. Подацима и функцијама, члановима једне класе, приступа се преко објеката тих класа и помоћу оператора тачка (.): *ime_objekta_klase.ime_clana.*

Иако се у неким програмима ове збирке не дефинишу класе и не декларишу објекати класа, неизбежна је употреба класа и њихових чланова, функција и оператора из стандардних *C++* библиотека.

Стандардна библиотека улаза/излаза програмског језика *C* (*stdio.h*) може се користити и у програмима на језику *C++*, али углавном се не користи због елегантнијег решења у језику *C++*. Програмски језик *C++* има своје библиотеке улаза/излаза (*iostream* и *fstream*) са скупом класа (*istream, ostream, fstream, ifstream, ofstream*) и њихових објеката (*cin, cout, cerr*). За коришћење поменутих *C++* библиотека, потребно је укључити глобални именски простор std у коме су дефинисане ове библиотеке, помоћу наредбе using namespace std; (линија 6.).

Овде ће бити поменуте само класе и објекти за рад са тастатуром и екраном. Класа *istream* садржи операције улаза са тастатуре, класа *ostream* операције излаза, а класа *iostream* операције улаза и излаза. Постоје три објекта који се користе:

- *cin* је објекат класе *istream* који је повезан са главним стандардним улазом (тастатуром),
- *cout* је објекат класе *ostream* који је повезан са главним стандардним излазом (екраном),
- *cerr* је објекат класе *ostream* који је повезан са главним стандардним излазом (екраном), без бафероваља и могућности преусмеравања.

Уместо стандардних функција *scanf()* и *printf(),* писања симбола % и тачног формата, у програмима на језику *C++* користе се стандардни оператори (оператор улаза >> и оператор излаза <<) и могућност преклапања оператора (једна од најбољих могућности коју пружа језик *C++*). Преклапање оператора значи да се симбол >> може користити за читање са тастатуре било ког податка стандардног *C++* типа, а симбол << за приказ на екрану било ког податка стандардног *C++* типа (у зависности од типа променљиве, написане десно од оператора, користи се одговарајући оператор).

У овом примеру програм прво приказује на екрану поруку (линије 11. и 12.). Ево еквивалентних наредби на језицима *C* и *C++*:

C: *printf("\nUlaz/Izlaz podataka u C programu\n")*

C++: *cout << "\nUlaz/Izlaz podataka u C++ programu\n";*

Као што се може видети, оператор излаза (<<) не генерише прелаз у нови ред већ се користи контрола *\n* (као и код функције *printf()*).

Програм (из линије 13.) чита са тастатуре одговор (*D/N*) податак типа *char* (*cin >> odgovor*). Оператор улаза (>>) чита са тастатуре све до првог белог знака. За прихватање знака за прелаз у нови ред у овом случају је искоришћена функција *ignore(),* над објектом *cin (cin.ignore()),* класе *istream*. Ако је унето слово *D (d)* (линија 16.), програм тражи једну по једну врсту података и

користи преклопљени оператор улаза (>>) да учита: цео број (линија 18.), реалан број (линија 22.) и низ знакова до првог белог знака (линија 27.). На крају линије 19. може се видети контрола *endl,* која преводи у нови ред и празни све бафере излазног тока.

Ако задати низ знакова има више речи, остатак речи из задатог реда може се учитати (линија 31.) помоћу функције *get()* класе *istream*. Функцији *get()* предаје се показивач на низ карактера који треба да чува знаковни низ и максимални број знакова. Помоћу наредбе *cin.get(string, MAX1+1)* програм чита *MAX1* знакова или до знака за прелаз у нови ред и смешта их у низ *string*.

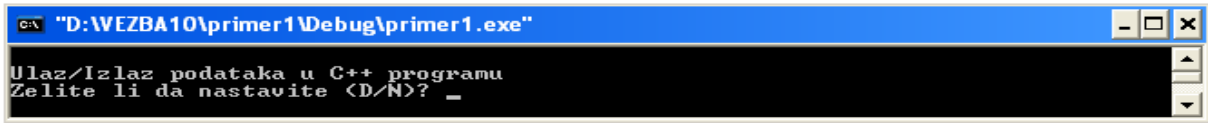
За читање траженог низа знакова са тастатуре и прихватање знака за прелаз у нови ред, уместо позива функција *cin.get(string, MAX1+1)* и *cin.ignore()* може се писати позив једне функције *cin.getline(string, MAX1+1)* (биће коришћена даље у програмима).

Функција *get()* без аргумената може се користити у комплету са функцијом *put()* класе *ostream* за читање и приказ на екрану једног по једног знака (линије 36. и 37.). Функцији *put()* је потребан аргумент и да би приказала знак (а не кôд знака) потребно је назначити јој то у аргументу (*char*).

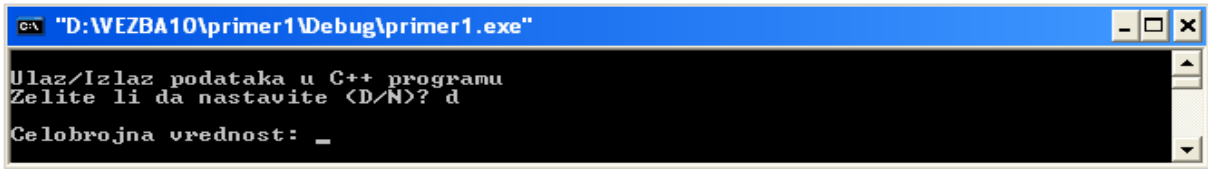
У следећем делу програма (линија 41.) функцији *get()* се предају аргументи: име низа карактера (*string*) и број знакова *MAX2+1,* да прочита са тастатуре највише *MAX2* (10*)* знакова.

Дакле, функција *get()* има више верзија. То је једна од преклопљених функција. Функције у језику *C++* са истим именом и различитим бројем и типовима аргумената зову се преклопљене.

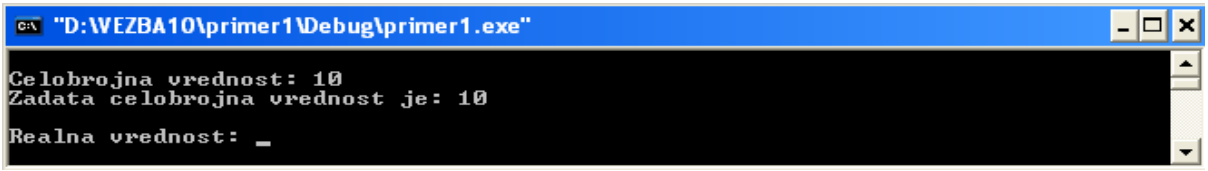
Резултати описаног тока програма могу се видети на сликама од 7.1 а) до 7.1 е).



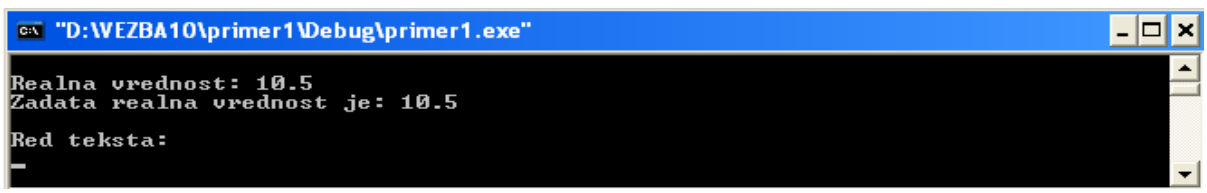
Слика 7.1 а) Излаз програма **primer7_1** (први део)



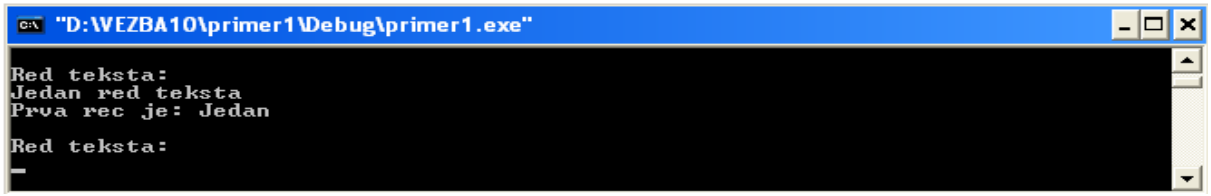
Слика 7.1 б) Излаз програма **primer7_1** (други део)



Слика 7.1 в) Излаз програма **primer7_1** (трећи део)

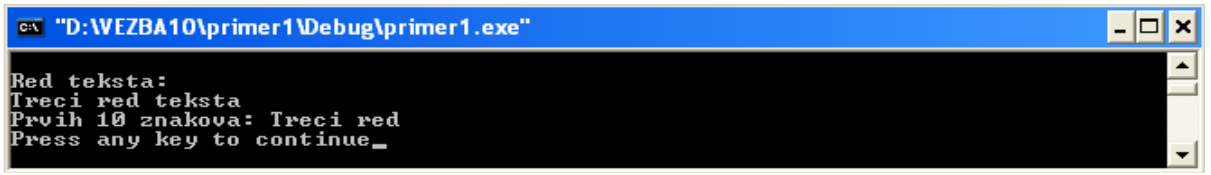


Слика 7.1 г) Излаз програма **primer7_1** (четврти део)



Слика 7.1 д) Излаз програма **primer7_1** (пети део)

Слика 7.1 ђ) Излаз програма **primer7_1** (шести део)



Слика 7.1 е) Излаз програма **primer7_1** (седми део)

Изворни кôд програма **primer7_2**

```
1:      //primer7_2.cpp
2:      //Primena funkcija za konverziju i formatiranje ulaza/izlaza
3:      #include <iostream>
4:      #include <iomanip>
5:      using namespace std;
6:
7:      void stampa_rbr( void );
8:
9:      int main( )
10:     {       char slovo = 'A';
11:             char string[] = "Formatiranje C++ ulaza/izlaza";
12:             int broj_i = 1234;
13:             double broj_d = 1.23456;
14:
15:             //1. stampanje samog slova A
16:             stampa_rbr();
17:             cout << slovo;
18:
19:             //2. stampanje ASCII koda slova A
20:             stampa_rbr();
21:             cout << (int)slovo;
22:
23:             //3. stampanje znaka zadate ASCII vrednosti (slova a)
24:             stampa_rbr();
25:             cout << (char)97;
26:
```

```
27:             //4. stampanje niza znakova
28:             stampa_rbr();
29:             cout << string;
30:
31:             //5. stampanje prvih 5 znakova od niza
32:             stampa_rbr();
33:             cout.write(string,5);
34:
35:             //6. stampanje celog broja u oktalnom obliku
36:             stampa_rbr();
37:             cout.setf(ios::oct);
38:             cout << broj_i;                         //moze i: cout << oct << broj_i;
39:             cout.unsetf(ios::oct);
40:
41:             //7. stampanje celog broja u heksadecimalnom obliku (malim slovima)
42:             stampa_rbr();
43:             cout.setf(ios::hex);
44:             cout << broj_i;                         //moze i: cout << hex << broj_i;
45:
46:             //8. stampanje celog broja u heksadecimalnom obliku (velikim slovima)
47:             stampa_rbr();
48:             cout.setf(ios::uppercase);
49:             cout << broj_i;
50:             cout.unsetf(ios::hex);
51:
52:             //9. stampanje celog broja u decimalnom obliku
53:             stampa_rbr();
54:             cout << broj_i;                         //moze i: cout << dec << broj_i;
55:
56:             //10. stampanje celog broja,u polju zadate sirine, desno poravnanje
57:             stampa_rbr();
58:             cout.width(10);
59:             cout << broj_i;
60:
61:             //11. stampanje celog broja, u polju zadate sirine, levo poravnanje
62:             stampa_rbr();
63:             cout.width(10);
64:             cout.setf(ios::left);
65:             cout << broj_i;
66:             cout.unsetf(ios::left);
67:
68:             //12. stampanje celog broja, desno poravnanje, nule ispred broja
69:             stampa_rbr();
70:             cout.width(10);
71:             cout.fill('0');
72:             cout << broj_i;
73:             cout.fill(' ');
74:
75:             //13. stampanje realnog broja, sa standardnom preciznoscu
76:             stampa_rbr();
```

```
77:             cout << broj_d;
78:
79:             //14. stampanje realnog broja, u polju zadate sirine,
80:             stampa_rbr();
81:             cout.width(10);
82:             cout << broj_d;
83:
84:             //15. stampanje realnog broja, sa 2. decimalne cifre
85:             stampa_rbr();
86:             cout.width(10);
87:             cout.precision(2);
88:             cout << broj_d;
89:
90:             //16. stampanje realnog broja, sa eksponentom
91:             stampa_rbr();
92:             cout.setf(ios::scientific);
93:             cout << broj_d << endl;
94:             cout.unsetf(ios::scientific);
95:
96:             return 0;
97:     }
98:     //Funkcija koja prikazuje redni broj na pocetku reda
99:     void stampa_rbr (void)
100:    {       static int redni_broj = 0;
101:            cout << "\n";
102:            cout.width(2);
103:            cout << ++redni_broj << ".  ";
104:    }
```

Анализа програма **primer7_2**

Као и у језику *C* и овде је могуће форматирање улаза и излаза. Да би се могле користити све могућности (функције и ознаке формата у њима) потребно је укључити и библиотеку са функцијама за манипулацију формата *(iomanip).*

Овај програм приказује на екрану вредности целобројних и реалних података (декларисаних у линијама од 10. до 13.) у разним форматима (линије од 15. до 94.):

- Ако променљива *slovo* садржи кôд слова A, важи да:
  cout << slovo; - приказује се на екрану слово,
  cout << (int)slovo; - приказује на екрану вредност *ASCII* кôда.
- За приказ на екрану знака (на пример слова a) чији је кôд (на пример 97) познат користи се наредба:
  cout << (char)97;
  а за приказ низа знакова, који чува низ карактера *string*
  cout << string;
- Помоћу функције *write()* класе *ostream* може се дефинисати минимална ширина поља (5) на екрану за приказ податка (*string*):
  cout.write(string,5);
- Цели бројеви се читају и пишу у децималном облику, ако се не дефинише другачије:

cout.setf(ios::oct); - мења подразумевани децимални облик за приказ целог броја у октални,

cout.unsetf(ios::oct); - враћа на децимални облик за приказ целог броја,

cout.setf(ios::hex); - мења подразумевани децимални облик за приказ целог броја у хексадецимални,

cout.setf(ios::uppercase); - мења подразумевана мала слова у иста велика у хексадецималном облику,

cout.unsetf(ios::hex); - враћа на претходно дефинисани облик за приказ целог броја,

cout.width(10); - поставља минималну ширину поља (10) за приказ податка, поравнање вредности податка у резервисаном пољу је подразумевано десно,

cout.setf(ios::left); - мења подразумевано десно поравнање у лево,

cout.unsetf(ios::left); - враћа на подразумевано десно поравнање,

cout.fill('0'); - попуњава места испред податка нулама,

cout.fill(' '); - попуњава места испред податка празним знаковима,

cout.precision(2); - мења подразумевану прецизност приказа реалног броја од шест децималних места на два,

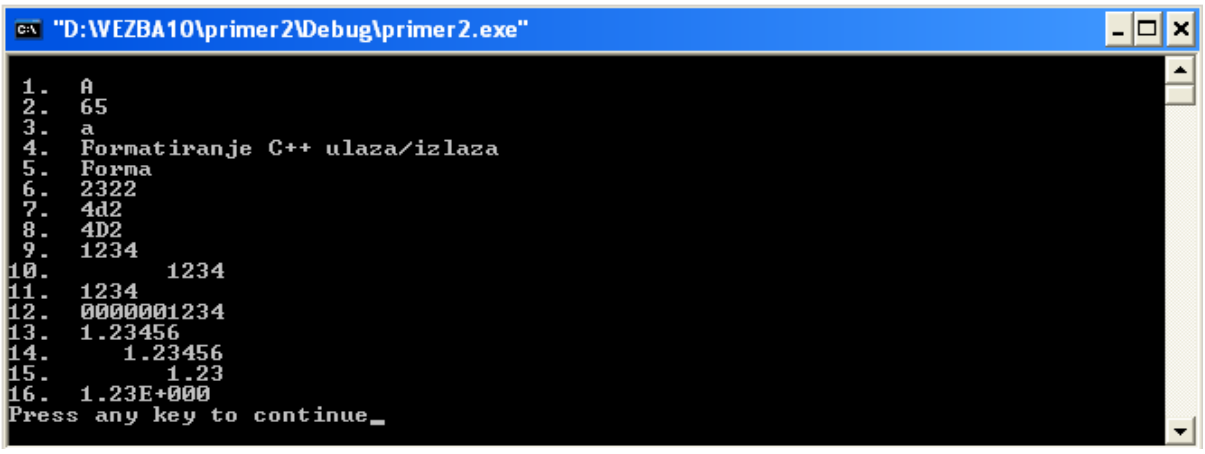
cout.setf(ios::scientific); - мења подразумевани облик за приказ реалног броја са децималном тачком, на облик са експонентом,

cout.unsetf(ios::scientific); - враћа на облик са децималном тачком за приказ реалног броја.

Формати који се поставе помоћу функције *setf()* важе од те наредбе даље у програму, до позива функције *unsetf()* за укидање постављеног формата.

Функцију *stampa_rbr()* (дефиниција у линијама од 99. до 104.) програм позива, да на почетку сваког реда прикаже на екрану редни број и знак двотачка.

Резултат програма може се видети на слици 7.2.



Слика 7.2 Излаз програма **primer7_2**

Изворни кôд програма **primer7_3**

```
1:      //primer7_3.cpp
2:      //Tabeliranje vrednosti izraza: 10 stepenovan brojem x
3:      //u zadatom opsegu i sa zadatim korakom za x
4:      #include <iostream>
5:      #include <iomanip>
6:      #include <math.h>
7:      #define MIN          0       //donja granica opsega vrednosti x
8:      #define MAX          20      //gornja granica opsega vrednosti x
9:      #define MAX1         2       //sirina polja za prikaz vrednosti x
10:     #define MAX2         10      //sirina polja za prikaz vrednosti izraza
11:     using namespace std;
12
13:     int main()
14:     {       int xmin, xmax, dx;
15:             do
16:             {       cout << "\nUnesite cele brojeve ";
17:                     cout << "xmin>=" << MIN << ", xmax<=" << MAX << " i dx: ";
18:                     cin >> xmin >> xmax >> dx;
19:             }while(xmin<MIN || xmax>MAX || dx>xmax);
20:
21:             //Stampanje na ekranu zaglavlja tabele i tabele
22:             cout << "\nx\t10 stepenovano brojem x\t\t10 stepenovano brojem x"
23:                     << "\n\t(desno poravnanje)\t\t(levo poravnanje)"
24:                     << "\n================================"
25:                     << "================================\n\n";
26:
27:             cout.setf(ios::fixed);
28:             cout.precision(0);
29:
30:             for(int x = xmin; x<=xmax; x+=dx)
31:             {       //Desno poravnanje vrednosti x  u 1. koloni tabele
32:                     cout.width(MAX1);
33:                     cout.unsetf(ios::left);
34:                     cout << x << "\t\t";
35:
36:                     //Desno poravnjanje vrednosti izraza u 2. koloni tabele
37:                     cout.width(MAX2);
38:                     cout.unsetf(ios::left);
39:                     cout << pow(10,x) << "\t\t";
40:
41:                     //Levo poravnjanje vrednosti izraza u 3. koloni tabele
42:                     cout.width(MAX2);
43:                     cout.setf(ios::left);
44:                     cout << pow(10,x) << "\n";
45:             }
46:             cout << "\n";
47:             return 0;
48:     }
```

Анализа програма **primer7_3**

Програм у овом примеру тражи (слика 7.3 а)) да се преко тастатуре задају целобројне вредности за број *x*: доња и горња граница опсега и корак промене (линије од 15. до 19.). Следи приказ (у петљи у линијама од 30. до 45.) на екрану у једној табели свих задатих вредности *x*:, а у другој и трећој табели вредности израза 10^x на два начина: са десним и левим поравнањем вредности израза у пољу дефинисане ширине. Програм је урађен коришћењем функција и сетовања формата описаних у претходном примеру.

Помоћу оператора излаза на екран (<<) и одговарајућих контролних секвенци за форматиран излаз, програм приказује на екрану заглавље табеле (линије од 22. до 25.).

За тражено степеновање користи се функција *pow(10,x)* из библиотеке *math.h*, чији је резултат типа *double*. Да би резултат ове функције био приказан у табели без експонента и без децимала, програм у линијама 27. и 28. користи функције *setf(ios::fixed)* и *precision(0)* над објектом *cout*. Програм даље, у једној *for* петљи приказује на екрану табелу у три колоне.

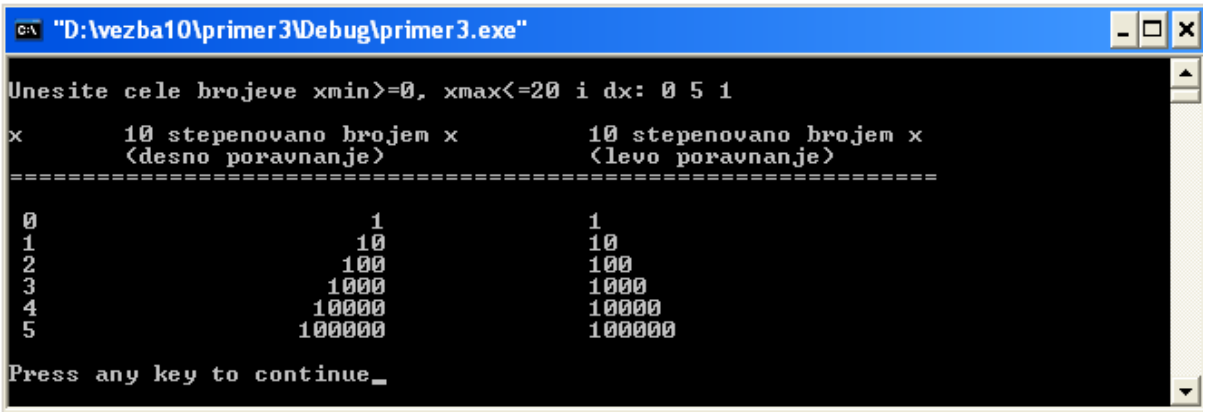
У првој колони табеле приказује се вредност броја *x*, у задатом опсегу, са задатим кораком. Ширина поља за приказ броја је *MAX1* (2) и поравнање уз десну ивицу. Ако је претходно у табели било постављено поравнање уз леву ивицу, помоћу функције *unsetf(ios::left)* враћа се подразумевано поравнање уз десну ивицу (линије 32. и 33.).

У другој колони табеле приказује се вредност израза 10^x, за све вредности из табеле броја *x*. Ширина поља за приказ броја је *MAX2* (10) и поравнање уз десну ивицу. Ако је претходно у табели било постављено поравнање уз леву ивицу, помоћу функције *unsetf(ios::left)* враћа се подразумевано поравнање уз десну ивицу (линије 37. и 38.).

У трећој колони табеле приказује се вредност израза 10^x, за све вредности броја *x* из табеле, али са поравнањем уз леву ивицу, помоћу функције *setf(ios::left),* док је ширина поља *MAX2* (10) , исто као и за другу колону табеле (линије 42. и 43.).

Резултат програма може се видети на слици 7.3 б).

Слика 7.3 а) Излаз програма **primer7_3** (први део)



Слика 7.3 б) Излаз програма **primer7_3** (други део)

Изворни кôд програма **primer7_4**

```
1:      //primer7_4.cpp
2:      //Citanje niza celih brojeva iz komandne linije i prikaz na ekranu
3:      //njegovih elemenata u oktalnom, heksadecimalnom i binarnom obliku
4:      #include <iostream>
5:      #include <iomanip>
6:      #include <stdlib.h>
7:      using namespace std;
8:
9:      void stampa_bin( int decimal_value );
10:
11:     int main(int argc, char *argv[])
12:     {       int decimalni;
13:
14:             if( argc < 2 )
15:             {       cerr << "\nUneti ime programa i niz celih brojeva.\n";
16:                     exit(1);
17:             }
18:
19:             cout << "\nZadati niz brojeva u raznim oblicima:\n\n";
20:             cout << "dec\tokt\theksa\tbin\n";
21:
22:             for(int i=1; i<argc; i++)
23:             {       decimalni = atoi( argv[i]);
24:                     cout << "\n" << decimalni << "\t";
25:                     cout << oct << decimalni << "\t";
26:                     cout << hex << decimalni << "\t";
27:                     stampa_bin(decimalni);
28:             }
29:
30:             return 0;
31:     }
32:     //Funkcija koja prikazuje na ekranu niz u binarnom obliku
33:     void stampa_bin(int broj)
34:     {       int broj_cifara = 0;
35:             int niz_binarnih[50];
36:
37:             while( broj != 0 )
38:             {       niz_binarnih[broj_cifara++] = broj % 2;
39:                     broj /= 2;
40:             }
41:
42:             for( int i=broj_cifara-1; i>=0; i-- )
43:                     cout << dec << niz_binarnih[i];
44:     }
```

Анализа програма **primer7_4**

Програм у овом примеру чита низ целих бројева из командне линије и приказује на екрану вредности свих елемената у октално, хексадецимално и бинарном облику.

Главна функција би у овом случају требало да прихвати из командне линије име програма и низ бројева (бар два аргумента). Ако то није случај (линије од 14. до 17.), користи се објекат *cerr* класе *ostream* за небаферовани излаз на екрану поруке и позив функције *exit(1)* за прекид програма. У супротном, ако су унети подаци у командној линији, следи у једној *for* петљи (линије од 22. до 28.):

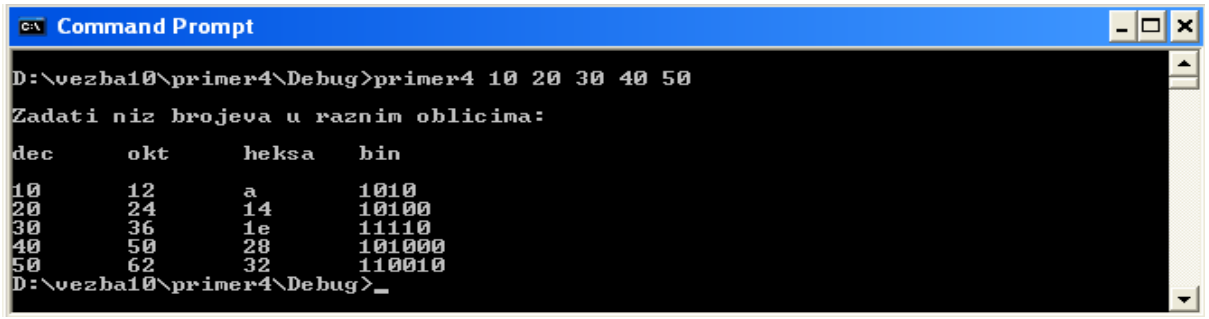
- конверзија податка у цео број, помоћу функције *atoi()* (линија 23),
- приказ броја *decimalni* у октално и хексадецимално облику (линије 25. и 26.). Овде се користити други могући начин (*decimalni*, *cout << oct << decimalni* и *cout << hex << decimalni*),
- приказ броја у бинарном облику (линија 27.) помоћу функције *stampa_bin(),* која користи познати алгоритам (линије од 33. до 44.).

Функција *stampa_bin()* користи као аргумент цео број и не враћа вредност, већ приказује на екрану овај број у бинарном облику.

У линији 35. декларисан је низ целих бројева *(niz_binarnih[50])*, који је даље искоришћен за конверзију у бинарни облик. У одговарајуће елементе формираног низа смештају се остаци дељења бројем 2 задатог броја и сваког броја који се добија даљим дељењем овог броја бројем 2, све док вредност тако добијеног броја није 0 (линије од 37. до 40.). Штампањем у петљи елемената низа, од последњег до првог (линије 42. и 43.), добија се на екрану бинарни облик броја који је функција добила у аргументу.

Командни промпт од програма и резултат рада програма за један задати низ целих бројева приказани су на сликама 7.4 а) и 7.4 б).

Слика 7.4 а) Излаз програма **primer7_4** (први део)



Слика 7.4 б) Излаз програма **primer7_4** (други део)

Практични савети

- ✓ За улаз/излаз података тастатура/екран, програми на језику *C++* користе библиотеку *iostream* (њене класе и објекте) и операторе улаза (>>) и излаза (<<). На располагању је и *C* стандардна библиотека *stdio.h*, али су њене функције форматираног улаза/излаза у језику *C++* замењене операторима, много једноставнијим за писање.
- ✓ Ако постоји дилема који је оператор улаза а који излаза, погледати како су усмерени симболи који се користе за ове операторе:
  - *cin* >> x; значи да са улазног тока (*cin*) податак доспева (>>) у претходно декларисану променљиву x,
  - *cout* << *x;* значи да се на излазни ток (*cout*) податак шаље (<<) из променљиве x.

Евентуалне грешке

- ➢ Формат постављен помоћу функције *setf()* важи до краја програма, а не само за следећу написану наредбу. За укидање овако постављеног формата неопходан је позив функције *unsetf().*
- ➢ И у програмима на језику *C++* треба обезбедити брисање заосталих карактера са стандардног улаза, на један од могућих начина (помоћу функције *ignore()* или *get()).*

Задаци за припрему вежбе 7. у лабораторији

Задатак 1.

Написати програм на језику *C++* који чита са тастатуре текст, реч по реч (реч је низ знакова између два бела знака) до речи *“kraj”*, или највише *n=20* речи, сортира све речи по абецедном реду и приказује их на екрану, сваку у посебном реду.

Задаци за вежбу 7. у лабораторији

Урадити следеће програме у две верзије:

1. са главном функцијом без аргумената и читањем свих потребних података током извршавања програма;
2. са главном функцијом која прихвата аргументе из командне линије и у овим аргументима добија све потребне податке.

Задатак 1.

Написати програм на језику *C++* који прихвата са тастатуре низ од *n (n≤10)* међусобно различитих целих бројева, сортира га у растући поредак и приказује на екрану.

Задатак 2.

Написати програм на језику *C++* који прихвата са тастатуре низ од *n (n≤5)* реалних бројева, израчунава и приказује на екрану вредности суме, најмањег и највећег елемента задатог низа.

Задатак 3.

Написати програм на језику *C++* који прихвата са тастатуре једну реч (реч је низ знакова између два празна знака) и одређује број појединих децималних цифара у задатом тексту.

Задатак 4.

Написати програм на језику *C++* који прихвата са тастатуре *n (n≤5)* речи (реч је низ знакова између два празна знака) и приказује на екрану задате речи, сортиране по дужини, од најдуже речи.

Вежба 8.

Функције, показивачи, динамичка додела меморије и структуре у програмима на језику *C++*

Функције са подразумеваним вредностима аргумената

Преклапање функција

Функције уграђене у кôд

Динамичка додела меморије

Структуре

Примери

Изворни кôд програма **primer8_1**

```
1:      //primer8_1.cpp
2:      //Pristup globalnoj promenljivoj (istog imena kao i lokalna)
3:      #include <iostream>
4:      using namespace std;
5:
6:      float suma(float f1, float f2);
7:      float f = 10.5;
8:
9:      int main()
10:     {       float a=10.5, b=10.5;
11:
12:             cout << "\n" << a << "(lok) + " << b << "(lok) + ";
13:             cout << f << "(glob) = " << suma(a, b) << endl;
14:
15:             return 0;
16:     }
17:     //Funkcija koja izracunava sumu
18:     float suma(float f1,float f2)
19:     {       float f = 20.5;
20:
21:             return(f1 + f2 + (::f) );
22:     }
```

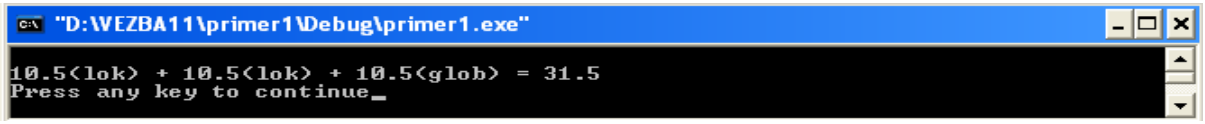
Анализа програма **primer8_1**

У програмима на језику *C++* важе иста правила досега промењљивих као и у програмима на језику *C* (локална вредност промењљиве има предност над глобалном вредношћу промењљиве са истим именом) али и нови оператор досега (::) који омогућује да се у функцији користи глобална вредност промењљиве, истог имена као што је локална за ту функцију.

У овом примеру промењљива *f* је глобална за целу датотеку (линија 7.) и могу је користити функције *main()* и *suma()*.

Главна функција има само приказ резултата функције *suma()* за стварне вредности *a=10.5* и *b=10.5* и приказ пропратног текста.

Функција *suma()* има локалну промењљиву *f* (исто име као од глобалне промењљиве), а у наредби повратка (линија 21.) не користи вредност локалне промењљиве *f*, већ истоимене глобалне, због тога што је испред имена промењљиве написан оператор досега *(::f)*. На овај начин превазиђен је локални досег и функција ће приказати на екрану збир вредности стварних аргумената (*a* и *b*) и глобално декларисане промењљиве *f* (слика 8.1).



Слика 8.1 Излаз програма **primer8_1**

Изворни кôд програма **primer8_2**

```
1:      //primer8_2.cpp
2:      //Koriscenje funkcije sa podrazumevanim vrednostima argumenata
3:      //za prikaz izabrane opcije i brojeva sa tastature
4:      #include <iostream>
5:      using namespace std;
6:
7:      void podr_argumenti(char c = 'X', int broj1 = 0, float broj2 = 0.5);
8:
9:      int main()
10:     {       podr_argumenti('A', 1, 1.5);        //stampa: A, 1, 1.5
11:
12:             podr_argumenti('B', 2);             //stampa: B, 2, 0.5
13:
14:             podr_argumenti('C');                //stampa: C, 0, 0.5
15:
16:             podr_argumenti();                   //stampa: X, 0, 0.5
17:
18:             return 0;
19:     }
20:     //Funkcija koja prikazuje na ekranu izabrane argumente
21:     void podr_argumenti(char c, int broj1, float broj2)
22:     {       cout << "\nIzabrana opcija: " << c << endl;
23:             cout   << "Ceo broj: " << broj1 << endl;
24:             cout   << "Realan broj: " << broj2 << endl;
25:     }
```

Анализа програма **primer8_2**

Програм у овом примеру користи особину функција на језику *C++* да имају при позиву подразумеване вредности својих аргумената.

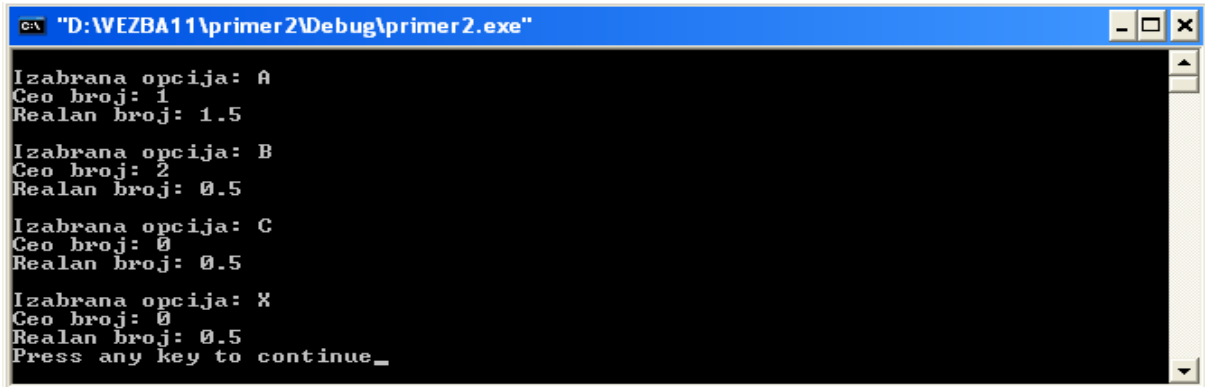
У линији 7. декларисана је функција *podr_argumenti()* и иницијализовани су њени формални аргументи (променљива *c* на *ASCII* кôд slova *X,* променљива *broj1* на вредност 0 и променљива *broj2* на вредност 0.5).

Дефиниција функције *podr_argumenti()* (линије од 21. до 25.) има наредбе за приказ на екрану вредности аргумената (*c, broj1 u broj2).*

Главна функција позива функцију *podr_argumenti()* на четири начина:

- предаје јој сва три стварна аргумента и то значи приказ на екрану ових вредности из функције *podr_argumenti()* (линија 10.),
- предаје јој два стварна аргумента, што значи да ће фунција *podr_argumenti()* користити подразумевану вредност трећег аргумента (*broj2 = 0.5)* (линија 12.),
- предаје јој само један стваран аргумент, што значи да ће фунција *podr_argumenti()* користити подразумеване вредности другог и трећег аргумента (*broj1 = 0* и *broj2 = 0.5)* (линија 14.),
- не предаје јој стварне аргументе, што значи да ће функција *podr_argumenti()* користити подразумеване вредности свих својих аргумената *(c = 'X', broj1 = 0* и *broj2 = 0.5)* (линија 16.).

На слици 8.2 могу се видети резултати позива функције *podr_argumenti().*



Слика 8.2 Излаз програма **primer8_2**

Изворни кôд програма **primer8_3**

```
1:      //primer8_3.cpp
2:      //Izracunavanje srednje vrednosti izabranog niza
3:      //pomocu preklopljenih funkcija
4:      #include <iostream>
5:      #define MAX     4                       //duzina niza
6:      using namespace std;
7:
8:      float srednja_vrednost(int niz_int[], int n);
9:      float srednja_vrednost(float niz_float[], int n);
10:
11:     int main()
12:     {       int a[MAX] = {1, 2, 3, 4};
13:             float b[MAX] = {1.25, 2.25, 3.25, 4.25};
14:
15:             cout << "\nSrednja vrednost niza celih brojeva:\t";
16:             cout << srednja_vrednost(a, MAX) << endl;
17:             cout << "Srednja vrednost niza realnih brojeva:\t";
18:             cout << srednja_vrednost(b, MAX) << endl;
19:
20:             return 0;
21:     }
22:     //Funkcija koja izracunava srednju vrednost niza celih brojeva
23:     float srednja_vrednost( int niz_int[], int n)
24:     {       int s = 0;
25:             for(int i=0; i<n; i++)
26:                     s += niz_int[i] ;
27:             return ((float)s/n);
28:     }
29:     //Funkcija koja izracunava srednju vrednost niza realnih brojeva
30:     float srednja_vrednost( float niz_float[], int n )
31:     {       float s = 0;
32:             for(int i=0; i<n; i++)
33:                     s += niz_float[i] ;
34:             return (s/n);
35:     }
```

Анализа програма **primer8_3**

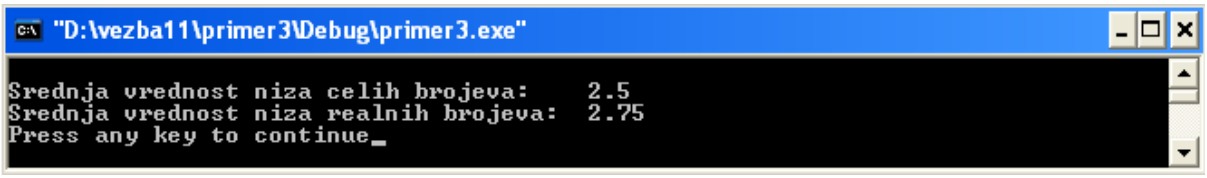
Поред преклапања оператора језик *C++* има и преклапање функција. То значи да у програму може бити више функција са истим именом али разликама у броју, типовима аргумената и типу повратне вредности. У зависности од броја и типова стварних аргумената позива се једна од више функција са истим именом. Разлика само у типу повратне вредности не може бити, јер би онда постојала дилема коју функцију позвати.

У овом програму декларисана је једна оваква функција *srednja_vrednost().* Једна од преклопљених функција је декларисана у линији 8. (један аргумент је показивач на целобројни низ, други је цео број, повратна вредност је реалног типа), а друга у линији 9. (један аргумент је показивач на низ реалних бројева, други је цео број, повратна вредност је реалног типа).

Прва од преклопљених функција (линије од 23. до 28.) израчунава средњу вредност целобројног низа чији је показивач и дужину добила у аргументима и враћа ову средњу вредност.

Друга од преклопљених функција (линије од 30. до 35.) израчунава средњу вредност низа реалних бројева, чији је показивач и дужину добила у аргументима и враћа ову средњу вредност.

Главна функција програма позива прву од преклопљених функција за низ целих бројева *a* (линија 16.), а онда другу за низ реалних бројева *b* (линија 18.) и приказује на екрану њихове резултате (слика 8.3).



Слика 8.3 Излаз програма **primer8_3**

Изворни кôд програма **primer8_4**

```
1:      //primer8_4.cpp
2:      //Prikaz na ekranu dela zadatog teksta, pomocu inline funkcije
3:      #include <iostream>
4:      #include <ctype.h>
5:      using namespace std;
6:
7:      inline void stampa_slovo(char kod_slova)        {  cout << kod_slova;  };
8:
9:      int main()
10:     {       char prvo, posl;
11:
12:             cout << "\n\"inline\" funkcija nema potrebe za stekom\n";
13:             cout << "C++ prevodilac proverava broj i tipove argumenata.\n";
14:
15:             do
16:             {       cout << "\nPrvo slovo za stampu: ";
17:                     cin >> prvo;
18:                     cout << "Poslednje slovo za stampu: ";
```

```
19:                     cin >> posl;
20:             }while(!isalpha(prvo) || !isalpha(posl) || prvo>posl);
21:
22:             cout << "\n";
23:             for(char c=prvo; c<=posl; c++)
24:                     stampa_slovo(c);
25:             cout << "\n\n";
26:
27:             return 0;
28:     }
```

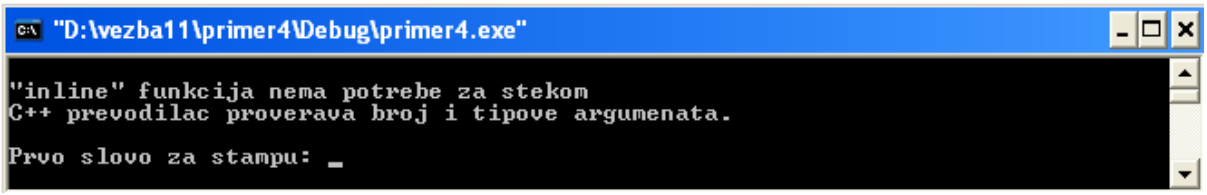
Анализа програма **primer8_4**

Програм у овом примеру садржи функцију која је уграђена у кôд (*inline* функцију).

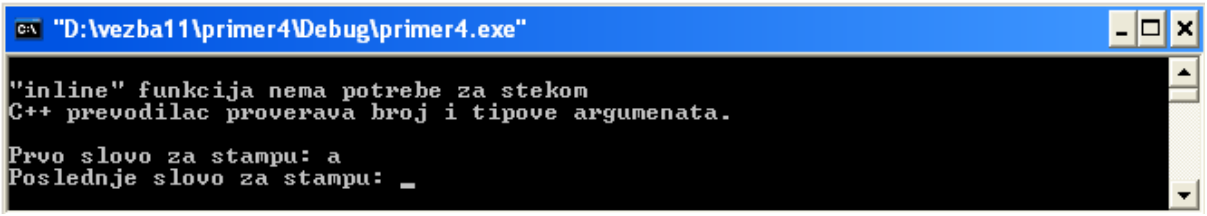
На овај начин тражи се од преводиоца да уметне функцију на место њеног позива. На тај начин штеди се време које би се утрошило на позив функције и на повратак из функције, што може бити значајно код функција које се често позивају, у кратком временском интервалу. Преводилац неће одобрити овај захтев ако процени да је дефиниција функције дуга.

У линији 7. дефинисана је *inline* функција *stampa_slovo()* која приказује на екрану слово чији *ASCII* кôд добија у аргументу.

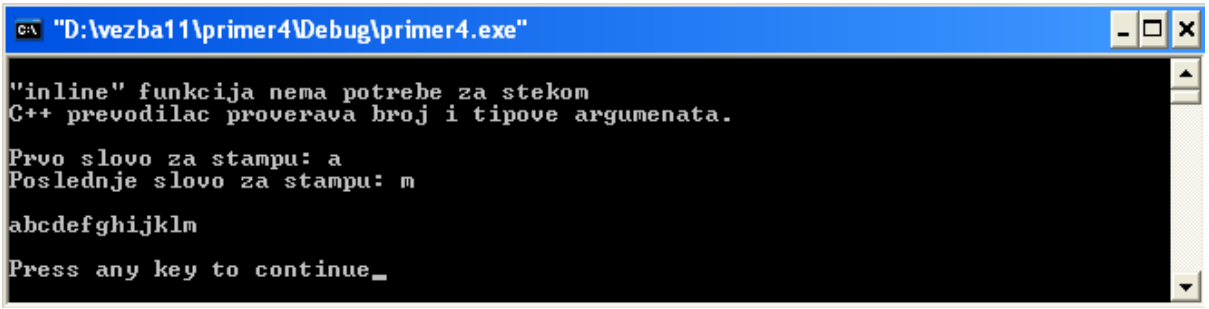
Главна функција тражи (слике 8.4 а) и 8.4 б)) у једној *do..while* петљи (линије од 15. до 20.) прво и последње слово за штампу, а онда у *for* петљи (линије 23. и 24.), у којој може бити велики број итерација, позива функцију *stampa_slovo().* Резултат програма може се видети на слици 8.4 в).



Слика 8.4 а) Излаз програма **primer8_4** (први део)



Слика 8.4 б) Излаз програма **primer8_4** (други део)



Слика 8.4 в) Излаз програма **primer8_4** (трећи део)

Изворни кôд програма **primer8_5**

```
1:      //primer8_5.cpp
2:      //Poziv funkcije za izmenu zadate strukture podataka na tri nacina:
3:      //po vrednosti, po referenci pomocu pokazivaca
4:      //i po referenci pomocu reference.
5:      #include <iostream>
6:      #define MAX     30                  //broj korisnih karaktera u stringu
7:      using namespace std;
8:
9:      struct student{
10:             char ime[MAX+1];
11:             int reg_broj;
12:             int god_upisa;
13:             int polozenih;
14:             };
15:
16:     void po_vrednosti(student x);
17:     void po_ref_pok(student *ptr_x);
18:     void po_ref_ref(student &ref_x);
19:
20:     int main()
21:     {       student s1;
22:
23:             s1.polozenih = 10;
24:             cout << "Vrednost broja, clana strukture\n\n";
25:             cout << "na pocetku: " << s1.polozenih << endl;
26:
27:             //Predaja funkciji kopije strukture
28:             po_vrednosti(s1);
29:             cout << "posle predaje funkciji strukture po vrednosti:\t";
30:             cout << s1.polozenih << endl;
31:
32:             //Predaja funkciji pokazivaca na strukturu
33:             po_ref_pok(&s1);
34:             cout << "posle predaje funkciji pokazivaca na strukturu:\t";
35:             cout << s1.polozenih << endl;
36:
37:             //Predaja funkciji reference strukture
38:             po_ref_ref(s1);
39:             cout << "posle predaje funkciji reference na strukturu:\t";
40:             cout << s1.polozenih << “\n\n”;
41:
42:             return 0;
43:     }
44:     //Funkcija koja dobija strukturu po vrednosti
45:     void po_vrednosti(student x)
46:     {       x.polozenih++;                  //sintaksa strukture
47:     }
```

```
48:  //Funkcija koja dobija pokazivac na strukturu
49:  void po_ref_pok(student *ptr_x)
50:  {      ptr_x->polozenih++;              //sintaksa pokazivaca
51:  }
52:  //Funkcija koja dobija referencu na strukturu
53:  void po_ref_ref(student &ref_x)
54:  {      ref_x.polozenih++;               //sintaksa reference
55:  }
```

Анализа програма **primer8_5**

Овај програм користи три начина предаје функцијама информација о члановима једне структуре.

У линијама од 9. до 14. дефинисан је тип структуре *student (ime, reg_broj, god_upisa, polozenih)* и декларисане су три функције: *po_vrednosti(), po_ref_pok(), po_ref_ref()* (линије од 16. до 18.).

Из дефиниција поменутих функција (линије од 45. до 55.) може се видети да свака од њих добија аргумент: прва структуру типа student *(x),* друга показивач на структуру типа *student (*ptr_x),* а трећа референцу на структуру типа student *(&ref_x).* Свака од ових функција има наредбу која инкрементира вредност једног члана структуре (*polozenih*).

Прва функција има наредбу *x.polozenih++;* којом не постиже измену вредности члана структуре, јер је у аргументу по вредности добила само копију садржаја структуре.

Друга функција има наредбу *ptr_x->polozenih++;* којом реализује измену вредности члана структуре јер је у аргументу по референци, помоћу показивача, добила адресу структуре у меморији.

Трећа функција има наредбу *ref_x.polozenih++;* којом реализује измену вредности члана структуре јер је у аргументу по референци, помоћу референце, добила адресу структуре. Овај начин (предаја референце) у односу на претходни (предаја показивача), има једноставнију синтаксу. Овде нема потребе за операторима: -> у наредби у којој се приступа члану структуре (користи се оператор тачка, као у линији 54.), као ни за оператором & у позиву функције (пише се само име структуре, као у линији 38.). У декларацији функције (линија 18.) пише се адресни оператор *(&ref_x).* Дакле, референца дефинише адресу, али не захтева оператор индиректног адресирања (*) и на тај начин поједностављује употребу показивача код функција.

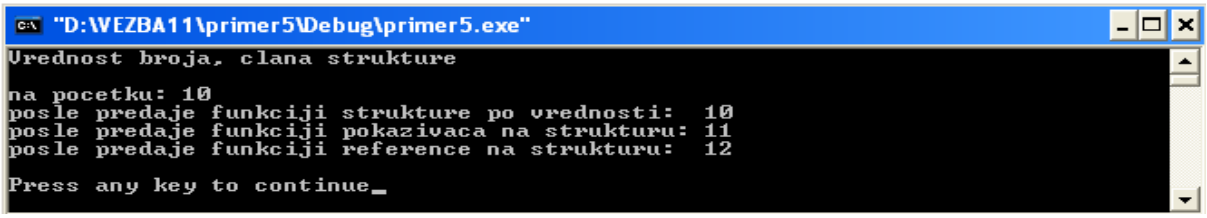
Главна функција има прво декларацију структуре дефинисаног типа *student (s1)*, иницијализацију њеног члана *s1.polozenih* на вредност 10 и приказ на екрану ове вредности (линије од 21. до 25.). У програмима на језику *C++,* декларација структуре дефинисаног типа *student* може се писати и на следећи начин: *student s1* (уместо *struct student s1*). Све што важи за структуре у програмима на језику *C* важи и овде.

Функцији *po_vrednosti()* предаје се копија садржаја стварне структуре *s1* и приказује се на екрану вредност њеног члана *s1.polozenih* (линије од 28. до 30.). Функцији *po_ref_pok()* предаје се показивач на стварну структуру *s1* и приказује се на екрану вредност њеног члана *s1.polozenih* (линије од 33. до 35.).

Функцији *po_ref_ref()* предаје се референца на стварну структуру *s1* и приказује на екрану вредност њеног члана *s1.polozenih* (линије од 38. до 40.).

Иако позиви функција у линијама 28. и 38. (*po_vrednosti(s1)*; и *po_ref_ref(s1)*;) на месту стварног аргумента имају име структуре *s1,* немају аргументе истог типа (аргумент прве наведене функције је примерак структуре, а друге референца на структуру, јер је друга функција тако декларисана).

На слици 8.5 може се видети да позив прве функције не мења садржај структуре, а да је на било који од преостала два начина измена изведена.



Слика 8.5 Излаз програма **primer8_5**

Изворни кôд програма **primer8_6**

```
1:      //primer8_6.cpp
2:      //Odredjivanje najveceg broja u zadatom dinamickom nizu celih brojeva
3:      #include  <iostream>
4:      #include <stdlib.h>
5:      #define MAX     50                  //dozvoljena duzina niza
6:      using namespace std;
7:
8:      int main( )
9:      {       int i, n, max, *niz;
10:
11:             do
12:             {       cout << "\nBroj elemenata niza: ";
13:                     cin >> n;
14:             }while(n<1 || n>MAX);
15:
16:             niz = new int[n];
17:
18:             if(niz==NULL)
19:             {       cerr << "Nije uspela dodela memorije\n\n";
20:                     exit(1);
21:             }
22:
23:             cout << "\nUnesite vrednosti elemenata niza: ";
24:             for(i=0; i<n; i++)
25:                     cin >> niz[i];
26:
27:             max = niz[0];
28:
29:             for(i=1; i<n; i++)
30:                     if(niz[i]>max)
31:                             max = niz[i];
32:
33:             cout<< "\nNajveca vrednost zadatog niza je: " << max << "\n\n";
```

```
34:
35:             delete niz;
36:
37:             return 0;
38:     }
```

Анализа програма **primer8_6**

Као и у програмима на језику *C*, и овде се показивачи користе код динамичке доделе меморије током извршавања програма. За разлику од језика *C*, који користи стандардне функције, језик *C++* за динамичку доделу меморије и ослобађање динамички додељене меморије уводи операторе *new* и *delete,*.

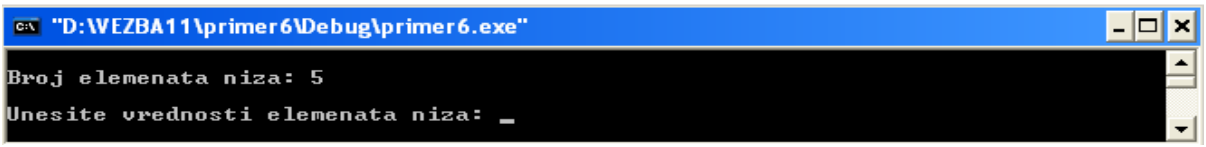
Оператор *new* користи се над изразом који треба да садржи број бајтова које треба доделити за објекат у меморији и даје као резултат показивач на почетак додељене меморије или *NULL* показивач (ако није успела додела).

Оператор *delete* користи се над адресом динамички додељеног простора у меморији, који треба да се ослободи.

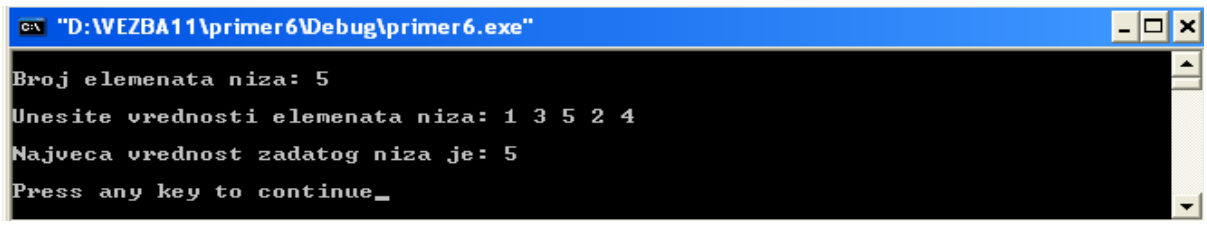
Овај програм илуструје динамичку доделу меморије за низ целих бројева. У линији 9. декларисан је показивач **niz* на тип *int*, који ће бити искоришћен као адреса почетка целобројног низа истог типа.

Програм прво тражи број елемената низа (слика 8.6 а)) и после учитаног овог броја са тастатуре (линије од 11. до 14.) тражи доделу меморије за низ тако што оператору *new* пружа име типа као и број елемената низа (у средњим заградама после наведеног типа) (линија 16.). Резултат примене овог оператора додељује се показивачу *niz*. Ако није додељена меморија вредност овог показивача је *NULL*, порука о томе иде на екран и прекида се програм (линије од 18. до 21.). У супротном, ако је додељена меморија, траже се са тастатуре (слика 8.6 б)) и читају сами елементи низа (линије од 23. до 25.), одређује највећи елемент у учитаном низу (линије од 27. до 31.) и приказује његова вредност на екрану (слика 8.6 в)). На крају, програм помоћу оператора *delete* ослобађа додељену меморију за низ тако што оператору пружа име показивача на почетак низа (линија 35.).

Слика 8.6 а) Излаз програма **primer8_6** (први део)



Слика 8.6 б) Излаз програма **primer8_6** (други део)



Слика 8.6 в) Излаз програма **primer8_6** (трећи део)

Изворни кôд програма **primer8_7**

```
1:      //primer8_7.cpp
2:      //Dinamicka dodela memorije za zadatu strukturu  podataka
3:      #include <iostream>
4:      #include <stdlib.h>
5:      #define MAX 15              //broj korisnih karaktera u stringu
6:      using namespace std;
7:
8:      struct  student{
9:              char ime[MAX+1];
10:             char prezime[MAX+1];
11:             int reg_broj;
12:             int god_upisa;
13:             };
14:
15:     int main()
16:     {       student *ptr_s;
17:
18:             ptr_s = new student;
19:
20:             if(ptr_s==NULL)
21:             {       cerr << "Nije uspela dodela memorije\n";
22:                     exit(1);
23:             }
24:
25:             cout << "\nIme, prezime, registarki broj i godina upisa:\n";
26:             cin >> ptr_s->ime >> ptr_s->prezime;
27:             cin >> ptr_s->reg_broj >> ptr_s->god_upisa;
28:
29:             cout << "\nUneti podaci:\n";
30:             cout << ptr_s->ime << " " << ptr_s->prezime << " ";
31:             cout << ptr_s->reg_broj << " " << ptr_s->god_upisa << "\n\n";
32:
33:             delete ptr_s;
34:
35:             return 0;
36:     }
```

Анализа програма **primer8_7**

Програм у овом примеру додељује и ослобађа меморију за структуру података, иницијализовану са тастатуре.

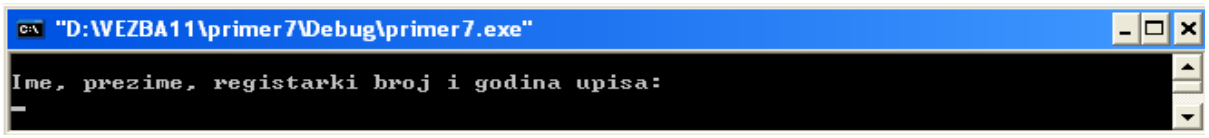
У линијама од 8. до 13. дефинисан је тип структуре *student* (*ime, prezime, reg_broj* и *god_upisa*).

На почетку главна функција декларише показивач **ptr_s* на структуру типа *student*, оператору *new* пружа име типа структуре (из кога је познат потребан број бајтова) и добија од њега адресу почетка додељене меморије (линија 18.)

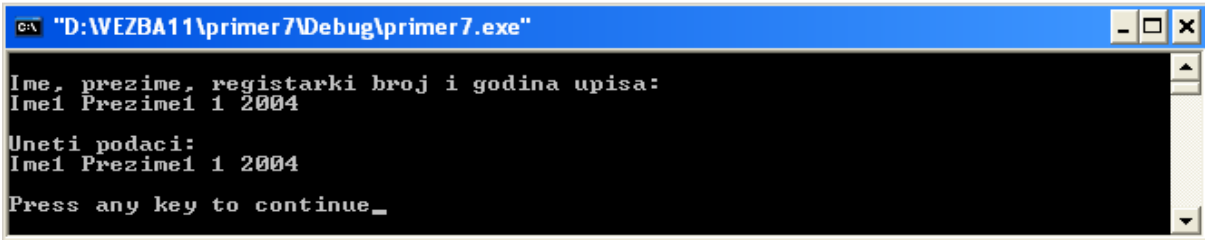
или *NULL* показивач (линије од 20. до 23.). Програм даље тражи са тастатуре (слика 8.7 а)) податке и учитава их у одговарајуће чланове струкетуре:

ptr_s->ime,
ptr_s->prezime,
ptr_s->reg_broj,
ptr_s->god_upisa,

приказује вредности ових података на екрану (слика 8.7 б)) и ослобађа додељену меморију (линија 33.).



Слика 8.7 а) Излаз програма **primer8_7** (први део)



Слика 8.7 б) Излаз програма **primer8_7** (други део)

Изворни кôд програма **primer8_8**

```
1:      //primer8_8.cpp
2:      //Prikaz na ekranu podataka zadatog dinamickog niza struktura
3:      #include <iostream>
4:      #include <stdlib.h>
5:      #define BROJ        2           //broj adresa (struktura u nizu)
6:      #define MAX         30          //broj korisnih karaktera u stringu
7:      using namespace std;
8:
9:      struct  adresa{
10:             char ime[MAX+1];
11:             char ulica[MAX+1];
12:             int broj;
13:             char grad[MAX+1];
14:             };
15:
16:     void prikaz_adresa(struct adresa adrese[], int n);
17:     void greska(void);
18:
19:     int main()
20:     {       adresa *adrese;
21:
22:             adrese = new adresa[BROJ];
23:
24:             if(adrese==NULL)
```

```
25:                     greska();
26:
27:             cout << "\n==============UNOS ADRESA ============== \n";
28:             for(int i=0; i<BROJ; i++)
29:             {       cout << "\nIme i prezime: ";
30:                     cin.getline(adrese[i].ime, MAX+1);
31:
32:                     cout << "Ulica: ";
33:                     cin.getline(adrese[i].ulica, MAX+1);
34:
35:                     cout << "Broj: ";
36:                     cin >> adrese[i].broj;
37:                     cin.ignore();
38:
39:                     cout << "Grad: ";
40:                     cin.getline(adrese[i].grad, MAX+1);
41:             }
42:
43:             prikaz_adresa(adrese, BROJ);
44:
45:             delete adrese;
46:
47:             return 0;
48:     }
49:     //Funkcija koja prikazuje na ekranu sadrzaj niza struktura
50:     void prikaz_adresa(struct adresa adrese[], int n)
51:     {       cout << "\n================ ADRESE ================ \n";
52:             for(int i=0; i<BROJ; i++)
53:             {       cout << "\nIme:\t\t" << adrese[i].ime << endl;
54:                     cout << "Ulica i broj:\t" << adrese[i].ulica << " ";
55:                     cout << adrese[i].broj << endl;
56:                     cout << "Grad :\t\t" << adrese[i].grad << endl;
57:             }
58:             cout << "\n";
59:     }
60:     //Funkcija koja prikazuje na ekranu poruke o greskama
61:     void greska(void)
62:     {       cerr << "Nije uspela dodela memorije\n";
63:             exit(1);
64:     }
```

Анализа програма **primer8_8**

Програм у овом примеру додељује и ослобађа меморију за низ структура података, иницијализован са тастатуре.

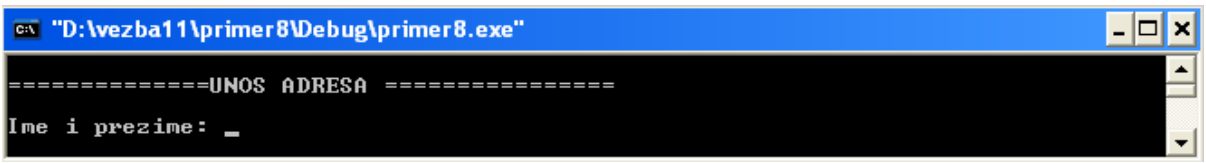
На почетку (линије од 9. до 14.) програм дефинише тип структуре *adresa* (*ime, ulica, broj* и *grad*), као и функције *prikaz_adresa()* и *greska().*

Функција *prikaz_adresa()* (дефиниција у линијама од 50. до 59.) добија од остатка програма показивач на низ структура типа *adresa* и број елемената низа и у једној *for* петљи приказује на екрану садржај овог низа структура.

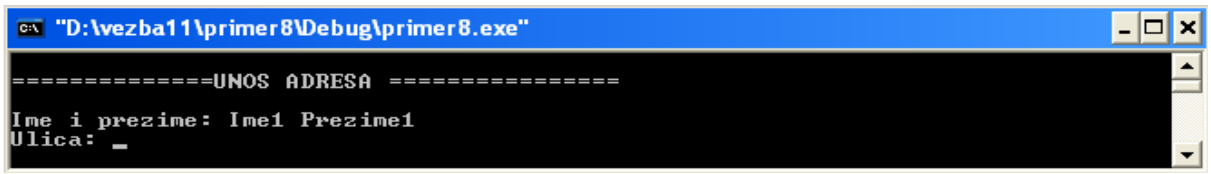
Функција *greska(void)* шаље поруку на екран и прекида извршавање програма (дефиниција у линијама од 61. до 64.).

Главна функција програма прво декларише показивач **adrese* на структуру типа *adresa*, који ће бити коришћен као адреса низа структура истог типа. Онда тражи доделу меморије за низ тако што пружа оператору *new* име типа структуре и у средњим заградама, после имена број структура у низу, а добија адресу додељеног простора (линија 22.) или *NULL* показивач, ако није успела додела (линије 24. и 25.). Програм даље тражи унос преко тастатуре (слика 11.8 а)) свих потребних података за сваку структуру у низу (линије од 27. до 41.), позива функцију *prikaz_adresa()* (линија 43.) и ослобађа додељену меморију за низ структура (линија 45.).

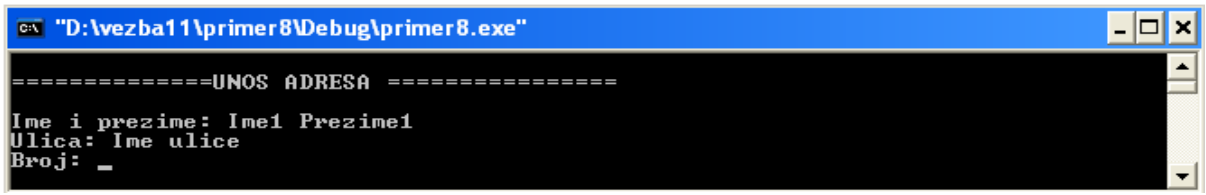
На сликама од 8.8 а) до 8.8 ђ) може се видети излаз програма за један задати комплет адреса, задат преко тастатуре.



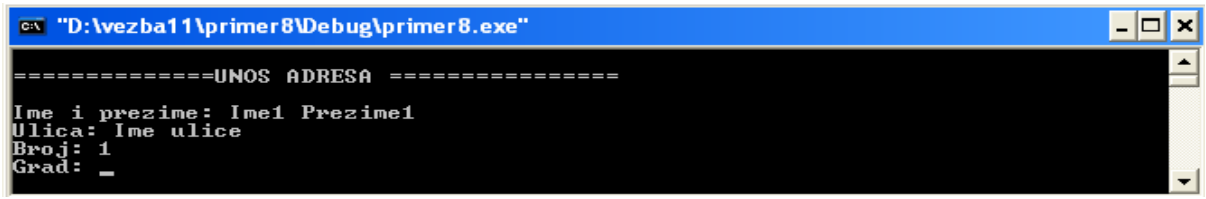
Слика 8.8 а) Излаз програма **primer8_8** (први део)



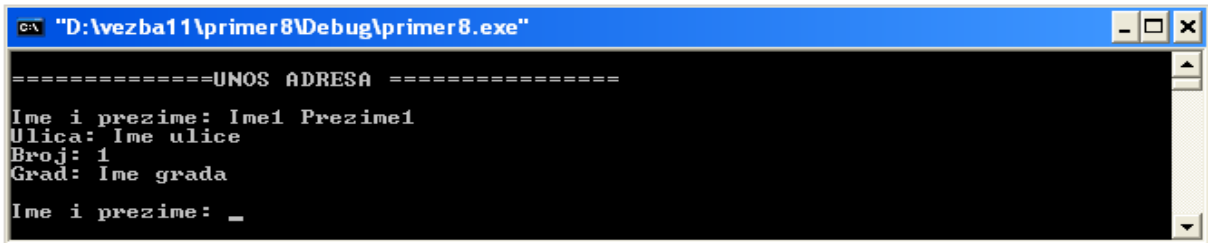
Слика 8.8 б) Излаз програма **primer8_8** (други део)



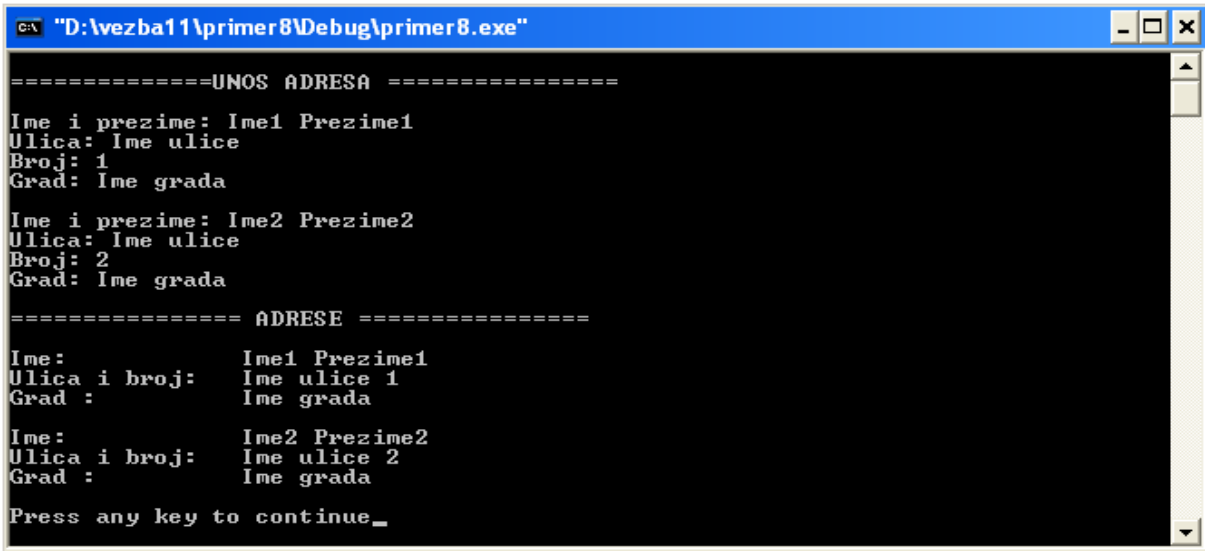
Слика 8.8 в) Излаз програма **primer8_8** (трећи део)



Слика 8.8 г) Излаз програма **primer8_8** (четврти део)



Слика 8.8 д) Излаз програма **primer8_8** (пети део)



Слика 8.8 ђ) Излаз програма **primer8_8** (шести део)

Практични савети

- ✓ Писање функција са подразумеваним вредностима аргумената може бити корисно у програмима.
- ✓ Писање речи *inline* испред имена функције само је захтев преводиоцу да функцију угради у кôд.

Евентуалне грешке

- ➢ Функције које примају аргументе по вредности добијају само копије тих вредности. Функције које примају аргументе по референци (било да се ради о запису помоћу показивача или помоћу референце) добијају стварне адресе променљивих у оперативној меморији и могу мењати садржаје тих променљивих.

Задаци за припрему вежбе 8. у лабораторији

Задатак 1.

Написати програм на језику *C++* који од датума, задатих у облику три броја *(dan, mesec* и *godina)* приказује на екрану списак датума у следећем облику:

Januar 1. 2004.
Februar 1. 2004.
:

Креирати тип структуре чији су чланови: *redni_broj_meseca* и *ime_meseca*. Урадити програм помоћу динамичке доделе меморије за структуре.

Задаци за вежбу 8. у лабораторији

Задатак 1.

Написати програм на језику *C++* који у зависности од опције (*i, l* или *d*) задате преко тастатуре одређује највећи и најмањи елемент низа типа int, long или *double*. Низ се задаје преко тастатуре и дужине је *n (n≤50).* Користити одговарајуће преклопљене функције.

Задатак 2.

Написати програм на језику *C++* који помоћу одговарајуће функције од низа међусобно различитих *n (n≤50)* целих бројева, задатих преко тастатуре, формира и приказује на екрану опадајући или растући низ, у зависности од задате опције (O/R). Ако није изабрана ни једна од понуђених опција програм треба да позива функцију са једним аргументом мање, а функција да користи подразумевану вредност аргумената и прикаже на екрану неизмењени низ.

Урадити следеће програме помоћу динамичке доделе меморије. Издвојити подзадатке програма у посебне функције.

Задатак 3.

Написати програм на језику *C++* који

а) Приказује на екрану опције:

1. Citanje niza
2. Sazimanje niza
3. Prosirenje niza
4. Prikaz niza
5. Izlaz

све док се не изабере опција 5.

б) за изабрану опцију 1. чита са тастатуре низ целих бројева, дужине *n (n≤20),*

в) за изабрану опцију 2. избацује из задатог низа све нула елементе,

г) за изабрану опцију 3. додаје на крај задатог низа нови низ, дужине *n (n≤20),* прочитан са тастатуре,

д) за изабрану опцију 4. приказује на екрану низ.

Задатак 4.

Написати програм на језику *C++* који:

а) креира списак података о становима којима располаже једна агенција: *ime_vlasnika, adresa (ulica* и *broj), broj_soba, broj_kvadrata* и *cena*. Списак може имати податке за највише 100 станова,

б) приказује на екрану, у табели, списак свих података о становима који имају тражени број соба и број квадрата у траженом опсегу,

в) приказује на екрану податке о свим становима у задатој улици.

Вежба 9.

Рад са датотекама у програмима на језику *C++*

Текстуалне и бинарне датотеке

Стандардне класе за рад са датотекама

Креирање објеката за повезивање са датотекама

Позиционирање у датотекама

Размена података са датотекама

Примери

Изворни кôд програма **primer9_1**

```
1:      //primer9_1.cpp
2:      //Kopiranje sadrzaja jedne tekst datoteke u drugu,
3:      //uz  konverziju svih velikih slova u ista mala
4:      #include <iostream>
5:      #include <fstream>
6:      #include <stdlib.h>
7:      #include <ctype.h>
8:      #define MAX     30                  //broj korisnih karaktera u stringu
9:      using namespace std;
10:     int main()
11:     {       int c;
12:             char ime1[MAX+1], ime2[MAX+1];
13:
14:             ifstream fin;                       //Kreiranje ulaznog toka
15:             ofstream fout;                      //Kreiranje izlaznog toka
16:
17:             //Zadavanje imena datoteke za citanje
18:             cout << "\nIme originalne tekst datoteke: ";
19:             cin >> ime1;
20:
21:             //Otvaranje datoteke za citanje
22:             fin.open(ime1, ios::in | ios::nocreate);
23:
24:             //Izvestaj o neuspesnom otvaranju za citanje ako datoteka ne postoji
25:             if(!fin)
26:             {       cerr << "\nNEMOGUCE CITANJE";
27:                     cerr << "\nDatoteka sa zadatim imenom ne postoji!\n";
28:                     exit(1);
29:             }
30:
31:             //Zadavanje imena datoteke za upis
32:             cout << "Ime datoteke kopije: ";
33:             cin >> ime2;
34:
35:             //Otvaranje datoteke za upis
36:             fout.open(ime2,ios::out | ios::noreplace);
37:
38:             //Izvestaj o neuspesnom otvaranju za upis ako datoteka vec postoji
39:             if(!fout)
40:             {       cerr << "\nNEDOZVOLJEN UPIS";
41:                     cerr << "\nDatoteka sa zadatim imenom vec postoji!\n";
42:                     exit(1);
43:             }
44:
```

```
45:             //Kopiranje uz konverziju velikih slova u mala
46:             while((c = fin.get()) != EOF)
47:             {       fout.put((char)tolower(c));
48:             }
49:             cout << "\nUSPESNO IZVRSENO KOPIRANJE\n";
50:
51:             //Zatvaranje otvorenih datoteka za citanje i upis
52:             fin.close();
53:             fout.close();
54:
55:             return 0;
56:     }
```

Анализа програма **primer9_1**

За повезивање датотеке са програмом на језику *C++* потребно је дефинисати објекат једне од следеће три класе:

- *fstream* – класа изведена из класе *iostream* (када се датотека отвара за упис и читање),
- *ifstream* – класа изведена из класе *istream* (када се датотека отвара само за читање),
- *ofstream* – класа изведена из класе *ostream* (када се датотека отвара само за упис).

Код класа је заступљено наслеђивање. Свака од поменутих класа (*fstream, ifstream* и *ofstream*) садржи све функције *(get(), put()…)* и операторе (улаза са тастатуре (>>), излаза на екран (<<)…) класе из које је изведена (*iostream, istream* и *ostream)*.

За размену података са датотекама користе се истоимене функције као и у *C* програмима само што су сада ове функције чланице поменутих класа. Због тога је у овим програмима неопходно декларисати објекте класа и написати позиве функција чланица над објектима ових класа.

Програм у овом примеру копира садржај једне текст датотеке у другу и при томе конвертује сва велика слова у иста мала.

Заглавље *fstream* укључује за рад са датотекама (линија 5.). У линијама 14. и 15. декларисани су: објекат *fin* класе *ifstream* и објекат *fout* класе *ofstream.*

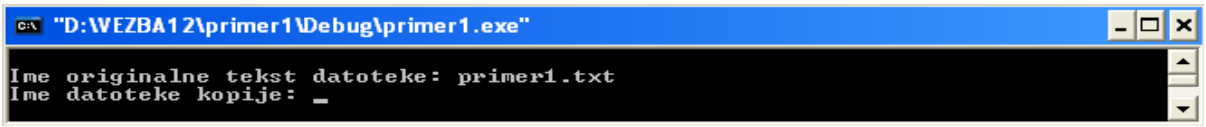
После учитаног са тастатуре (слика 9.1 а)) имена прве датотеке и смештања овог имена у низ карактера *ime1* (линије 18. и 19.), програм позива функцију *open()* над објектом *fin* (линија 22.), за отварање датотеке у режиму *ios::in* (отвара за читање) и *ios::nocreate* (нема креирања, ако датотека не постоји). На овај начин програм повезује објекат *fin* са датотеком чије име чува *ime1*. Ако није успело повезивање објекта са датотеком (на пример датотека не постоји) следи порука на екрану и прекид програма (линије од 25. до 29.).

После учитаног са тастатуре (слика 9.1 б)) имена друге датотеке и смештања овог имена у низ карактера *ime2* (линије 32. и 33.), програм позива функцију *open* над објектом *fout* (линија 36.) за отварање датотеке у режиму *ios::out* (за упис) и *ios::noreplace* (нема преписивања садржаја ако датотека постоји). На овај начин програм повезује објекат *fout* са датотеком чије име чува *ime2*. Ако није успело повезивање објекта са датотеком (на пример датотека већ постоји) следи порука на екрану и прекид програма (линије од 39. до 43.).

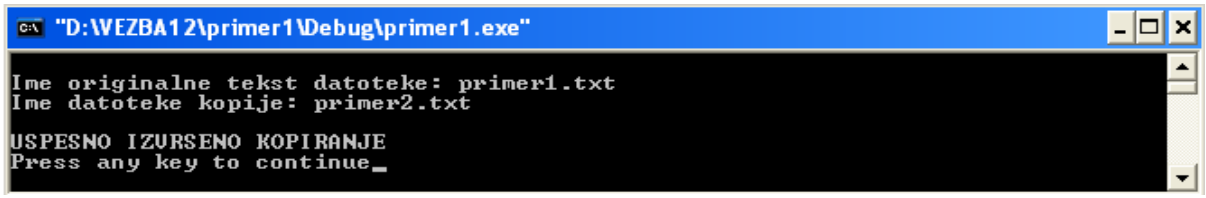
У једној *while* петљи (линије од 46. до 48.) програм чита из датотеке, која је отворена за читање, један по један знак, до знака *EOF* (применом функције *get()* над објектом *fin* и приказује на екрану сваки учитани знак, са конверзијом великог слова у исто мало (применом функције *put()* над објектом *fout*).

У линијама 52. и 53. написани су позиви функције *close()* над објектима *fin* и *fout*, за затварање отворених датотека. На слици 9.1 в) може се видети извештај који програм приказује на екрану, после успешно изведеног копирања. На сликама 9.1 г) и 9.1 д) могу се видети садржаји једне оригиналне датотеке и њене копије.

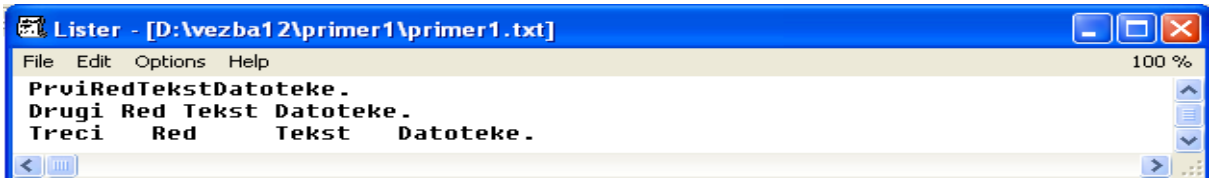
Слика 9.1 а) Излаз програма **primer9_1** (први део)



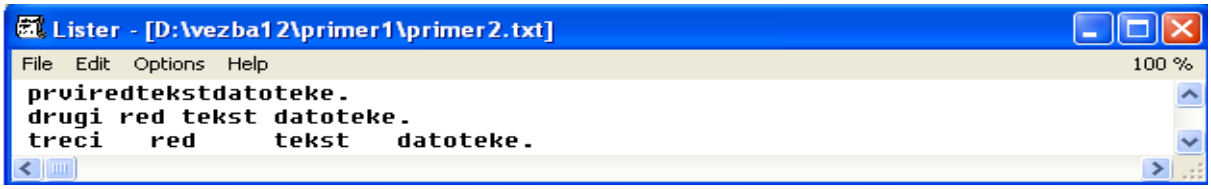
Слика 9.1 б) Излаз програма **primer9_1** (други део)



Слика 9.1 в) Излаз програма **primer9_1** (трећи део)



Слика 9.1 г) Излаз програма **primer9_1** (четврти део)



Слика 9.1 д) Излаз програма **primer9_1** (пети део)

Изворни кôд програма **primer9_2**

```
1:      //primer9_2.cpp
2:      //Kopiranje sadrzaja jedne tekst datoteke u drugu dodavanjem
3:      //rednog broja linije, i u trecu dopisivanjem na kraj postojeceg sadrzaja
4:      #include <iostream>
5:      #include <fstream>
6:      #include <stdlib.h>
7:      #define MAX     80          //broj korisnih karaktera u stringu
8:      using namespace std;
```

```
9:      enum Greska{GRESKA_CITANJA, GRESKA_UPISA};
10:     char *poruke_o_greskama[] = {
11:             "\nGreska pri otvaranju datoteke za citanje!\n",
12:             "\nGreska pri otvaranju datoteke za upis!\n"      };
13:
14:     void greska(Greska status);
15:
16:     int main()
17:     {       char red[MAX+1];
18:             int rbr = 1;
19:
20:             //Kreiranje ulazno/izlaznih tokova
21:             fstream f1, f2, f3;
22:
23:             //Otvaranje datoteka za kopiranje sadrzaja
24:             f1.open("primer1.txt", ios::in | ios::nocreate);
25:             if(!f1)
26:                     greska(GRESKA_CITANJA);
27:
28:             f2.open("primer2.txt", ios::out);
29:             if(!f2)
30:                     greska(GRESKA_UPISA);
31:
32:             //Kopiranje uz dodavanje rednog broja na pocetak svake linije
33:             while(!f1.eof())
34:             {       f1.getline(red, MAX+1);
35:                     f2 << rbr++ << ": " << red << endl;
36:             }
37:             cout << "\nUSPESNO IZVRSENO KOPIRANJE\n";
38:
39:             //Zatvaranje otvorenih datoteka
40:             f1.close();
41:             f2.close();
42:
43:             //Otvaranje datoteka za dopisivanje sadrzaja
44:             f1.open("primer1.txt", ios::in | ios::nocreate);
45:             if(!f1)
46:                     greska(GRESKA_CITANJA);
47:
48:             f3.open("primer3.txt", ios::out | ios::app);
49:             if(!f3)
50:                     greska(GRESKA_UPISA);
51:
52:             //Dopisivanje sadrzaja
53:             while(!f1.eof())
54:             {       f1.getline(red, MAX+1);
55:                     f3 << "\n" << red;
56:             }
57:             cout<<"\nUSPESNO IZVRSENO DOPISIVANJE\n\n";
58:
```

```
59:             //Zatvaranje otvorenih datoteka
60:             f1.close();
61:             f3.close();
62:
63:             return 0;
64:     }
65:     //Funkcija koja prikazuje na ekranu poruke o greskama
66:     void greska(Greska status)
67:     {       cerr << poruke_o_greskama[status];
68:             exit(1);
69:     }
```

Анализа програма **primer9_2**

Овај програм има задатак да садржај једне текст датотеке копира у другу тако што ће на почетак сваког реда додати редни број и знак двотачка.

У програмима на језику *C++* нема потребе да се пише службена реч *typedef* уз службену реч *enum*, због тога што се овде идентификатор набрајања *enum* користи као идентификатор типа и без речи *typedef* (линија 9.). Идентификатор *typedef* може се изоставити и код дефинисања типа структуре.

Програм прво декларише објекте *f1, f2* и *f3* да би их користио за три предвиђене датотеке (линија 21.).

Следи повезивање објеката *f1* и *f2* са датотекама *"primer1.txt"* и *"primer2.txt",* респективно. Ако ово успе датотека *"primer1.txt"* отворена је у режиму *ios::in* (за читање) и *ios::nocreate* (нема креирања, ако не постоји), а датотека *"primer2.txt"* у режиму *ios::out* (за упис) (линије од 24. до 30.).

У једној *while* петљи програм испитује статус датотеке повезане са објектом *f1,* тако што проверава да ли је вредност функције *eof()* над објектом *f1* једнака 0 (линија 33.). Све док је тако није откривен крај датотеке и помоћу функције *getline()* чита се сваки ред текста (или првих *MAX* знакова реда) из ове датотеке (линија 34.) и уписује у датотеку која је повезана са објектом *f2* (линија 35.). Када се открије крај прве датотеке прекида се петља, приказује на екрану извештај (линија 37.) и затварају отворене датотеке (линије 40. и 41.).

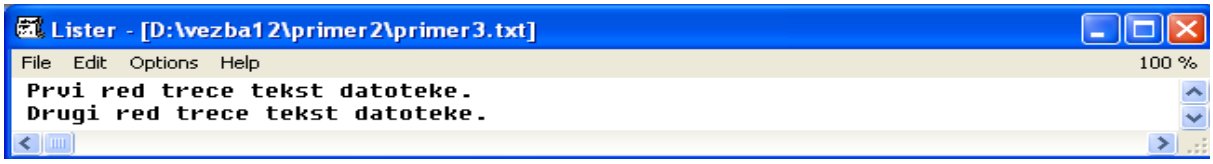
Програм онда повезује објекте *f1* и *f3* са датотекама *"primer1.txt"* и *"primer3.txt",* респективно. Ако ово успе, датотека *"primer1.txt"* отворена је у режиму *ios::in* (за читање) и *ios::nocreate* (нема креирања ако не постоји), а датотека *"primer2.txt"* у режиму *ios::out* (за упис) и *ios::app* (упис иде од краја постојећег садржаја датотеке) (линије од 44. до 50.).

У другој *while* петљи програм испитује статус датотеке повезане са објектом *f1* тако што проверава да ли је вредност функције *eof()* над објектом *f1* једнака 0 (линија 53.). Све док је тако није откривен крај датотеке и помоћу функције *getline()* чита се сваки ред текста (или првих *MAX* знакова реда) из ове датотеке (линија 54.) и дописује на крај постојећег садржаја у датотеку, која је повезана са објектом *f3* (линија 55.). Када се открије крај прве датотеке прекида се петља, приказује на екрану извештај (линија 57.) и затварају отворене датотеке (линије 60. и 61.).

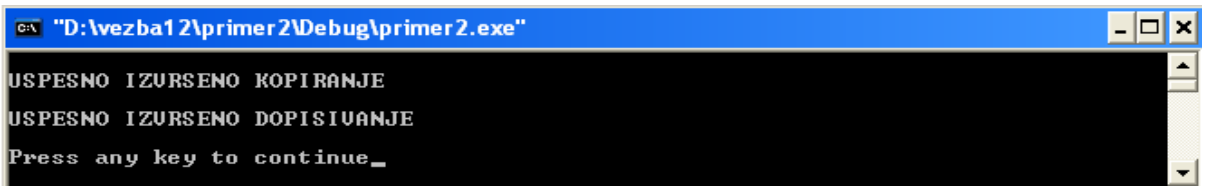
На сликама 9.2 а), 9.2 б) и 9.2в) могу се видети садржаји од три датотеке пре пуштања програма, на слици 9.2 г) извештај од програма о успешно извршеном копирању и на сликама 9.2 д) и 9.2 ђ) садржаји датотека у које је копиран и дописан садржај из оригиналне датотеке.

Слика 9.2 а) Излаз програма **primer9_2** (први део)

Слика 9.2 б) Излаз програма **primer9_2** (други део)



Слика 9.2 в) Излаз програма **primer9_2** (трећи део)



Слика 9.2 г) Излаз програма **primer9_2** (четврти део)

Слика 9.2 д) Излаз програма **primer9_2** (пети део)

Слика 9.2 ђ) Излаз програма **primer9_2** (шести део)

Изворни кôд програма **primer9_3**

```
1:      //primer9_3.cpp
2:      //Upis u tekst datoteku / citanje  iz tekst datoteke stringova i
3:      //prikaz na ekranu i izmene vrednosti tekuce pozicije u istoj datoteci
4:     #include <fstream>
5:      #include <fstream.h>
6:      #include <stdlib.h>
7:      #include <string.h>
8:      #define MAX1        4              //broj korisnih karaktera u jednom stringu
9:      #define MAX2        7              //broj korisnih karaktera u drugom stringu
10:    using namespace std;
```

```
11:     int main()
12:     {       char string1[MAX1+1] = "Test";
13:             char string2[MAX2+1] = "Program";
14:             char string[MAX1+MAX2+1];
15:
16:             fstream f;
17:
18:             f.open("test.txt", ios::out | ios::in);
19:             if(!f)
20:             {       cerr << "\nGreska pri otvaranju datoteke\n";
21:                     exit(1);
22:             }
23:
24:             cout << "\nUPIS PRVOG STRINGA";
25:             for(int i = 0; string1[i] != '\0'; i++)
26:             {       f.put(string1[i]);
27:                     cout << "\nUpisan znak: " << (char)string1[i];
28:                     cout <<"\t\t\tTekPoz: " << f.tellp();
29:             }
30:
31:             cout << "\n\nUPIS DRUGOG STRINGA: ";
32:             f.write(string2,strlen(string2));
33:             cout << string2 << ",\tTekPoz: " << f.tellp();
34:
35:             cout << "\n\nPOVRATAK NA POCETAK";
36:             f.seekp(0);
37:             cout << ",\t\tTekPoz: " << f.tellp();
38:
39:             cout << "\n\nCITANJE DATOTEKE: ";
40:             f.getline(string, MAX1+MAX2+1);
41:             cout << string;
42:             cout << ",\tTekPoz: " << f.tellp();
43:
44:             cout << "\n\nPOMERAJ " << MAX1 << " OD POCETKA";
45:             f.seekp(MAX1, ios::beg);
46:             cout << ",\t\tTekPoz: " << f.tellp();
47:
48:             cout << "\n\nPOMERAJ " << MAX2 << " OD TRENUTNE POZICIJE";
49:             f.seekp(MAX2, ios::cur);
50:             cout << ",\tTekPoz: " << f.tellp();
51:
52:             cout << "\n\nPOMERAJ " << -(MAX1+MAX2) << " OD KRAJA";
53:             f.seekp(-(MAX1+MAX2), ios::end);
54:             cout << ",\t\tTekPoz: " << f.tellp() << "\n\n";
55:
56:             f.close();
57:             return 0;
58:     }
```

Анализа програма **primer9_3**

Овај програм садржи упис података у текст датотеку, читање података из датотеке, као и праћење и приказ на екрану текуће позиције у датотеци.

У програмима на језику *C++* постоје комплети стандардних функција за читање и измене текуће позиције у датотеци, у зависности од тога да ли следи читање или упис. Текућу позицију у датотеци за читање даје функција *tellg()*, а нову поставља функција *seekg().* За упис у датотеку текућу позицију даје функција *tellp(),* а нову поставља функција *seekp().*

Програм прво декларише потребне низове карактера (линије 12., 13. и 14.), а онда и објекат *f* класе *fstream* (линија 16.), који може бити повезан са датотеком за упис и за читање. Следи повезивање објекта *f* са датотеком *"test.txt"*. Ако ово успе датотека *"test.txt"* отворена је у режиму *ios::out* | *ios::in* (за упис и читање) (линије од 18. до 22.).

Садржај првог низа карактера *string1* програм уписује у отворену датотеку: знак по знак. При упису сваког знака позива функцију *tellp()* над објектом *f* да прикаже на екрану померај текуће позиције (линије од 24. до 29.).

Садржај другог низа карактера *string2* програм уписује у исту датотеку од текуће позиције, позивом једне функције *write()* над објектом *f*. После тога, позива функцију *tellp()* над објектом *f* да прикаже на екрану како се текућа позиција у датотеци увећала за број уписаних знакова (линије од 31. до 33.).

После позива функције *seekp(0)* над објектом *f* текућа позиција у датотеци враћена је на нулу. У овом случају користи се функција *seekp()* само са једним аргументом. После тога програм позива функцију *tellp()* над објектом *f*, да прикаже на екрану текућу позицију у датотеци (линије од 35. до 37.).

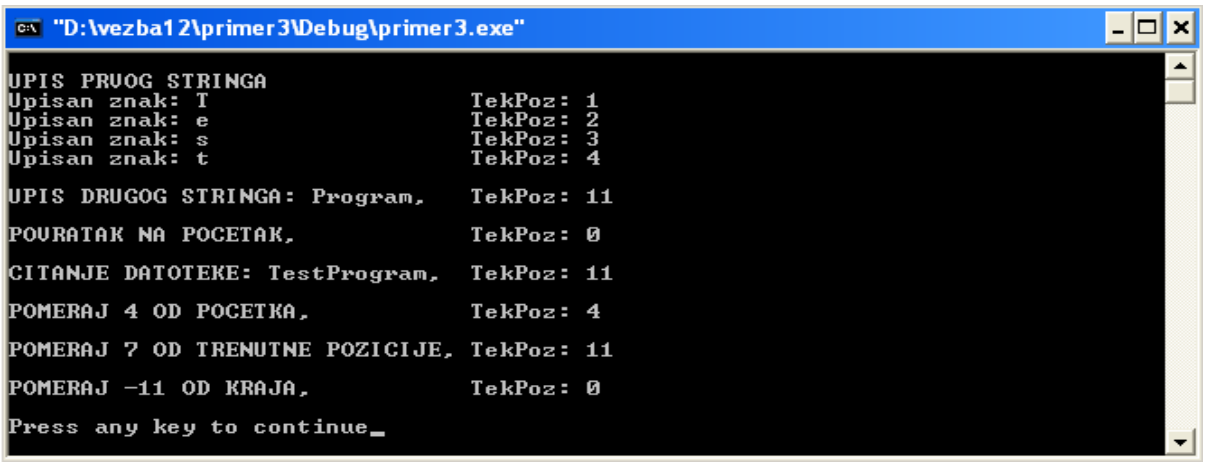
После читања *MAX1+MAX2* знакова из датотеке, помоћу функције *getline()* над објектом *f*, текућу позицију у датотеци даје позив функције *tellp()* над објектом *f* и то је број прочитаних знакова(линије од 39. до 42.).

После позива функције *seekp(MAX1,ios::beg)* над објектом *f*, текућа позиција у датотеци померена је за *MAX1* од почетка *(ios::beg).*

После позива функције *seekp(MAX2,ios::cur)* над објектом *f*, текућа позиција у датотеци померена је за *MAX2* од тренутног места *(ios::cur).*

После позива функције *seekp(-(MAX1+MAX2),ios::end)* над објектом *f*, текућа позиција у датотеци померена је за *–(MAX1+ MAX2)* од краја *(ios::end).*

Сваки од поменутих помераја праћен је позивом функције *tellp(),* за приказ текуће позиције на екрану и резултат се може видети на слици 9.3.



Слика 9.3 Излаз програма **primer9_3**

Изворни кôд програма **primer9_4**

```
1:      //primer9_4.cpp
2:      //Citanje iz binarne datoteke prethodno upisanog niza celih brojeva
3:      #include <iostream>
4:      #include <fstream>
5:      #include <stdlib.h>
6:      #define MAX     10              //dozvoljena duzina niza
7:      using namespace std;
8:      fstream fbin;
9:
10:     void upis(int niz[], int n);
11:     void prikaz_jednog(int niz[], int n);
12:     void prikaz_svih(int niz[], int n);
13:
14:     int main()
15:     {       int niz[MAX], opcija, izlaz = 0;
16:
17:             //Otvaranje binarne datoteke za upis i citanje
18:             fbin.open("primer.bin", ios::out | ios::in | ios::binary);
19:             if(!fbin)
20:             {       cout << "\nGreska pri otvaranju datoteke!\n";
21:                     exit(1);
22:             }
23:
24:             //Inicijalizacija i upis niza u binarnu datoteku
25:             upis(niz, MAX);
26:
27:             //Prikaz jednog ili svih elemenata niza (opciono) iz binarne datoteke
28:             while(1)
29:             {       cout << "\n1. Prikaz vrednosti jednog elementa niza";
30:                     cout << "\n2. Prikaz vrednosti svih elemenata niza\n\n";
31:
32:                     opcija = cin.get();
33:                     cin.ignore();
34:
35:                     switch(opcija)
36:                     {case '1':
37:                             prikaz_jednog(niz, MAX);
38:                             break;
39:                     case '2':
40:                             prikaz_svih(niz, MAX);
41:                             break;
42:                     default:
43:                             izlaz = 1;
44:                             break;
45:                     }
46:                     if(izlaz)
47:                             break;
48:             }
```

```
49:
50:             fbin.close();
51:             return 0;
52:     }
53:     //Funkcija koja upisuje u datoteku niz celih brojeva
54:     void upis(int niz[], int n)
55:     {       cout << "\nNiz je inicijalizovan na vrednosti:\n";
56:             for (int i=0; i<MAX; i++)
57:             {       niz[i] = 10*(i+1);
58:                     cout << niz[i] << " ";
59:             }
60:             cout << "\ni upisan u datoteku.\n";
61:
62:             fbin.write((char *)niz, MAX*sizeof(int));
63:     }
64:     //Funkcija koja prikazuje na ekranu vrednost elementa niza
65:     //sa rednim brojem zadatim preko tastature
66:     void prikaz_jednog(int niz[], int n)
67:     {       int rbroj, broj;
68:
69:             do
70:             {       cout << "Redni broj elementa: ";
71:                     cin >> rbroj;
72:                     cin.ignore();
73:             }while(rbroj<1 || rbroj>n);
74:
75:             //Pomeraj tekuce pozicije na odredjeni element,za citanje
76:             fbin.seekg((rbroj-1)*sizeof(int), ios::beg);
77:
78:             //Citanje vrednosti odredjenog elementa niza iz datoteke
79:             fbin.read((char *)&broj, sizeof(int));
80:             cout << "Vrednost elementa, procitana iz datoteke: ";
81:             cout << (int)broj << endl;
82:     }
83:     //Funkcija koja prikazuje na ekranu niz celih brojeva
84:     void prikaz_svih(int niz[], int n)
85:     {       //Pomeraj tekuce pozicije na pocetak datoteke, za citanje
86:             fbin.seekg(0);
87:
88:             //Citanje vrednosti svih elemenata niza iz datoteke
89:             fbin.read((char *)niz, n*sizeof(int));
90:
91:             cout << "\nCeo niz: ";
92:             for(int i=0; i<n; i++)
93:                     cout << niz[i] << " ";
94:             cout << endl;
95:     }
```

Анализа програма **primer9_4**

Овај програм уписује у бинарну датотеку низ целих бројева задат преко тастатуре, после тога чита исти низ из датотеке и приказује га на екрану.

На почетку програма глобално је декларисан објекат *fbin* класе *fstream* (линија 8.) који може бити повезан са датотеком за упис и читање. У линијама од 10. до 12. декларисане су функције по којима је распоређен кôд програма: *upis(), prikaz_jednog()* и *prikaz_svih().* Свака од ових функција добија у аргументима адресу и дужину низа и не враћа вредност.

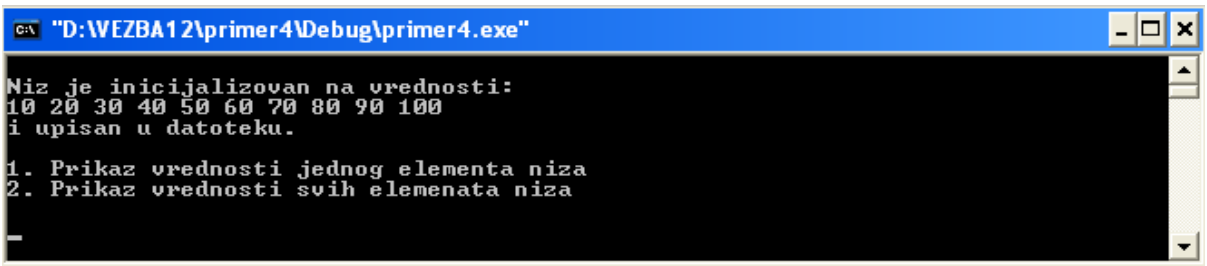
Функција *upis()* (дефиниција у линијама од 54. до 63.) иницијализује низ чију је адресу и дужину добила, приказује све елементе низа на екрану (слика 9.4 а)) и уписује их у датотеку позивом функције *write()* над објектом *fbin*.

Функција *prikaz_jednog()* (дефиниција у линијама од 66. до 82.) тражи са тастатуре редни број елемента (*rbroj*) (линије од 69. до 73.) и позива функцију *seekg()* над објектом *fbin* да помери текућу позицију од почетка датотеке *(ios::beg)* на почетак елемента типа *int* са задатим редним бројем *((rbroj-1)∗sizeof(int)* у датотеци (линија 76.). После позиционирања чита се вредност елемента низа из бинарне датотеке помоћу функције *read((char ∗)&broj, sizeof(int))* над објектом *fbin* (линија 79.). На месту првог аргумента ове функције написана је адреса променљиве резервисане за вредност целог броја прочитаног из датотеке (адреса је конвертована у адресу од типа *char*), а на месту другог аргумента је опсег броја који се чита. Прочитана вредност броја приказује се на екрану (слика 9.4 б)).

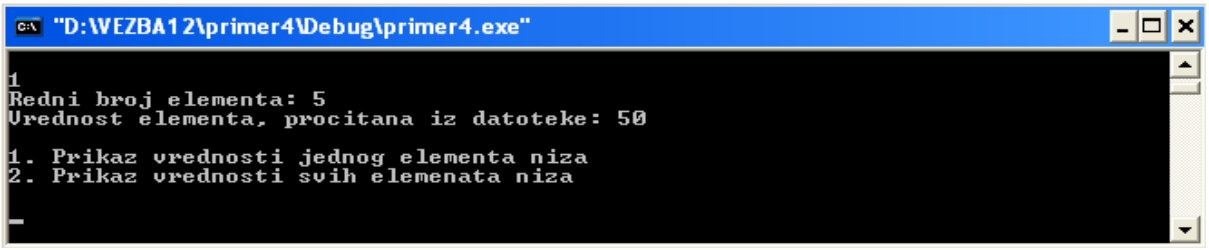
Функција *prikaz_svih()* (дефиниција у линијама од 84. до 95.) позива функцију *seekg(0)* над објектом *fbin* да помери текућу позицију на почетак датотеке (линија 86.). После позиционирања чита цео низ из бинарне датотеке помоћу функције *read((char ∗)niz, n∗sizeof(int))* над објектом *fbin* (линија 89.). На месту првог аргумента ове функције написана је адреса низа променљивих, резервисаног за *n* целих бројева прочитаних из датотеке (адреса је конвертована у адресу од типа *char*), а на месту другог аргумента је дужина низа целих бројева који се читају. Прочитани низ вредности приказује се на екрану (слика 9.4 в)).

У главној функцији програм декларише низ типа *int* и повезује објекат *fbin* са датотеком *"primer.bin"* (линија 18.). Ако ово успе датотека *"primer.bin"* отворена је у режиму *ios::out* | *ios::in* | *ios::binary* (бинарна, за упис и читање).

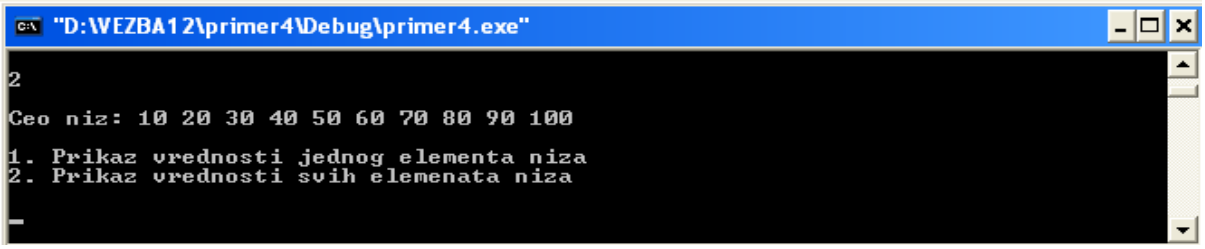
После позива функције *upis(),* за адресу и дужину стварног низа, главна функција у једној *while(1)* петљи (линије од 28. до 48.) пружа две опције: читање вредности једног елемента и читање вредности свих елемената низа и у зависности од изабране опције позива једну од описаних функција (*prikaz_jednog()* или *prikaz_svih()*). Када се изабере непозната опција (слика 9.4 г)), прекида се петља (линија 47.) и затвара отворена датотека (линија 50.).



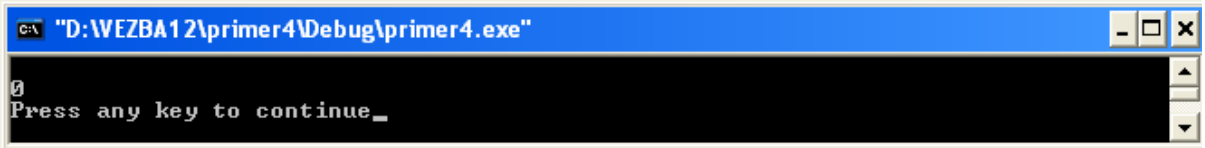
Слика 9.4 а) Излаз програма **primer9_4** (први део)



Слика 9.4 б) Излаз програма **primer9_4** (други део)



Слика 9.4 в) Излаз програма **primer9_4** (трећи део)



Слика 9.4 г) Излаз програма **primer9_4** (четврти део)

Изворни кôд програма **primer9_5**

```
1:      //primer9_5.cpp
2:      //Azuriranje sadrzaja binarne datoteke. Zapis je fiksne duzine:
3:      //ime studenta, smer, reg. broj i broj polozenih ispita.
4:      #include <iostream>
5:      #include <fstream>
6:      #include <stdlib.h>
7:      #include <string.h>
8:      #define MAX     15              //broj korisnih karaktera u stringu
9:      using namespace std;
10:     enum Greska{GR_OTV_UP, GR_OTV_CIT};
11:     char *poruke_o_greskama[] = {
12:                     "\nGreska pri otvaranju datoteke za upis!\n",
13:                     "\nGreska pri otvaranju datoteke za citanje!\n"};
14:
15:     fstream fbin;
16:
17:     struct student{
18:             char ime[MAX+1];
19:             char prezime[MAX+1];
20:             char smer[MAX+1];
21:             char reg_broj[MAX+1];
22:             int br_pol_ispita;
23:             }s1;
24:     int broj_struktura = 0;
```

```
25:
26:     void izmene(void);
27:     void prikaz(void);
28:     void greska(Greska status);
29:
30:     int main()
31:     {       char opcija; int izlaz = 0;
32:
33:             //Otvaranje binarne datoteke za upis
34:             fbin.open("studenti.bin", ios::out | ios::binary);
35:             if(!fbin)
36:                     greska(GR_OTV_UP);
37:
38:             //Prihvatanje vrednosti clanova strukture i upis u datoteku
39:             cout << "\nUnositi: ime, prezime, smer, reg. broj i ";
40:             cout << "broj pol. ispita (do reci Kraj)\n";
41:             while(1)
42:             {       cin >> s1.ime;
43:                     if(strcmp(s1.ime, "Kraj")==0)  break;
44:                     cin >> s1.prezime >> s1.smer;
45:                     cin >> s1.reg_broj >> s1.br_pol_ispita;
46:
47:                     //Upis strukture u datoteku
48:                     fbin.write((char *)&s1, sizeof(student));
49:                     broj_struktura++;
50:             }
51:             fbin.close();
52:             do
53:             {       cout << "\n1. Izmena\n2. Prikaz\n3. Izlaz";
54:                     cout << "\n\nIzabrati opciju: ";  cin >> opcija;
55:                     switch(opcija)
56:                     {       case '1':               izmene(); break;
57:                             case '2':               prikaz(); break;
58:                             default:                izlaz = 1; break;       }
59:             }while(!izlaz);
60:             return 0;
61:     }
62:     //Funkcija koja azurira sadrzaj datoteke
63:     void izmene(void)
64:     {       int rbroj, promena;
65:
66:             //Otvaranje binarne datoteke za citanje i upis
67:             fbin.open("studenti.bin", ios::in | ios::out | ios::binary);
68:             if(!fbin)
69:                     greska(GR_OTV_CIT);
70:
71:             //Izmena podataka u datoteci
72:             cout << "\nRedni broj studenta (do broja 0) i dodatni broj pol. Ispita\n";
73:             while(1)
74:             {       cin >> rbroj;
```

```
75:                     if(rbroj == 0)
76:                             break;
77:                     cin >> promena;
78:
79:                     //Pomeraj na strukturu i citanje njenog sadrzaja
80:                     fbin.seekg(-(rbroj-1)*sizeof(student), ios::beg);
81:                     fbin.read((char *)&s1, sizeof(student));
82:                     s1.br_pol_ispita += promena;
83:
84:                     //Povratak na pocetak strukture i upis njenog sadrzaja
85:                     fbin.seekp(-sizeof(student), ios::cur);
86:                     fbin.write((char *)&s1, sizeof(student));
87:             }
88:             fbin.close();
89:     }
90:     //Funkcija koja prikazuje na ekranu sadrzaj datoteke
91:     void prikaz(void)
92:     {       int i;
93:
94:             //Otvaranje binarne datoteke za citanje
95:             fbin.open("studenti.bin", ios::in | ios::binary);
96:             if(!fbin)
97:                     greska(GR_OTV_CIT);
98:
99:             //Prikaz na ekranu sadrzaja datoteke
100:            fbin.seekg(0);
101:            for(i=0; i<broj_struktura;i++)
102:            {       fbin.read((char *)&s1, sizeof(student));
103:
104:            cout << s1.ime << " " << s1.smer << " ";
105:            cout << s1.reg_broj << " " << s1.br_pol_ispita << endl;
106:            }
107:            fbin.close();
108:    }
109:    //Funkcija koja prikazuje na ekranu poruke o greskama
110:    void greska(Greska status)
111:    {       cerr << poruke_o_greskama[status];
112:            exit(1);
113:    }
```

Анализа програма **primer9_5**

Програм у овом примеру ажурира садржај бинарне датотеке (уписује податке прочитане са тастатуре, мења их по потреби и приказује на екрану).

На почетку програма глобално је декларисан објекат *fbin* класе *fstream* (линија 15.), који може бити повезан са датотеком за упис и за читање.

У линијама од 17. до 23. декларисан је тип структуре *student* (*ime, prezime, smer, reg_broj, broj_pol_ispita*) и примерак *s1* овог типа структуре, а у линији 24. декларисана је и иницијализована на вредност 0 променљива *broj_struktura.*

У линијама од 26. до 28. декларисане су функције програма: *izmene(), prikaz()* и *greska().* Прве две функције не користе аргументе и не враћају вредности. Функција *greska()* је познатог облика.

Функција *izmene()* (дефиниција у линијама од 63. до 89.) повезује објекат *fbin* са датотеком *"studenti.bin".* Ако ово успе, датотека *"studenti.bin"* отворена је у режиму *ios::in* | *ios::out* | *ios::binary* (бинарна, за читање и упис) (линије од 67. до 69.). Функција даље тражи да се уносе преко тастатуре редни бројеви студената, код којих постоје измене (то су редни бројеви записа у датотеци) и измене за бројеве положених испита (линије од 74. до 77.) и за сваки учитани редни број (*rbroj*) и број положених испита (*promena*):

- помери текућу позицију у датотеци на почетак записа са подацима о студенту са задатим редним бројем (линија 80.),
- прочита садржај записа (све податке о студенту са задатим редним бројем) (линија 81.),
- измени тражени податак о студенту (*br_pol_ispita*) (линија 82.),
- помери текућу позицију за један запис уназад (опет на почетак записа са подацима о истом студенту) (линија 85.),
- упише на место одговарајућег записа у датотеци нове податке о студенту (линија 86.),

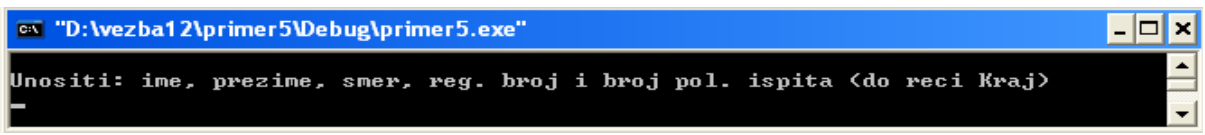
На крају, функција затвара отворену датотеку (линија 88.).

Функција *prikaz()* (линије од 91. до 108.) повезује објекат *fbin* са датотеком *"studenti.bin".* Ако ово успе датотека *"studenti.bin"* отворена је у режиму *ios::in* | *ios::binary* (бинарна, за читање) (линије од 95. до 97.). Функција помера текућу позицију на почетак датотеке (линија 100.), чита у једној *for* петљи садржај сваког записа (линија 102.), приказује на екрану ове податке (линије 104. и 105.) и затвара отворену датотеку (линија 107.).

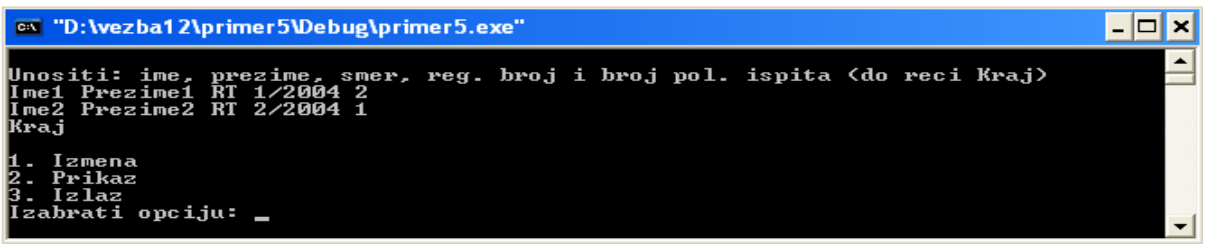
Програм у главној функцији повезује објекат *fbin* са датотеком *"studenti.bin".* Ако ово успе датотека *"studenti.bin"* отворена је у режиму *ios::out* | *ios::binary* (бинарна, за упис) (линије од 34. до 36.). За сваког студента учитавају се потребни подаци, смештају у одговарајуће чланове структуре (линије од 39. до 45.), уписују се у отворену датотеку (линија 48.), инкрементира се садржај бројача структура (линија 49.) и затвара отворена датотека (линија 51.).

Главна функција садржи још и приказ на екрану опција: *1.Izmena, 2. Prikaz* и *3. Izlaz,* све док се не изабере опција 3. У зависности од изабране опције (1. или 2.) позива одговарајућу функцију (*izmene()* или *prikaz()*).

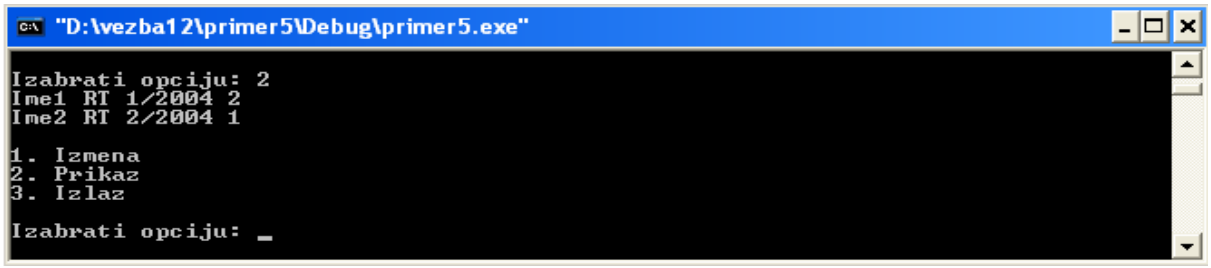
На сликама од 9.5 а) до 9.5 е) може се видети један могући сценарио извршавања програма.



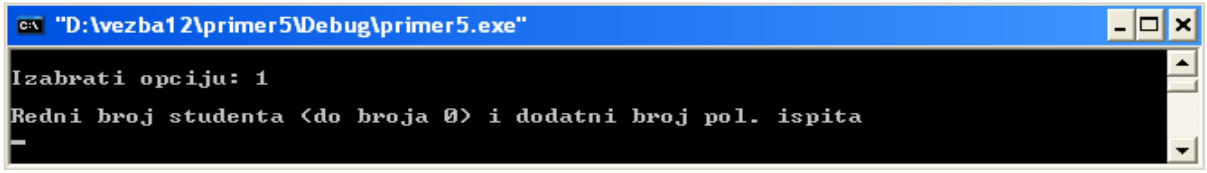
Слика 9.5 а) Излаз програма **primer9_5** (први део)



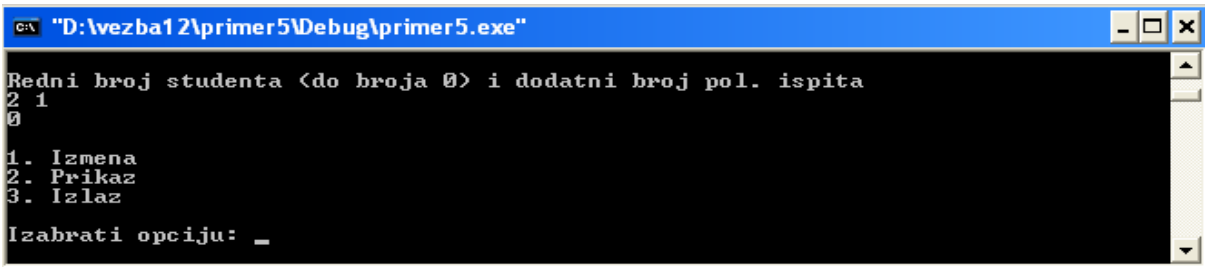
Слика 9.5 б) Излаз програма **primer9_5** (други део)



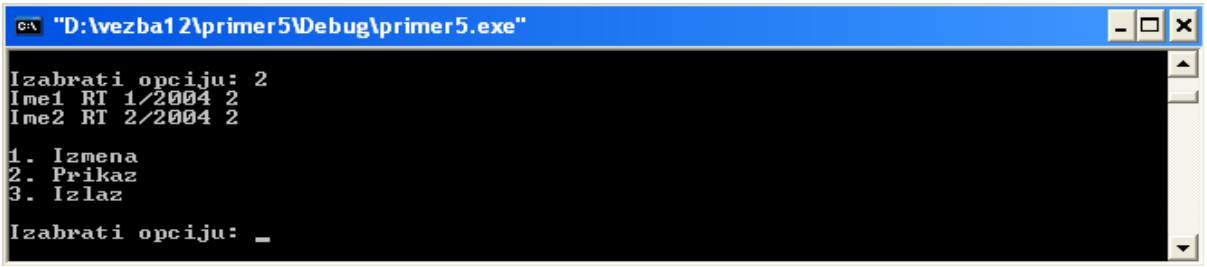
Слика 9.5 в) Излаз програма **primer9_5** (трећи део)



Слика 9.5 г) Излаз програма **primer9_5** (четврти део)



Слика 9.5 д) Излаз програма **primer9_5** (пети део)



Слика 9.5 ђ) Излаз програма **primer9_5** (шести део)

Слика 9.5 е) Излаз програма **primer9_5** (седми део)

Изворни кôд програма **primer9_6***

```
1:      //primer9_6.cpp
2:      //Pretrazivanje spiska stanara
3:      //(upotreba dinamicke dodele memorije i povezanih lista)
4:      #include <iostream>
5:      #include <stdlib.h>
6:      #include <ctype.h>
7:      #include <string.h>
8:      #define MAX     30              //broj korisnih karaktera u stringu
9:      using namespace std;
```

```
10:
11:     struct adresa{
12:             char ime[MAX+1];
13:             char ulica[MAX+1];
14:             int broj;
15:             char grad[MAX+1];
16:             adresa *sledeca;
17:             };
18:
19:     void cita(adresa *ptr_a);
20:     void nalazi(adresa *ptr_a);
21:     void stampa(adresa *ptr_a);
22:     void greska(void);
23:
24:     int main()
25:     {       char odgovor = 'd';
26:             adresa *prva, *tekuca;
27:
28:             prva = new adresa;
29:             tekuca = prva;
30:
31:             do
32:             {       if( tekuca == NULL )
33:                             greska();
34:
35:                     tekuca->sledeca = NULL;
36:
37:                     cita(tekuca);
38:
39:                     cout << "\n\nUnesite d za nastavak: ";
40:                     cin  >> odgovor;
41:                     cin.ignore();           //cita znak za prelaz u novi red
42:
43:                     if( tolower(odgovor) == 'd')
44:                     {       tekuca->sledeca = new adresa;
45:                             tekuca = tekuca->sledeca;
46:                     }
47:             }while(tolower(odgovor) == 'd');
48:
49:             nalazi(prva);
50:
51:             return 0;
52:     }
53:     //Funkcija koja cita sa tastature podatke o adresi
54:     void cita(adresa *ptr_a)
55:     {       cout << "\nUnesite ime i prezime stanara: ";
56:             cin.getline(ptr_a -> ime, MAX+1);
57:
58:             cout << "Unesite ime ulice: ";
59:             cin.getline(ptr_a -> ulica, MAX+1);
```

```
60:
61:             cout << "Unesite broj ulaza: ";
62:             cin >> ptr_a -> broj;
63:             cin.ignore();
64:
65:             cout << "Unesite ime grada: ";
66:             cin.getline(ptr_a -> grad, MAX+1);
67:     }
68:     //Funkcija koja nalazi zadatu adresu u postojecem spisku
69:     void nalazi(adresa *ptr_a)
70:     {       char grad[MAX+1], ulica[MAX+1];
71:             int broj_ulaza;
72:
73:             cout << "\nTrazeni grad? ";
74:             cin.getline(grad, MAX+1);
75:
76:             cout << "Trazena ulica: ";
77:             cin.getline(ulica, MAX+1);
78:
79:             cout << "Trazeni broj ulaza? ";
80:             cin >> broj_ulaza;
81:             cin.ignore();
82:
83:             cout << "\nSpisak stanara:\n";
84:
85:             while(ptr_a != NULL )
86:             {       if(     strcmp(grad, ptr_a -> grad) == 0 &&
87:                             strcmp(ulica, ptr_a -> ulica) == 0 &&
88:                             broj_ulaza == ptr_a -> broj                     )
89:                                     stampa(ptr_a);
90:                     ptr_a = ptr_a -> sledeca;
91:             }
92:     }
93:     //Funkcija koja prikazuje na ekranu ime od adrese
94:     void stampa(adresa *ptr_a)
95:     {       cout << ptr_a -> ime << endl;
96:     }
97:     //Funkcija koja prikazuje na ekranu poruke o greskama
98:     void greska(void)
99:     {       cerr << "Nije uspela dinamicka dodela memorije\n";
100:            exit(1);
101:    }
```

Анализа програма **primer9_6***

Овај програм претражује садржај бинарне датотеке (претходно задат са тастатуре) и резултате претраживања приказује на екрану.

Програм на почетку декларише тип структуре *adresa* (*ime, ulica, broj, grad* и *sledeca*), као и функције: *cita() , nalazi(), stampa()* и *greska().* Свака од ових функција, изузев функције *greska(),* користи показивач на структуру типа *adresa.*

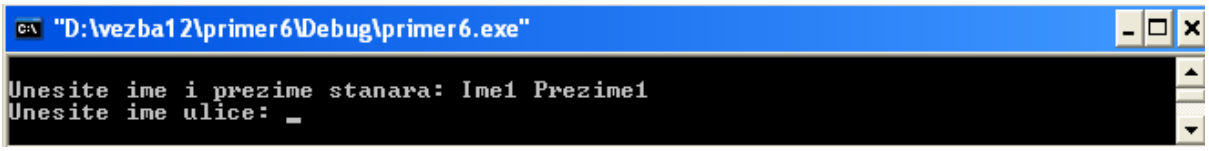
Функција *cita()* (линије од 54. до 67.) чита са тастатуре податке о адреси и смешта их у чланове структуре чију је адресу добила у аргументу.

Функција *nalazi()* (линије од 69. до 92.) прво прихвата са тастатуре податке по којима ће извести претраживање (*grad, ulica* и *broj_ulaza*) у листи структура (адресу прве структуре у листи добија у аргументу). У једној *while* петљи (за сваку структуру у листи) пореди садржаје задатих података са одговарајућим члановима структуре и за нађене податке у структури позива функцију *stampa().*

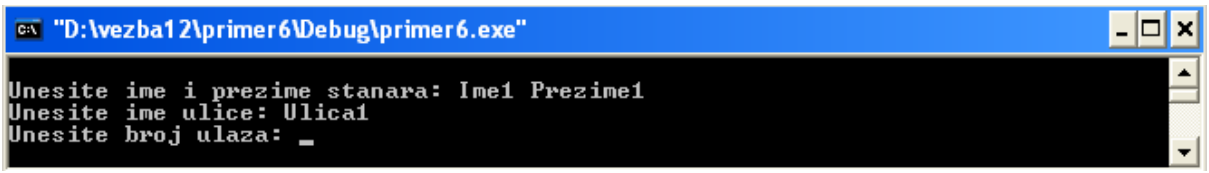
Функција *stampa()* (линије од 94. до 96.) приказује на екрану вредност једног члана (*ime)* структуре (чију адресу добија у аргументу).

Главна функција (линије од 24. до 52.) у једној *do..while* петљи (линије од 31. до 47.) позива функцију *cita()*, да прихвати податке са тастатуре за сваку структуру у листи, све док се потврђује наставак петље. После прекида петље следи позив функције *nalazi()* која претражује листу структура и приказује на екрану резултат претраге. На сликама 9.6 а) до 9.6 ж) може се видети један могући сценарио извршавања програма.

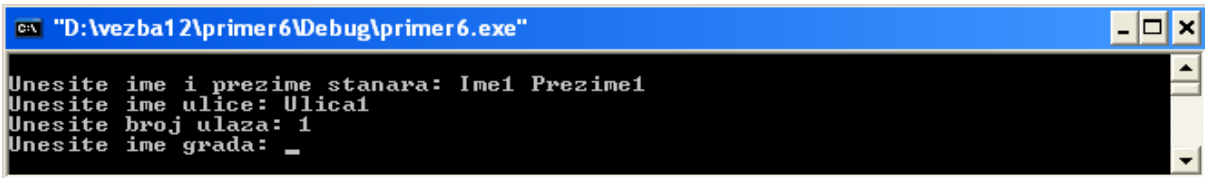
Слика 9.6 а) Излаз програма **primer9_6** (први део)



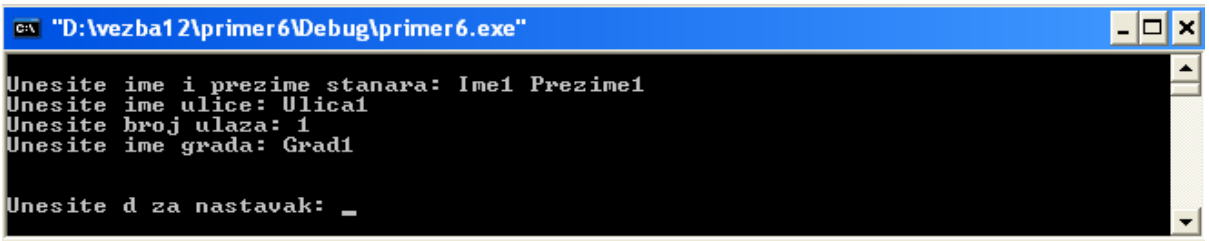
Слика 9.6 б) Излаз програма **primer9_6** (други део)



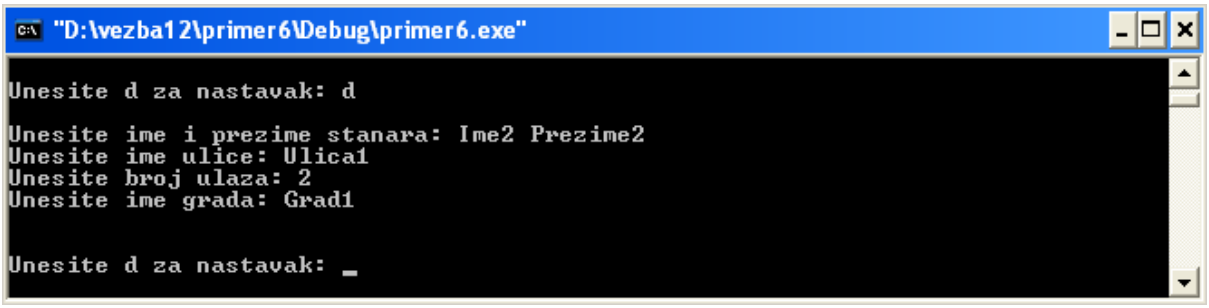
Слика 9.6 в) Излаз програма **primer9_6** (трећи део)



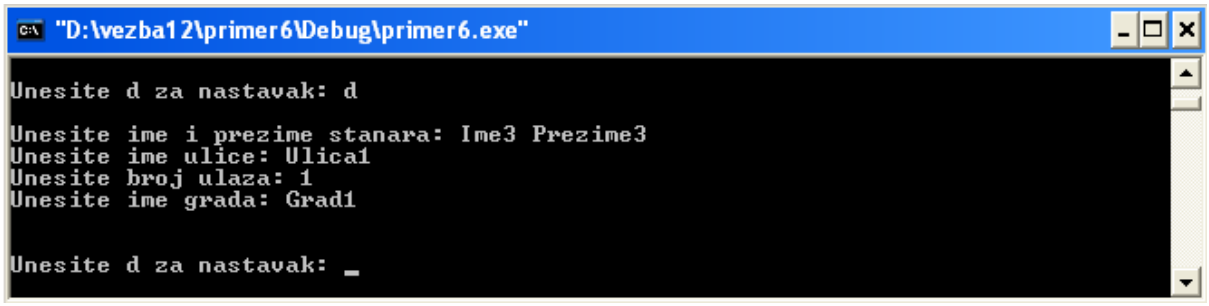
Слика 9.6 г) Излаз програма **primer9_6** (четврти део)



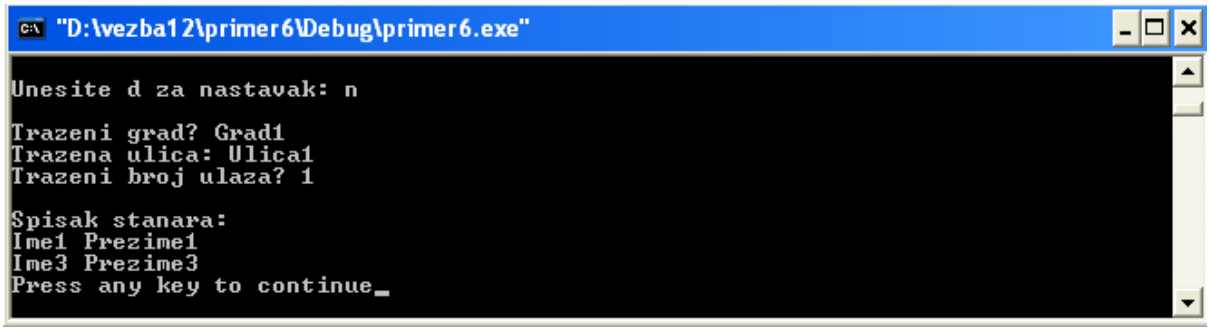
Слика 9.6 д) Излаз програма **primer9_6** (пети део)



Слика 9.6 ђ) Излаз програма **primer9_6** (шести део)



Слика 9.6 е) Излаз програма **primer9_6** (седми део)



Слика 9.6 ж) Излаз програма **primer9_6** (осми део)

Практични савети

- ✓ За рад са датотекама потребно је укључити стандардну библиотеку *fstream* и у том случају на располагању су класе *fstream, ofstream* и *ifstream.* За упис и читање садржаја датотеке може се користити класа *fstream*. Ако се само уписује садржај у датотеку, користи се класа *ofstream*, а ако се само чита садржај, класа *ifstream*.

Евентуалне грешке

- ➢ Не треба изоставити проверу успостављања тражене везе између објекта класе *fstream* и одговарајуће датотеке.
- ➢ Не треба изоставити раскидање сваке успостављене везе између објекта класе *fstream* и датотеке да последње измене у датотеци не би биле евентуално изгубљене.

Задаци за припрему вежбе 9. у лабораторији

Задатак 1.

Написати програм на језику *C++* који копира текст из једне датотеке у другу на следећи начин: прво слово од сваке речи конвертује у исто велико. Предвидети да имена датотека програм прихвата са тастатуре.

Задаци за вежбу 9. у лабораторији

Предвидети у следећим програмима да прихватају са тастатуре имена свих потребних датотека.

Задатак 1.

Написати програм на језику *C++* који чита садржај текстуалне датотеке и приказује на екрану извештај о дужини најдужег реда у овој датотеци.

Задатак 2.

Написати програм на језику *C++* који копира садржај једне текст датотеке у другу на следећи начин: избацује све празне редове.

Задатак 3.

Написати програм на језику *C++* који из датотеке са изворним кôдом овог програма копира текст у другу датотеку на следећи начин: на почетак сваког реда додаје његов редни број и ред помера за хоризонтални табулатор.

Задатак 4.

Написати програм на језику *C++* који:

а) од *m (m≤100)* динамичких низова целих бројева, сваки дужине *n (n≤100),* задатих преко тастатуре, формира *m* записа у бинарној датотеци,
б) разврстава записе ове датотеке у две бинарне датотеке, тако да у једној буду низови који не садрже број 0, а у другој сви остали низови,
в) приказује на екрану садржаје креираних датотека.

Задатак 5.

Написати програм на језику *C++* који:

а) од података задатих преко тастатуре (*ime, broj_indeksa, ocena*) за *n (n≤100)* студената, формира *n* записа у бинарној датотеци,
б) формира другу бинарну датотеку од записа из прве датотеке за оне студенте који су положили испит,
в) за задате податке са тастатуре (*ime* и *broj_indeksa)* чита из друге датотеке и приказује на екрану податак *ocena*.

Вежба 10.

Класе и објекти у програмима на језику *C++*

Креирање класа и њихових објеката

Конструктори и деструктори класа

Функције чланице класа

Наслеђивање код класа

Примери

Изворни кôд програма **primer10_1**

```
1:      //primer10_1.cpp
2:      //Izracunavanje broja sati, minuta i sekundi u zadatom broju sekundi
3:      //(Koriscenje konstruktora i destruktora klase)
4:      #include <iostream>
5:      using namespace std;
6:
7:      class konv {
8:              int ukupno_sek;
9:          public:
10:             konv(void)
11:             {       ukupno_sek=0;
12:                     cout << "\nPocetak konvertovanja.\n\n";  }
13:             ~konv(void)
14:             {       cout << "\nKraj konvertovanja.\n\n";       }
15:
16:             void prikaz(int);
17:             void u_sate(void);
18:             void u_min(void);
19:             void u_sek_ost(void);
20:     };
21:     int main( )
22:     {       konv konv_obj;
23:
24:             int ukupno;
25:
26:             cout << "Ukupan broj sekundi? "; cin >> ukupno;
27:
28:             konv_obj.prikaz(ukupno);
29:             konv_obj.u_sate();
30:             konv_obj.u_min();
31:             konv_obj.u_sek_ost();
32:
33:             return 0;
34:     }
35:     void konv::prikaz(int sek)
36:     {       if(sek>0)
37:                     ukupno_sek = sek;
38:             cout << ukupno_sek << " sekundi je:" << endl;       }
39:     void konv::u_sate(void)
40:     {       cout<<ukupno_sek/3600 << " sati, ";                 }
41:     void konv::u_min(void)
42:     {       cout << (ukupno_sek / 60) % 60 << " minuta, ";      }
43:     void konv::u_sek_ost(void)
44:     {       cout << ukupno_sek% 60  << " sekundi." << endl;   }
```

Анализа програма **primer10_1**

Програм у овом примеру конвертује задат укупан број секунди у одговарајући број сати, као и у бројеве преосталих минута и секунди и приказује на екрану резултат ове конверзије.

На почетку програма дефинисана је класа *konv* (линије од 7. до 20.), која се уводи за конверзију из једног у други облик задатог времена.

Следи опис чланова класе *konv*. Податак *ukupno_sek* (линија 8.) приватни је члан податак (ако се испред члана не напише другачије, приватан је за класу), што значи да је ограничен само на своју класу (није доступан из функција које нису чланице исте класе). Све функције класе дефинисане су као њене јавне чланице (линије од 9. до 19.) и због тога су доступне и ван своје класе, у овом примеру из главне функције програма.

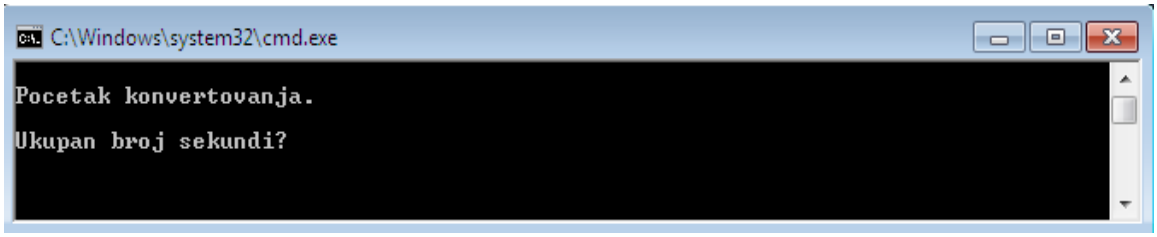
Функција *konv()* конструктор је класе (линије од 10. до 12.), има исти назив као и класа и позива се аутоматски након креирања сваког објекта класе. У овом случају, предвиђено је креирање једног објекта и функција конструктор користи се за иницијализацију podatka *ukupno_sek* (линика 11.) и штампање поруке за почетак рада програма (линија 12.).

Функција *~konv()* деструктор је класе (линије од 13. до 14.), има исти назив као и класа, само са префиксом ~ и позива се аутоматски након укидања сваког објекта класе. У овом случају, функција деструктор користи се за штампање поруке за крај рада програма (линија 14.).

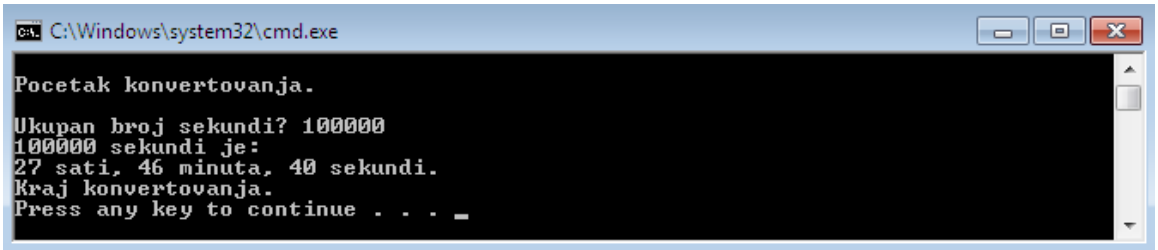
Функције *prikaz(), u_sate(), u_min()* и *u_sek_ost()* унутар дефиниције класе имају само декларације (линије од 16. до 19.).

Након креирања објекта *konv_obj* на почетку главне функције (линија 22.) и декларације и читања са тастатуре укупног броја секунди *ukupno* (линије од 24. до 26.), програм из главне функције позива редом јавне функције класе *konv: prikaz(), u_sate(), u_min()* и *u_sek_ost()* над објектом *konv_obj* (линије од 28. до 31.). Свака од позваних функција извршава један корак задате конверзије. У заглављу дефиниције сваке ове функције (дефиниције су у линијама од 35. до 44.), може се видети да је, с обзиром на то да је дефиниција написана ван дефиниције класе, неопходно помоћу оператора досега (::) назначити припадање класи *(void konv::prikaz(), void konv::u_sate(), void konv::u_min(),void konv::u_sek_ost())*. Из сваке од ових функција приступа се податку *ukupno_sek* (ово је могуће јер су и податак и функције из којих се приступа овом податку, чланови исте класе).

На сликама од 10.1 а) до 10.1 б) може се видети један могући сценарио извршавања овог програма.



Слика 10.1 а) Излаз програма **primer10_1** (први део)



Слика 10.1 б) Излаз програма **primer10_1** (други део)

Изворни кôд програма **primer10_2**

```
1:      //primer10_2.cpp
2:      //Izracunavanje apsolutnih vrednosti brojeva u raznim oblicima
3:      //(Preklapanje funkcija, clanica klase)
4:      #include <iostream>
5:      #include <stdlib.h>
6:      #include <math.h>
7:      using namespace std;
8:
9:      class apsolutna {
10:         public:
11:             int f_aps(int);
12:             double f_aps(double);
13:             double f_aps (char *);
14:     };
15:     int main( )
16:     {       apsolutna aps_obj;
17:
18:             cout << "\nApsolutna vrednost broja -547 je: "
19:                  << aps_obj.f_aps(-547) << endl;
20:
21:             cout << "\nApsolutna vrednost broja -547.256 je: "
22:                  << aps_obj.f_aps(-547.256) << endl;
23:
24:             cout << "\nApsolutna vrednost broja \"-547.82abc\" je: "
25:                  << aps_obj.f_aps("-547.82abc" ) << endl;
26:
27:             return 0;
28:     }
29:     int apsolutna:: f_aps(int broj)
30:     {       int aps_i;
31:             aps_i = abs(broj);
32:             return aps_i;
33:     }
34:     double apsolutna:: f_aps(double broj)
35:     {       double aps_d;
36:             aps_d = fabs(broj);
37:             return aps_d;
38:     }
```

```
39:     double apsolutna:: f_aps(char *s)
40:     {       double broj, aps_d;
41:             broj = atof(s);
42:             aps_d = fabs(broj);
43:             return aps_d;
44:     }
```

Анализа програма **primer10_2**

Програм у овом примеру израчунава апсолутну вредност података задатих у разним облицима и приказује на екрану резултат.

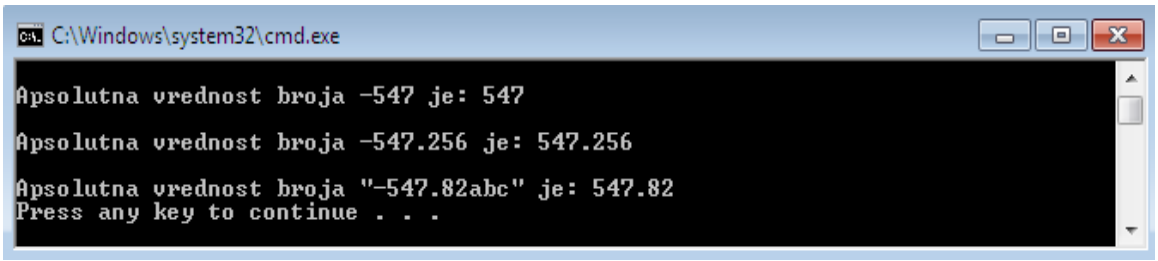
На почетку програма дефинисана је класа *apsolutna* (линије од 9. до 14.), која се уводи за израчунавање апсолутних вредности података, задатих преко тастатуре у разним облицима: целог броја, реалног броја и реалног броја препознатог на почетку знаковног низа.

Следи опис дефинисаних чланова класе *apsolutna*. Све функције чланице ове класе дефинисане су као јавне (линије од 10. до 13.), с међусобно истим именом *f_aps(),* али с разликом у листи аргумената, евентуално и у типу повратне вредности, што значи да су међусобно преклопљене функције (свака од њих ради са одређеним типом података и враћа резултат одговарајућег типа). Прва функција користи број типа int и враћа резултат истог типа, друга користи број типа double и враћа резултат истог типа, док трећа користи адресу стринга и враћа резултат типа double.

Након креирања објекта *aps_obj* на почетку главне функције (линија 16.), програм из главне функције позива редом јавне функције класе *apsolutna* (три преклопљене функције под називом *f_aps()*) над објектом *aps_obj* (линије од 18. до 25.). Свака од позваних функција, након тога, извршава своје наредбе

Прва од преклопљених функција *f_aps()* израчунава апсолутну вредност задатог целог броја, друга задатог реалног броја, а трећа реалног броја пронађеног у задатом стрингу (линије од 29. до 44.).

На слици 10.2 може се видети резултат извршавања овог програма.



Слика 10.2 Излаз програма **primer10_2**

Изворни кôд програма **primer10_3**

```
1:      //primer10_3.cpp
2:      //Provera lozinke studenta i prikaz poena ostvarenih na testu
3:      //(Poziv funkcija, privatnih i javnih clanica klase)
4:      #include <iostream>
5:      #include <string.h>
6:      #include <stdlib.h>
```

```
7:      #define MAX 30
8:      using namespace std;
9:
10:      class test {
11:                        char ime[MAX+1];
12:                        int lozinka,
13:                            br_tacnih,
14:                            br_netacnih;
15:                        double br_poena;
16:
17:                        int Br_tacnih(void);
18:                        int Br_netacnih(void);
19:                        double Br_poena(void);
20:
21:           public:
22:                        void Prompt(int neka_lozinka);
23:                        void Pocetak(char *poc_ime,
24:                                          int poc_lozinka,
25:                                          int poc_tacnih,
26:                                          int poc_netacnih,
27:                                          double poc_poena);
28:                        void Pozdrav(void);
29:                       void Tacnih(int broj);
30:                       void Netacnih(int broj);
31:      } student;
32:
33:      int main( )
34:      {        int uneta_lozinka;
35:
36:               student.Pocetak("Student Student", 1234, 0, 0, 0);
37:               cout<<"Unesite lozinku: ";  cin>>uneta_lozinka;
38:
39:               student.Pozdrav();  student.Prompt(uneta_lozinka);
40:               student.Tacnih(7);  student.Prompt(uneta_lozinka);
41:               student.Netacnih(3);  student.Prompt(uneta_lozinka);
42:
43:               return 0;
44:      }
45:      int test::Br_tacnih(void)
46:      {        return(br_tacnih);
47:      }
48:      int test::Br_netacnih(void)
49:      {        return(br_netacnih);
50:      }
51:      double test::Br_poena(void)
52:      {        return(br_poena);
53:      }
54:      void test::Prompt(int neka_lozinka)
55:      {        char izbor;
56:               if( neka_lozinka == lozinka )
```

```
57:             {       cout<<"\nBirati opciju P za broj poena ";
58:                     cout<<"ili bilo koju drugu opciju za kraj: ";
59:                cin>>izbor;
60:                     if(izbor == 'P' || izbor == 'p')
61:                     {       cout << "\nTacnih odgovora: ";
62:                             cout << student.Br_tacnih();
63:                             cout << "\nNetacnih odgovora: ";
64:                             cout << student.Br_netacnih();
65:                             cout << "\nTrenutno ostvarenih poena: ";
66:                             cout << student.Br_poena()<<endl;
67:                     }
68:                     else
69:                     {       cout<<"\nIzabrana opcija za kraj!\n\n"; exit(1);
70:                     }
71:             }
72:             else
73:             {       cout<<"\nUneta nekorektna lozinka!\n\n"; exit(1);
74:             }
75:     }
76:     void test::Pocetak( char *poc_ime,
77:                                             int poc_lozinka,
78:                                             int poc_tacnih,
79:                                             int poc_netacnih,
80:                                             double poc_poena)
81:     {       strcpy(ime,poc_ime);
82:             lozinka = poc_lozinka;
83:             br_tacnih = poc_tacnih;
84:             br_netacnih = poc_netacnih;
85:             br_poena = poc_poena;
86:     }
87:     void test::Pozdrav(void)
88:     {       cout<<"\nDobro dosli na test "<<ime<<endl;
89:             cout<<"\nZa tacan odgovor ostvarujete 1 poen";
90:             cout<<"\nZa netacan odgovor -0.5 poena"<<endl;
91:     }
92:     void test::Tacnih(int broj)
93:     {       br_tacnih += broj;
94:             br_poena += broj;
95:     }
96:     void test::Netacnih(int broj)
97:     {       br_netacnih += broj;
98:             br_poena -= broj*0.5;
99:     }
```

Анализа програма **primer10_3**

Програм у овом примеру проверава унапред дефинисану лозинку студента који приступа тесту, ажурира и приказује на екрану резултате теста.

На почетку програма дефинисана је класа *test* (линије од 10. до 31.), уведена за проверу лозинке, ажурирање и приказ резултата теста. Унутар

дефиниције класе, након дефинисаних свих њених чланова креиран је и објекат ове класе под називом *student* (линија 31.).

Следи опис дефинисаних чланова класе *test*. Подаци *ime, lozinka br_tacnih, br_netacnih* и *br_poena* (линије од 11. до 15.) приватни су чланови класе *test*. Функције *Br_tacnih(), Br_Netacnih()* и *Br_poena()* такође су приватне чланице (линије од 17. до 19.), што значи да им је свима могућ приступ само из функција чланица исте класе. Функције *Prompt(), Pocetak(), Pozdrav(), Tacnih()* и *Netacnih()* дефинисане су као јавне чланице исте класе (линије од 21. до 30.) и због тога су доступне и ван своје класе, у овом примеру из главне функције.

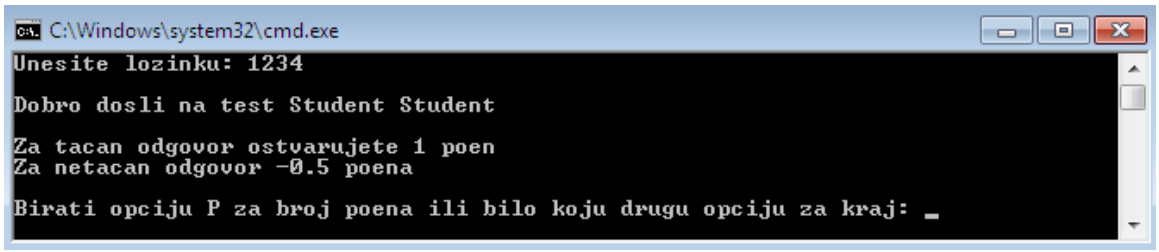
На почетку главне функције програм декларише податак *uneta_lozinka* (линија 34.). Следи позив јавне функције класе *Pocetak()* над објектом *student* (линија 36.), која иницијализује име, лозинку, број тачних одговора, број нетачних одговора и број поена. Након тога, програм из главне функције тражи да се унесе вредност податка *uneta_lozinka* и позива јавне функције класе *test: Pozdrav(), Tacnih()* и *Netacnih()* над објектом *student*, да поздрави корисника, и ажурира број поена на основу унапред дефинисаних бројева тачних и нетачних одговора на тесту. Након позива сваке од поменутих функција, програм из главне функције позива функцију *Prompt() (*такође јавну функцију класе *test)* над објектом *student,* да би у случају исправне лозинке и одговарајуће опције приказао на екрану у сваком кораку теста број тачних и нетачних одговора, као и остварених поена (линије од 36. до 41.).

Дефиниције свих приватних функција, *Br_tacnih(), Br_Netacnih()* и *Br_poena()*, као и јавних, *Prompt()*, *Pocetak()*, *Pozdrav(), Tacnih()* и *Netacnih()* написане су након дефиниције главне функције (линије од 45. до 99.).

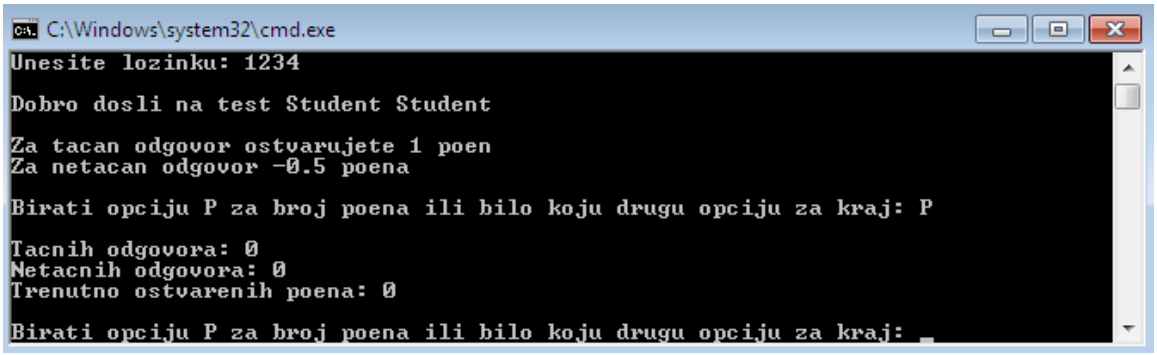
Из наредби неких јавних функција чланица класе *test*, приступа се приватним подацима и приватним функцијама исте класе. Функција *Prompt()* тако позива приватне функције *Br_tacnih(), Br_Netacnih()* и *Br_poena()* и користи њихове резултате (линије од 61. до 66.), док функција *Pocetak()* приступа приватним подацима *ime, lozinka br_tacnih, br_netacnih* и *br_poena* и додељује им иницијалне вредности (линије од 81. до 85.).

На сликама од 10.3 а) до 10.3 г) може се видети један могући сценарио извршавања овог програма.

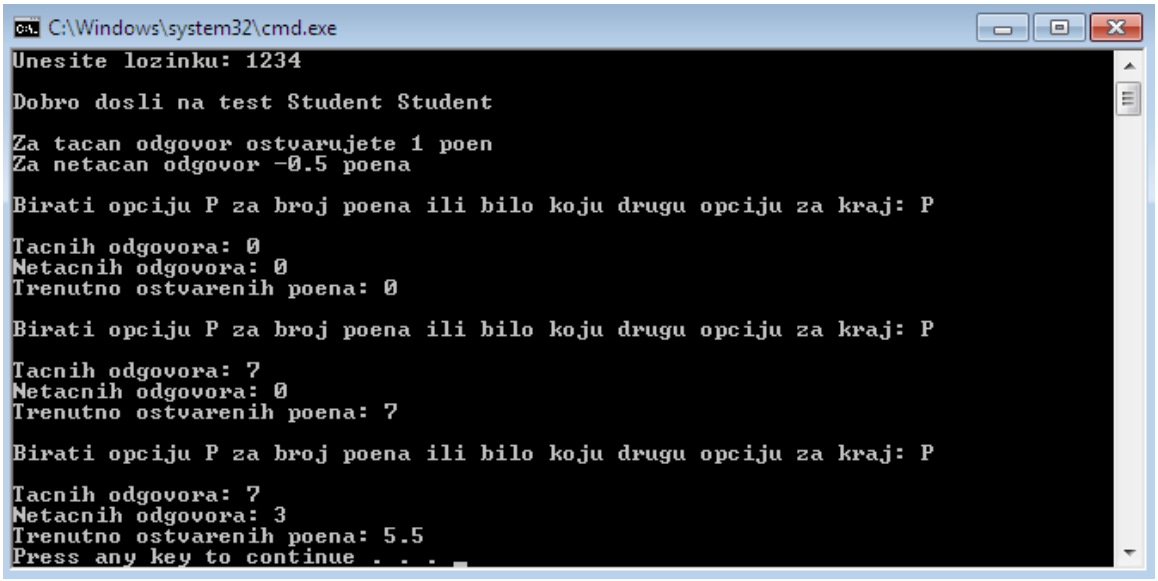
Слика 10.3 а) Излаз програма **primer10_3** (први део)



Слика 10.3 б) Излаз програма **primer10_3** (други део)



Слика 10.3 в) Излаз програма **primer10_3** (трећи део)



Слика 10.3 г) Излаз програма **primer10_3** (четврти део)

Изворни кôд програма **primer10_4**

```
1:      //primer10_4.cpp
2:      //Azuriranje podataka o studentu i ocenama na ispitima
3:      //(Nasledjivanje kod klasa)
4:      #include <iostream>
5:      #include <string.h>
6:      #define MAX 30
7:      using namespace std;
8:
9:      class student {
10:                     char ime[MAX+1],
11:                     st_program[MAX+1],
12:                     broj_ind[MAX+1];
13:             public:
14:                     void citanje( void );
15:                     void prikaz( void );
16:     };
```

```
17:    class obavezno:public student{
18:                    char predmet_ob[MAX+1];
19:                    int ocena_ob;
20:            public:
21:                    void citanje_ob( void );
22:                    void prikaz_ob( void );
23:     };
24:     class izborno:public student{
25:                    char predmet_izb[MAX+1];
26:                    int ocena_izb;
27:            public:
28:                    void citanje_izb( void );
29:                    void prikaz_izb( void );
30:     };
31:     int main( )
32:     {       obavezno obav_obj;
33:             izborno izb_obj;
34:
35:             cout << "\nStudent koji je polozio obavezan predmet: \n";
36:             obav_obj.citanje_ob();
37:
38:             cout << "\nStudent koji je polozio izborni predmet: \n";
39:             izb_obj.citanje_izb();
40:
41:             cout << "\nPolozen obavezan predmet: \n";
42:             obav_obj.prikaz_ob();
43:
44:             cout << "\nPolozen izborni predmet: \n";
45:             izb_obj.prikaz_izb();
46:
47:             return 0;
48:             }
49:     void student::citanje(void)
50:     {       cout << "Ime studenta: " ;
51:            cin.get(ime, MAX+1, '\n') ;
52:            cin.ignore() ;
53:
54:             cout << "Studijsksi program: " ;
55:             cin.get(st_program, MAX+1, '\n') ;
56:             cin.ignore() ;
57:
58:             cout << "Broj indeksa: " ;
59:             cin.get(broj_ind, MAX+1, '\n') ;
60:             cin.ignore() ;
61:     }
62:      void student::prikaz(void)
63:      {         cout << "Ime: " << ime << endl ;
64:               cout << "Studijski program: " << st_program << endl ;
65:               cout << "Broj indeksa: " << broj_ind << endl ;
66:      }
```

```
67:     void obavezno::citanje_ob(void)
68:     {       citanje();
69:
70:             cout << "Naziv predmeta: ";
71:             cin.get(predmet_ob, MAX+1, '\n');
72:             cin.ignore();
73:
74:             cout << "Ocena: ";
75:             cin >> ocena_ob;
76:             cin.ignore();
77:     }
78:     void obavezno::prikaz_ob(void)
79:     {       prikaz();
80:
81:             cout << "Naziv predmeta: " << predmet_ob << endl ;
82:             cout << "Ocena: " << ocena_ob << endl ;
83:     }
84:     void izborno::citanje_izb(void)
85:     {       citanje();
86:
87:             cout <<"Naziv predmeta: ";
88:             cin.get(predmet_izb, MAX+1, '\n');
89:             cin.ignore();
90:
91:             cout << "Ocena: ";
92:             cin >> ocena_izb;
93:             cin.ignore();
94:     }
95:     void izborno::prikaz_izb(void)
96:     {       prikaz();
97:
98:             cout <<"Naziv predmeta: " << predmet_izb << endl ;
99:             cout <<"Ocena: " << ocena_izb << endl ;
100:      }
```

Анализа програма **primer10_4**

Програм у овом примеру ажурира податке о студенту и оценама на једном испиту из обавезног / изборног предмета.

На почетку програма дефинисана је класа *student* (линије од 9. до 16.), уведена као основна класа (класа родитељ) за ажурирање података о студенту и његовим оценама на испитима. Поред ове класе, програм садржи и дефиниције две класе изведене из класе *student* (свака изведена класа садржи све особине основне класе и неке само своје особине, које се накандно дефинишу). Изведене класе у овом примеру су: класа *obavezno*, само за обавезне предмете (линије од 17. до 23.) и класа *izborno,* само за изборне предмете (линије од 24. до 30.).

У дефиницији сваке изведене класе, неопходно је назначити и назив основне класе. У овом примеру, заглавља дефиниција изведених класа

obavezno и *izborno* облика су: *class obavezno:public student* (линија 17.) и *class izborno:public student* (линија 24.)

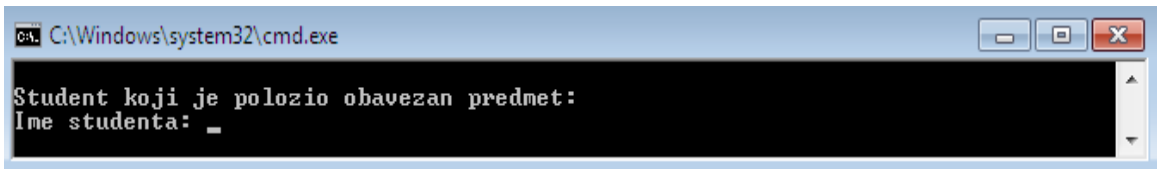
Следи кратак опис дефинисаних чланова свих класа овог програма. Основна класа *student* има приватне податке *ime, st_program* и *broj_ind* (линије од 10. до 12.) и јавне функције *citanje()* и *prikaz()* (линије од 14. до 15). Изведена класа *obavezno* има приватне податке *predmet_ob* и *ocena_ob* и јавне функције *citanje_ob()* и *prikaz_ob()* (линије од 18. до 22). Друга изведена класа *izborno* има приватне податке *predmet_izb* и *ocena_izb*, као и јавне функције *citanje_izb()* и *prikaz_izb()* (линије од 25. до 29).

На почетку програм из главне функције креира објекте, за класу *obavezno* објекат obav_obj и за класу *izborno* објекат izb_obj (линије од 32. до 33.). Следе позиви јавних функција *citanje_ob(), citanje_izb(), prikaz_ob()* и *prikaz_izb()*(линије од 35. до 45.), над одговарајућим објектима изведених класа.

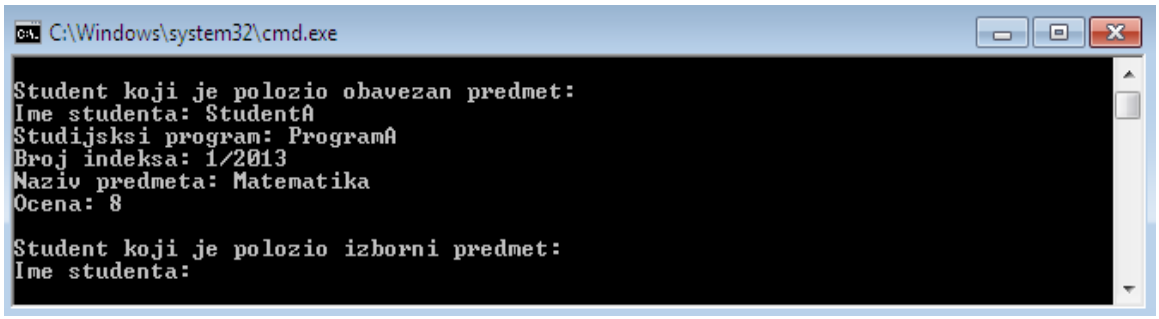
Дефиниције свих функција основне класе *student*, као и функција изведених класа *obavezno* и *izborno*, написане су након дефиниције главне функције (линије од 49. до 100.).

Функција *student::citanje()* чита са тастатуре податке о студенту (линије од 49. до 61.), а функција *student::prikaz()* приказује ове податке на екрану (линије од 62. до 66.). Свака од функција *obavezno::citanje()* (линије од 67. до 77.) и *izborno::citanje()* (линије од 84. до 94.) садржи на самом почетку позив функције *citanje()* своје основне класе *student*, а након тога приступа својим приватним подацима. Свака од функција *obavezno::prikaz()* (линије од 78. до 83.) и *izborno::prikaz()* (линије од 95. до 100.) садржи на самом почетку позив функције *prikaz()* своје основне класе *student*, а након тога приступа својим приватним подацима.

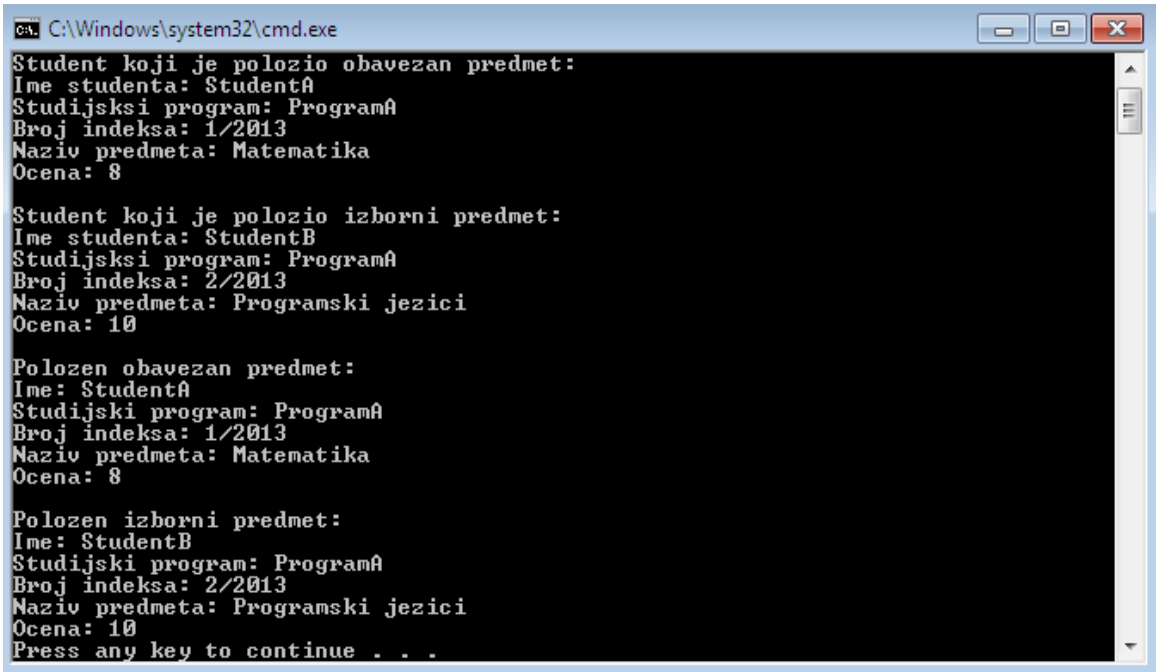
На сликама од 10.4 а) до 10.4 в) може се видети један могући сценарио извршавања овог програма.



Слика 10.4 а) Излаз програма **primer10_4** (први део)



Слика 10.4 б) Излаз програма **primer10_4** (други део)



Слика 10.4 в) Излаз програма **primer10_4** (трећи део)

Практични савети

- ✓ Писање објектно-оријентисано пројектованих програма на језику C++ тражи више од једноставног прекодирања програма са језика C на језик C++, због тога што је још у фази пројектовања програма уместо приступа "од функција надоле", потребно користити приступ "од података нагоре".
- ✓ Основну класу (класу родитеља) у објектно-оријентисано пројектованом програму треба осмислити тако, да представља самосталну целину са међусобно логички повезаним подацима и функцијама, али не толико самосталну да се њене особине не могу наслеђивати и користити на одговарајући начин у осталим, изведеним из ње класама програма.

Евентуалне грешке

- ➢ Члановима класе (приватним или јавним) немогуће је приступати без имена објекта и оператора приступа *(ime_objekta.ime_clana).*
- ➢ Подацима приватним члановима и функцијама приватним чланицама класе недозвољено је приступати из функција нечланица исте класе.
- ➢ Дефиниције функција чланица класе, написане ван дефиниције исте класе, не могу бити написане без назначеног њиховог припадања класи *(tip_povr_vrednosti ime_klase::ime_funkcije(lista_argumenata){...}).*

Задаци за припрему вежбе 10. у лабораторији

Задатак 1.

Написати програм на језику *C++*, који извршава задатак програма из примера 10.4 ове збирке (ажурирање података о студенту и оценама на испитима из обавезних и изборних предмета) уз допуну података о студенту и о испитима, као и уз распоређивање кода програма по датотекама заглављима и модулима. Резервисати датотеку заглавље (или датотеке заглавља) за дефиниције класа, а модуле за главну функцију и функције чланице класа.

Задаци за вежбу 10. у лабораторији

Предвидети у следећим програмима њихово објектно-оријентисано пројектовање. Испројектовати основну и изведене класе. Уз помоћ конструктора иницијализовати објекте класа, а уз помоћ деструктора исписати одговарајуће поруке при укидању истих објеката. Предвидети код изведених класа коришћење особина основне класе, као и увођење нових особина.

Задатак 1.

Написати програм на језику *C++* који:

а) иницијализује податке о кориснику рачуна у банци (подаци: ime, pin, stanje_na_racunu...),
б) поздравља корисника и тражи му да унесе pin,
в) проверава pin и у случају исправног уноса приказује на екрану izvestaj() о појединим корацима током разних промена стања на рачуну, које корисник иницира помоћу одговарајуће од понуђених опција: podizanje_novca(), polaganje_depozita()...

Задатак 2.

Написати програм на језику *C++* који:

а) иницијализује податке о сваком кориснику: ime, adresa, drzava...,
б) иницијализује податке о сваком возилу које је на располагању: tip, snaga, starost...
в) тражи од корисника његове податке, приказује на екрану податке о свим возилима и тражи му да изабере: kupovina(), iznajmljivanje(), kraj_programa(),
г) Након избора опције од сваког корисника, приказује на екрану izvestaj() о корисницима приватних возила, о корисницима возила за изнајмљивање, као и о карактеристикама возила сваког корисника.

Решени програмски задаци који се могу срести у пракси

Задатак 1.

Креирати датотеку *zapis.txt* у којој се налази произвољан текст и написати програм на језику *C* који шифрује и дешифрује текст из датотеке *zapis.txt* на следећи начин:

а) функција *sifrira* чита карактер по карактер из датотеке *zapis.txt* до краја датотеке и сваки карактер мења карактером чији *ASCII* кôд представља резултат ротационог померања *ASCII* кôда за једно место удесно од почетног податка, и шифровани садржај уписује у датотеку *sifra.txt.* (пример за ротационо померање: 10110101 се трансформише у 11011010, а 01101010 у 00110101...),

б) функција *desifrira* враћа шифровани текст у изворни текст.

Главна функција треба да прикаже на екрану садржај датотеке пре шифровања, после позива функције *sifrira*, као и после позива функције *desifrira*. Задатак приказивања садржаја датотеке на екрану такође треба издвојити у посебну функцију.

Решење 1.

```
/*Program za sifriranje i desifriranje teksta*/

#include <stdio.h>
#include <stdlib.h>

void sifruje();
void desifruje();
void prikaz_sadrzaja(char *ime);
void greska(char *poruka);

main()
{
        printf("*****************************************************\n");
        printf("Prikaz sadrzaja datoteke pre sifrovanja\n\n");
        prikaz_sadrzaja("zapis.txt");
        sifruje();
        printf("*****************************************************\n");
        printf("Sifrovani sadrzaj datoteke:\n\n");
        prikaz_sadrzaja("sifra.txt");
        printf("*****************************************************\n");
        printf("Desifrovani sadrzaj datoteke:\n\n");
        desifruje();
}

/*Sifrovanje:  rotaciono pomeranje za jedno mesto udesno*/
void sifruje()
{
        char c;
        unsigned short c1,pom;
        FILE *pt, *pt1;
```

```
        if(!(pt=fopen("zapis.txt","r+")))
                greska("Nije uspelo otvaranje datoteke za sifrovanje!\n");
        if(!(pt1=fopen("sifra.txt","w+")))
                greska("Nije uspelo otvaranje datoteke za sifrovanje!\n");
        while ((c=fgetc(pt))!= EOF)
        {
                c1=c;
                pom=1;
                pom=c1&pom;        /*sacuvan bit najmanje tezine*/
                c1>>=1;            /*pomeraj za jedno mesto udesno*/
                pom<<=7;
                c1=c1|pom;         /*postavljen bit najvece tezine*/
                c=c1;
                fputc(c,pt1);
        }
        fclose(pt);
        fclose(pt1);
}

/*Desifrovanje:  rotaciono pomeranje za jedno mesto ulevo*/
void desifruje()
{
        char c;
        unsigned short c1,pom;
        FILE *pt;

        if(!(pt=fopen("sifra.txt","r+")))
                greska("Nije uspelo otvaranje datoteke za desifrovanje!\n");
        while ((c=fgetc(pt))!= EOF)
        {
                c1=c;
                pom=1;
                pom<<=7;
                pom=pom&c1;        /*sacuvan bit najvece tezine*/
                pom>>=7;
                c1<<=1;            /*pomeraj za jedno mesto ulevo*/
                c1=c1|pom;         /*postavljen bit najmanje tezine*/
                c=c1;
                printf("%c",c);
        }
        fclose(pt);
}

/*Prikaz sadrzaja datoteke*/
void prikaz_sadrzaja(char ime[])
{
        FILE *pt;
        char c;
```

```
        if(!(pt=fopen(ime,"r")))
                greska("Nije uspelo otvaranje datoteke za citanje!\n");

        while ((c=fgetc(pt))!= EOF)
                printf("%c",c);
        printf("\n");

        fclose(pt);
}

void greska(char *poruka)
{
        printf("%s\n",poruka);
        exit(1);
}
```

Задатак 2.

Написати програм на језику *C* који у задатој текст датотеци замењује један низ знакова (*string1*) другим низом знакова (*string2*). При томе дати низови могу садржати и само по један карактер. Име текстуалне датотеке, као и низови знакова, задају се у командној линији.

После измена штампа се на екрану нов садржај датотеке.

Ако се зада само име датотеке на екрану се штампа оригинални садржај датотеке.

Ако се наведе само први знаковни низ, онда се тај знаковни низ брише из датотеке и штампа се на екрану нов садржај датотеке.

У осталим случајевима, било да се ради о више или мање аргумената, штампају се на екрану одговарајуће поруке.

Решење 2.

```
/*Program koji u datoj datoteci zamenjuje jedan niz znakova drugim*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

typedef enum {poziv,memorija,datoteka,zamena,argumenti} Poruke;

char *tekstovi_poruka[]={   "\nNema imena fajla\n",
                            "\nGreska prilikom dinamicke dodele memorije\n",
                            "\nGreska prilikom otvaranja datoteke\n",
                            "\nU tekstu se ne pojavljuje dati niz znakova\n",
                            "\nIma previse argumenata\n"                    };

void stampanje_poruke(Poruke status)
{
        fprintf(stderr,tekstovi_poruka[status]);
        exit(1);
}

main(int arg, char *argv[])
{
        FILE *fptr, *tmpfptr;
        unsigned broj_zamena,duzina1,duzina2,i,broj_znakova;
        char *tmp1, *tmp2, *p, *tmp;
        char c;

       /*naredba visestruke selekcije prema broju ucitanih argumenata*/
        switch(arg)
        {
                case 1:
                        stampanje_poruke(poziv);
                case 2:
```

```
                fptr=fopen(argv[1],"r");
                if(fptr==NULL)
                        stampanje_poruke(datoteka);

                while ((c=fgetc(fptr))!=EOF)
                        putchar(c);
                fclose(fptr);
                break;
        case 3:  /*naveden je i niz znakova koji se brise*/
        case 4: /*naveden je i niz znakova kojim se zamenjuje*/
                fptr=fopen(argv[1],"r");
                if(fptr==NULL)
                        stampanje_poruke(datoteka);

                broj_znakova=0;
                while((c=fgetc(fptr))!=EOF)
                        broj_znakova++;

                rewind(fptr);

                broj_zamena=0;
                duzina1=strlen(argv[2]);

                tmp1=malloc(broj_znakova+1);
                if(tmp1==NULL)
                        stampanje_poruke(memorija);

                /*sadrzaj cele datoteke se smesta u jedan niz znakova*/
                for(i=0;i<broj_znakova;i++)
                        *(tmp1+i)=fgetc(fptr);
                *(tmp1+i)='\0';
                p=tmp1;
                tmp=tmp1;

                /*prebrojavanje na koliko mesta treba obrisati odnosno
                 zameniti dati niz znakova*/
                 while(strstr(p,argv[2])!=NULL)
                 {
                        broj_zamena++;
                        p=strstr(p,argv[2])+duzina1;
                 }
                 if(broj_zamena==0)
                 {
                        puts(tmp1);
                        free(tmp1);
                        fclose(fptr);
                        stampanje_poruke(zamena);
                 }
                 /*brisanje niza znakova (zamena praznom niskom)*/
```

```
                    if(arg==4)
                        duzina2=strlen(argv[3]);
                    else
                        duzina2=0;

                    /*rezervacija mesta za izmenjeni niz znakova*/
                    tmp2=malloc(broj_znakova+1+
                        (duzina2-duzina1)*broj_zamena);
                    if(tmp2==NULL)
                        stampanje_poruke(memorija);
                    *tmp2='\0';

                    /*pronalazenje i zamena date niske znakova*/
                    while(1)
                    {
                        p=strstr(tmp,argv[2]);
                        if(p==NULL)
                            break;
                        strncat(tmp2,tmp,p-tmp);
                        if(arg==4)
                            strcat(tmp2,argv[3]);
                        tmp=p+duzina1;
                    }
                    strcat(tmp2,tmp);
                    puts(tmp2);
                    printf("\nUkupan broj izvrsenih zamena je %i\n",broj_zamena);

                    /*izmenjeni tekst se smesta u privremenu datoteku*/
                    tmpfptr=fopen("temp~","w");
                    if(tmpfptr==NULL)
                        stampanje_poruke(datoteka);

                    for(i=0;i<strlen(tmp2);i++)
                        fputc(*(tmp2+i),tmpfptr);

                    /*stara datoteka se brise a privremena datoteka se
                      preimenuje*/
                    fclose(fptr);
                    fclose(tmpfptr);
                    remove(argv[1]);
                    rename("temp~",argv[1]);
                    free(tmp1);
                    free(tmp2);
                    break;
                default:
                    stampanje_poruke(argumenti);
                    break;
        }
}
```

Задатак 3.

Написати програм на језику *C* који из командне линије прихвата имена трију датотека и од садржаја прве две датотеке, са подацима о запосленима у једној фирми, формира трећу на даље описан начин.

У прву датотеку уписују се подаци о запосленом представљени следећом структуром:

```
struct zaposleni{
        char imeIPrezime Radnika[40];
        int maticniBroj;
        int godineRadnogStaza;
        char strucnaSprema[45];
        double plata;
        }
```

У другу датотеку уписује се следећа структура за сваког радника:

```
struct licniPodaci{
        int maticniBroj;
        char adresa[40];
        char grad[30];
        long int brojTelefona;
        char opisPosla[50];
        }
```

Подаци се уносе све док се за матични број радника не унесе 0. Формирати трећу датотеку тако да садржи податке о радницима из града са именом задатим преко тастатуре: име и презиме, стручну спрему, плату и адресу.

Решење 3.

```
/*Program za evidenciju podataka o zaposlenima u jednoj firmi*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct zaposleni{
        char imeIPrezimeRadnika[40];
        int maticniBroj;
        int godineRadnogStaza;
        char strucnaSprema[45];
        double plata;
        };

struct licniPodaci{
        int maticniBroj;
        char adresa[40];
        char grad[30];
        long int brojTelefona;
        char opisPosla[50];
        };
struct podaci{
```

```
        char imeIPrezimeRadnika[40];
        char adresa[40];
        char strucnaSprema[45];
        double plata;
        };

void prikazuje(char *,int);
void greska(char g[]);

main(int argc, char *argv[])
{
        FILE *fpt1, *fpt2, *fpt3;
        struct zaposleni radnik;
        struct licniPodaci licni;
        char grad[30];
        struct podaci tekuci;
        double pl;
        int broj=0,br1=0,i;

        if(argc!=4)
                greska("Niste uneli potreban broj parametara.\n");

        if(!(fpt1=fopen(argv[1],"wb")))
                greska("Nije uspelo otvaranje prve datoteke za pisanje.\n");
        if(!(fpt2=fopen(argv[2],"wb")))
                greska("Nije uspelo otvaranje druge datoteke za pisanje.\n");

        while(1)
        {
                printf("\nUnesite maticni broj radnika.\n");
                scanf("%d",&radnik.maticniBroj);

                if(radnik.maticniBroj==0)
                        break;

                licni.maticniBroj=radnik.maticniBroj;
                printf("\nUnesite ime i prezime radnika ");
                getchar();
                gets(radnik.imeIPrezimeRadnika);
                printf("\nUnesite broj godina radnog staza radnika %s.",
                        radnik.imeIPrezimeRadnika);
                scanf("%d",&radnik.godineRadnogStaza);
                printf("\nUnesite strucnu spremu radnika %s.",
                        radnik.imeIPrezimeRadnika);
                scanf("%s",radnik.strucnaSprema);
                printf("\nDohodak radnika %s je: ",radnik.imeIPrezimeRadnika);
                scanf("%lf",&pl);radnik.plata=pl;
                printf("\nUnesite licne podatke o radniku %s.\nAdresa:",
                        radnik.imeIPrezimeRadnika);
                getchar();
```

```
            gets(licni.adresa);
            printf("\nGrad:");
            scanf("%s",licni.grad);
            printf("\nOpis posla: ");
            getchar();
            gets(licni.opisPosla);
            printf("\nBroj telefona: ");
            scanf("%d",&licni.brojTelefona);

            if(fwrite(&radnik, sizeof(struct zaposleni),1,fpt1)!=1)
                  greska("Greska pri upisu u prvu datoteku\n");
            if(fwrite(&licni,sizeof(struct licniPodaci),1,fpt2)!=1)
                  greska("Greska pri upisu u drugu datoteku\n");
            broj++;
      }
      fclose(fpt1);
      fclose(fpt2);

      if(!(fpt1=fopen(argv[1],"rb")))
            greska("Nije uspelo otvaranje prve datoteke za citanje.\n");
      if(!(fpt2=fopen(argv[2],"rb")))
            greska("Nije uspelo otvaranje druge datoteke za citanje.\n");
      if(!(fpt3=fopen(argv[3],"wb")))
            greska("Nije uspelo otvaranje trece datoteke za pisanje.\n");
      printf("\nUnesite ime grada iz koga zelite podatke o radnicima.\n");
      scanf("%s",&grad);

      for (i=0;i<broj;i++)
      {
            if(fread(&radnik, sizeof(struct zaposleni),1,fpt1)!=1)
                  greska("Greska pri citanju prve datoteke\n");
            if(fread(&licni,sizeof(struct licniPodaci),1,fpt2)!=1)
                  greska("Greska pri citanju druge datoteke\n");
            if(strcmp(grad,licni.grad)==0)
            {
                  strcpy(tekuci.imeIPrezimeRadnika,radnik.imeIPrezimeRadnika);
                  strcpy(tekuci.adresa, licni.adresa);
                  strcpy(tekuci.strucnaSprema, radnik.strucnaSprema);
                  tekuci.plata=radnik.plata;

                  if(fwrite(&tekuci,sizeof(struct podaci),1,fpt3)!=1)
                        greska("Greska pri upisu u drugu datoteku\n");
                  br1++;
            }
      }
      fclose(fpt1);
      fclose(fpt2);
      fclose(fpt3);

      prikazuje(argv[3],br1);
```

```
}

void prikazuje(char *ime, int br)
{
	FILE *ptr;
	int i;
	struct podaci tekuci;

	if(!(ptr=fopen(ime,"rb")))
		greska("Nije uspelo otvaranje datoteke za citanje.\n");

	printf("\nPodaci o odgovarajucim radnicima\n");
	for(i=0;i<br;i++){
		if(fread(&tekuci,sizeof(struct podaci),1,ptr)!=1)
				greska("Greska pri upisu u drugu datoteku\n");
		printf("\nIme i prezime: %s \n",tekuci.imeIPrezimeRadnika);
		printf("\nAdresa: %s",tekuci.adresa);
		printf("\nStrucna sprema: %s",tekuci.strucnaSprema);
		printf("\nLicni dohodak %.2f dinara \n", tekuci.plata);
		printf("\n");
	}

}

void greska(char g[])
{
	printf("%s",g);
	exit(1);
}
```

Задатак 4.

Структура обавезе је дефинисана на следећи начин:

```
struct obaveza{
        char   opis_obaveze[30];
        int dan, mesec, godina;
        int sat, minut;
        }
```

Написати програм на језику *C* који кориснику пружа следећи избор:

1. *Unos obaveze*
2. *Brisanje obaveze*
3. *Obaveze sa odredjenim datumom*
4. *Pregled svih unetih obaveza*
5. *Kraj rada*

и реализује изабрану опцију све док се не изабере опција 5.

Решење 4.

```
/*Program za evidenciju obaveza, po datumima i terminima*/

#include <stdio.h>
#include <stdlib.h>

struct  obaveza{
        char   opis_obaveze[30];
        int dan, mesec, godina;
        int sat, minut;
        };

struct cvor{
        struct obaveza ob;
        struct cvor *cv;
        };

struct cvor *brisi_obavezu(struct cvor *pocetak);

int prikaz_menija();
void prikazi_listu(struct cvor *pocetak);
void prikaz_dnevnih_obaveza(struct cvor *pocetak);

main()
{
        int op;
        struct cvor *pocetak=NULL, *tekuci, *novi;

        while(1)
        {
                op=prikaz_menija();
```

```
                switch(op)
                {
                case 1:
                        if ((novi=malloc(sizeof(struct cvor)))==NULL)
                        {
                                printf("Greska prilikom dodele memorije");
                                exit(1);
                        }

                        novi->cv=NULL;

                        printf("Unesite opis obaveze\n");
                        scanf("%s", novi->ob.opis_obaveze);
                        printf("Unesite datum obaveze u formi dd mm gggg ");
                        scanf("%d%d%d",   &novi->ob.dan,&novi->ob.mesec,
                                          &novi->ob.godina);
                        printf("Unesite sat i minut u formi ss mm ");
                        scanf("%d %d",&novi->ob.sat,&novi->ob.minut);

                        if(pocetak==NULL)  /*prvi element liste*/
                                pocetak=novi;
                        else                    /*kretanje kroz listu i dodavanje*/
                        {                       /*novog elementa na kraj*/
                                tekuci=pocetak;
                                while(tekuci->cv)
                                {
                                        tekuci=tekuci->cv;
                                }
                                tekuci->cv=novi;
                        }
                        break;
                case 2:
                        pocetak=brisi_obavezu(pocetak);
                        break;
                case 3:
                        prikaz_dnevnih_obaveza(pocetak);
                        break;
                case 4:
                        prikazi_listu(pocetak);
                        break;
                case 5:
                        printf("\n\n");
                        exit(1);
                default:
                        printf("Izabrali ste nepostojecu opciju!\n");
                        break;
                }
        }
}
```

```
int prikaz_menija()
{
	int opcija;

	printf("****************************************\n");
	printf("1. Unos obaveze\n");
	printf("2. Brisanje obaveze\n");
	printf("3. Obaveze sa odredjenim datumom\n");
	printf("4. Pregled svih obaveza\n");
	printf("5. Kraj rada\n");
	printf("****************************************\n");

	printf("Unesite redni broj opcije koju ste izabrali: ");
	scanf("%d",&opcija);

	return  opcija;
}

void prikazi_listu(struct cvor *pocetak)
{
	struct cvor *tekuci=pocetak;
	int brojac=1;

	while(tekuci)
	{
		printf("\n%d. Obaveza %s ",brojac,tekuci->ob.opis_obaveze);
		printf("datum obaveze: %d.%d.%d.",
				tekuci->ob.dan,tekuci->ob.mesec,tekuci->ob.godina);
		printf(" u %d sati i %d minuta.\n",tekuci->ob.sat,tekuci->ob.minut);
		tekuci=tekuci->cv;
		brojac++;
	}
}

struct cvor *brisi_obavezu(struct cvor *pocetak)
{
	struct cvor *tekuci=pocetak, *prethodni=pocetak;
	int br,j=1;

	prikazi_listu(pocetak);

	printf("Unesite redni broj obaveze koju zelite da izbacite\n");
	scanf("%d",&br);

	if(pocetak==NULL)		/*ispitivanje da li je lista prazna*/
	{
		printf("Nedozvoljeno brisanje!\n");
		return NULL;
	}
```

```
        if (br==1)
        {       /*treba izbaciti prvi element liste*/
                pocetak=tekuci->cv;
                free(tekuci);
                return pocetak;
        }
        else
        {
                while((tekuci->cv)&& j!=br)
                {
                        prethodni=tekuci;
                        tekuci=tekuci->cv;
                        j++;
                }
                if(tekuci->cv==NULL)
                        if(br==j)
                        {
                                prethodni->cv=NULL;
                                free(tekuci);
                                return pocetak;
                        }
                        else
                        {
                                printf("Uneli ste preveliki redni broj\n");
                                return pocetak;
                        }
                if(j==br)
                {
                                prethodni->cv=tekuci->cv;
                                free(tekuci);
                                return pocetak;
                }
                return NULL;
        }
}

void prikaz_dnevnih_obaveza(struct cvor *pocetak)
{
        struct cvor *tekuci=pocetak;
        int brojac=1,dan,mesec,godina;

        printf("Unesite datum za koji zelite pregled obaveza\n");
        scanf("%d%d%d",&dan,&mesec,&godina);

        while(tekuci)
        {
          if((dan==tekuci->ob.dan) &&(mesec==tekuci->ob.mesec)
                                && (godina == tekuci->ob.godina))
          {
                printf("\n%d. Obaveza %s ",brojac,tekuci->ob.opis_obaveze);
```

```
                printf("datum obaveze: %d.%d.%d.",
                        tekuci->ob.dan,tekuci->ob.mesec,tekuci->ob.godina);
                printf(" u %d sati i %d minuta.\n",tekuci->ob.sat,tekuci->ob.minut);

                brojac++;
            }
            tekuci=tekuci->cv;
        }
        if(brojac==1)
        {
                printf("Za uneti datum nema unetih obaveza!\n");
        }
}
```

Задатак 5.

Написати програм на језику C за игру погађања речи. Реч која се погађа је случајно изабрана из датотеке *reci.txt* која је претходно креирана. Игра се на следећи начин.

На екрану се налази онолико знакова доња црта (_) колико дата реч има слова. Од играча се тражи да унесе преко тастатуре слово. Ако се у задатој речи појављује то слово, свуда где га има, знак доња црта замењује се тим словом. Уколико реч не садржи унето слово приказује се одговарајућа порука на екрану.

Свако погођено слово доноси 5 поена, за свако промашено слово одузимају се 2 поена, а ако се поново унесе слово које је већ погађано одузима се 1 поен али се не сматра као промашај. Играч има право на 10 промашаја у току једног погађања. Такође, на екрану се стално налази информација о броју поена и о броју преосталих покушаја. Осим тога на екрану се налазе и слова која нису бирана. Када се изабере неко слово оно се скида са списка понуђених.

Ако после 10 промашаја играч не погоди реч приказује се на екрану одговарајућа порука и открива се задата реч. Уколико играч погоди реч на већ постојеће поене додаје се бонус који се рачуна као преостали број покушаја помножен бројем 2.

По завршеном погађању једне речи играчу се нуди могућност да настави да игра, односно да погађа нову реч или да изађе из игре. Ако настави да игра, поени се додају на претходно постигнути резултат. По изласку из игре појављује се на екрану листа 15 најбољих резултата свих оних који су до тада играли. Ако их нема 15 приказује се онолико колико их има.

Решење 5.

```
/*Program za igru pogadjanja reci*/

#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<ctype.h>
#include<time.h>
#define BROJ_POKUSAJA 10
#define POGODAK 5
#define PROMASAJ -2
#define BONUS 2
#define KAZNA 1

typedef enum {MEMORIJA,DATOTEKA,CITANJE,UPIS}Greska;

char *poruke[]={    "\nProblem sa memorijom\n",
                    "\nProblem sa datotekom\n",
                    "\nProblem sa citanjem podataka\n",
                    "\nProblem sa upisivanjem podataka\n"};
```

```
typedef struct {
        char ime[20];
        int bodovi;
        } Igrac;

void poruka(GRESKA);
void lista(int);

main()
{
        FILE *fptr;
        unsigned i,j,poeni=0,nadjen,ima,ind;
        char *rec, *copy,c;
        char slova1[]="A B C D E F G H I J K L M N O P R S T U V Z",slova[44];
        time_t timer;

        /*otvaranje datoteke u kojoj se nalaze zadate reci */
        fptr=fopen("reci.txt","r");
        if(fptr==NULL)
                poruka(DATOTEKA);

        /*pokretanje generatora slucajnih brojeva pomocu tekuceg vremena*/
        srand(time(&timer));

        while(1)
        {
                strcpy(slova,slova1);
                while (!(i=rand()));
                        rec=malloc(15);
                if(rec==NULL)
                        poruka(MEMORIJA);
                j=0;

                /*pomocu slucajnog broja ucitava se rec iz datoteke*/
                while(j<i)
                {
                        if(!fscanf(fptr,"%s",rec))
                                poruka(CITANJE);
                        j++;
                        if(feof(fptr))
                                rewind(fptr);
                }

                /*pravi se rec od donjih crta*/
                copy=malloc(strlen(rec)+1);
                if(copy==NULL)
                        poruka(MEMORIJA);
                realloc(rec,strlen(rec)+1);
                for(i=0;i<strlen(rec);i++)
                        *(copy+i)='_';
```

```
*(copy+i)='\0';

/*pocinje pogadjanje*/
for(i=0;i<BROJ_POKUSAJA;i++)
{
        printf("\n\n\nPOGADJA SE REC");
        printf("\t\tbroj poena: %i\t\tpreostalo pokusaja: %i\n",
                poeni,BROJ_POKUSAJA-i);
        printf("\nSlova na raspolaganju: %s\n\n",slova);
        for(j=0;j<strlen(copy);j++)
                printf("%c ",*(copy+j));
        putchar('\n');

        /*proverava se da li je uneti znak slovo*/
        while(1)
        {
                printf("\nUnesi slovo: ");
                scanf("%c",&c);
                c=toupper(c);
                getchar();

                if(c>'Z'||c <'A'||c=='Q'||c=='W'||c=='Y'||c=='X')
                        printf("\t\tNije slovo");
                else
                        break;
        }

        /*ako je slovo nadjeno brise se sa spiska ponudjenih*/
        nadjen=0;
        for(j=0;j<strlen(slova);j++)
                if(c==slova[j])
                {
                        nadjen=1;
                        slova[j]=' ';
                        break;
                }
                if(nadjen==0)
                {
                        printf("\t\tTo slovo je proslo");
                        ima=2;
                        poeni+=KAZNA;
                        i--;
                }

                /*nadjeno slovo se upisuje u rec umesto donje crte*/
                else
                {
                        ima=0;
                        for(j=0;j<strlen(rec);j++)
                                if(c==*(rec+j))
```

```
                                        {
                                                *(copy+j)=c;
                                                ima=1;
                                        }
                                }
                                if(ima==1)
                                {
                                        poeni+=POGODAK;
                                        i--;
                                }
                                else
                                {
                                        poeni+=PROMASAJ;
                                        if(ima!=2)
                                                printf("\t\tNema tog slova");
                                }

                                if(ind=(strcmp(rec,copy)==0))
                                {
                                    printf("\nBRAVO,pogodio si rec %s\n",rec);
                                    poeni=poeni+(BROJ_POKUSAJA-i-1)*BONUS;
                                    printf("\nBonus: %i poena, ukupno : %i poena\n",
                                            (BROJ_POKUSAJA-i-1)*2,poeni);
                                    break;
                                }
                        }
                if(!ind) printf("\nZao mi je, nisi pogodio rec %s iz %i puta\n",rec, i);
                        printf("\nZelis li jos da pogadjas (d/n)? ");
                scanf("%c",&c);
                getchar();
                if(c=='n' || c=='N')
                {
                        printf("\n\n");
                        break;
                }
        }
        fclose(fptr);
        lista(poeni);
}

void lista(int poeni)
{
        FILE *fptr;
        int i,j,mesto,plasman=0;
        char *novoime;
        Igrac *igrac,tmpigrac;

        printf("\nBroj poena %i\n",poeni);
        printf("\nTOP 15\n");
        fptr=fopen("najbolji.dat","rb+");
```

```
/*slucaj ako niko pre nije igrao*/
 if(fptr==NULL)
 {
        mesto=1;
        fptr=fopen("najbolji.dat","wb");
        if(fptr==NULL)
                poruka(DATOTEKA);
        printf("\nUnesi ime: ");
        fflush(stdin);
        gets(tmpigrac.ime);
        tmpigrac.bodovi=poeni;
        if(fwrite(&tmpigrac,sizeof(Igrac),1,fptr)!=1)
                poruka(UPIS);
        fclose(fptr);
        printf("\n1. %-20s  %i\n",tmpigrac.ime,poeni);
        printf("\nTrenutno se nalazis na %i. mestu\n",mesto);
 }

/*ispisivanje na ekranu liste rezultata*/
 else
 {
        igrac=malloc(15*sizeof(Igrac));
        if(igrac==NULL)
                poruka(MEMORIJA);
        for(j=0;j<15;j++)
        {
                if(fread(igrac+j,sizeof(Igrac),1,fptr)!=1 && !feof(fptr))
                        poruka(CITANJE);
                if(feof(fptr))
                        break;
                printf("\n%2i. %-20s %i",j+1,(igrac+j)->ime,(igrac+j)->bodovi);
        }
        putchar('\n');
        if(j<15) j++;

       /*ako se rezultat nalazi u prvih petnaest
         ubacivanje tog rezultata u listu*/
        if((igrac+j-1)->bodovi<poeni)
        {
                plasman=1;
                mesto=j;
                printf("\nUnesite ime ");
                novoime=malloc(20);
                if(novoime==NULL)
                        poruka(MEMORIJA);
                fflush(stdin);
                gets(novoime);
                (igrac+j-1)->bodovi=poeni;
                strcpy((igrac+j-1)->ime,novoime);
```

```
                        for(i=j-1;i>0;i--)
                                if((igrac+i)->bodovi>(igrac+i-1)->bodovi)
                                {
                                        mesto--;
                                        tmpigrac=*(igrac+i);
                                        *(igrac+i)= *(igrac+i-1);
                                        *(igrac+i-1)=tmpigrac;
                                }
                                else
                                        break;
                }

                rewind(fptr);
                if(plasman)
                {
                    printf("\n\nTOP 15\n\n");
                    for(i=0;i<j;i++)
                    {
                        if(fwrite(igrac+i,sizeof(Igrac),1,fptr)!=1) poruka(UPIS);
                          printf("%2i. %-20s  %i\n",i+1,(igrac+i)->ime,(igrac+i)->bodovi);
                    }
                    printf("\n\nTrenutno se nalazis na %i. mestu\n\n",mesto);
                }
                else
                        printf("\n\n");

                fclose(fptr);
                getchar();
        }
}

void poruka(Greska status)
{
        fprintf(stderr,poruke[status]);
        exit(1);
}
```

Задатак 6.

У датотеци *mob.bin* налазе се подаци о мобилним телефонима где је сваки модел представљен следећом структуром:

```
struct mob{
        char ime_modela[20];
        int kolicina;
        float cena;
        };
```

Написати програм на језику *C* који кориснику пружа следећи избор:

1. *Unos podataka*
2. *Kupovina*
3. *Kraj*

и реализује изабрану опцију на следећи начин:

1. корисник је изабрао опцију 1. за унос података. Проверити да ли датотека *mob.bin* већ постоји. У случају да постоји треба отворити датотеку тако да се претходни садржај сачува. Затим треба кориснику понудити да унесе податке за дати модел (пожељно је кориснику написати текст о томе који податак уноси). По уносу, ако се дати модел телефона већ налази у датотеци *mob.bin* и има исту цену треба само повећати количину, а ако се цена разликује, или је нови модел у питању, на крај датотеке треба додати нову набавку;
2. корисник је изабрао опцију 2. за куповину. Тражити од корисника да унесе назив модела. Претражити датотеку и у случају да се тај модел налази у датотеци *mob.bin* написати цену кориснику и питати га за количину коју жели да купи. Затим проверити да ли има довољно апарата. У случају да има довољан број апарата, умањити количину и написати кориснику колико треба да плати, у супротном приказати поруку о максималном броју апарата који може купити. Ако после обављене куповине нема више ни једног апарата тог модела избрисати ту структуру из датотеке. Ако је корисник унео модел који се не налази у датотеци треба га обавестити да таквог модела тренутно нема;
3. кориснику приказивати опције све док не изабере опцију 3.

Решење 6.

```
/*Program koji azurira podatke o mobilnim telefonima*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

struct mob{
        char ime_modela[20];
        int kolicina;
        float cena;
        };

struct mob model;
```

```
int broj_modela=0;

int prikaz_menija();
void unos_podataka();
void unos();
void kupovina();

main()
{
	int opcija;

	while(1)
	{
	    opcija=prikaz_menija();
	    switch(opcija)
	    {
		case 1:
			unos_podataka();
			break;
		case 2:
			kupovina();
			break;
		case 3:
			exit(1);
		default:
			printf("Izabrali ste nepostojecu opciju, birajte ponovo!\n");
			break;
	    }
	}
	return  0;
}

int prikaz_menija()
{
	int opcija;

	printf("****************************************\n");
	printf("1. Unos podataka\n");
	printf("2. Kupovina\n");
	printf("3. Kraj\n");
	printf("****************************************\n");
	printf("Unesite redni broj opcije koju ste izabrali:");
	scanf("%d",&opcija);

	return  opcija;
}


void unos_podataka()
{
```

```
struct mob *niz_struct;
FILE *pt1, *pt;
int i, postoji=1, unesen=0;
/*postoji=1 ako je mob.bin kreirana, inace je 0*/

if((pt1=fopen("mob.bin","rb"))==NULL)
        postoji=0;
else
        fclose(pt1);

/*mob.bin je prethodno kreirana*/
if((postoji==1) && (broj_modela!=0))
{
        if((pt=fopen("mob.bin","rb"))==NULL)
        {
                printf("Greska pri otvaranju datoteke za pisanje!\n");
                exit(1);
        }

        unos();
        if((niz_struct=malloc(sizeof(struct mob)*broj_modela))==NULL)
        {
                printf("Greska pri dodeli memorije\n");
                exit(1);
        }
        if(fread(niz_struct,sizeof(struct      mob),broj_modela,pt)!=broj_modela)
        {
                printf("Greska pri citanju");
                exit(1);
        }
        fclose(pt);

        for(i=0;i<broj_modela;i++)
          /*1. model je u mob.bin i cena je ista*/
          if(strcmp(model.ime_modela,niz_struct[i].ime_modela)==0)
                if(niz_struct[i].cena==model.cena)
                {
                        niz_struct[i].kolicina+=model.kolicina;
                        if((pt1=fopen("mob.bin","wb"))==NULL)
                        {
                            printf("Greska pri otvaranju datoteke za pisanje!\n");
                            exit(1);
                        }

                        if(fwrite(niz_struct,sizeof(struct mob),broj_modela,pt1)
                                !=broj_modela)
                        {
                                printf("Greska\n");
                                exit(1);
                        }
```

```
                        fclose(pt1);
                        unesen=1;
                        break;
                    }/*kraj 1. slucaja*/

                    /*2. model nije upisan u mob.bin, ili se cene razlikuju*/
                    if (unesen==0)
                    {
                      broj_modela++;
                      if((pt1=fopen("mob.bin","ab"))==NULL)
                      {
                            printf("Greska pri otvaranju datoteke za pisanje!\n");
                            exit(1);
                      }
                      if(fwrite(&model,sizeof(struct mob),1,pt1)!=1)
                      {
                            printf("Greska\n");
                            exit(1);
                      }
                      fclose(pt1);
                    }/*kraj 2. slucaja*/

            free(niz_struct);
        }

        /*mob.bin nije prethodno kreirana, unosi se prvi model*/
        else
        {
            if((pt=fopen("mob.bin","wb"))==NULL)
            {
                printf("Greska pri otvaranju datoteke za pisanje!\n");
                exit(1);
            }
            unos();
            printf("Model %s komada: %d cena: %f\n",
            model.ime_modela,model.kolicina,model.cena);
            if(fwrite(&model,sizeof(struct mob),1,pt)!=1)
            {
                printf("Greska\n");
                exit(1);
            }
            broj_modela++;
            fclose(pt);
        }
}
void kupovina()
{
        FILE *pt;
        struct mob *niz;
        int i,j,postoji=0,kol;
```

```
float racun;
struct mob temp;

if((niz=malloc(sizeof(struct mob)*broj_modela))==NULL)
{
        printf("Greska pri dodeli memorije\n");
        exit(1);
}
if((pt=fopen("mob.bin","rb"))==NULL)
{
        printf("Greska pri otvaranju datoteke za citanje!\n");
        exit(1);
}
if(fread(niz,sizeof(struct mob),broj_modela,pt)!=broj_modela)
{
        printf("Greska pri citanju");
        exit(1);
}
for(i=0;i<broj_modela;i++)
        printf("Model %s komada: %d cena: %f\n",
                niz[i].ime_modela,niz[i].kolicina,niz[i].cena);
fclose(pt);

printf("\nUnesite ime modela aparata koji zelite da kupite:");
scanf("%s",temp.ime_modela);

for(i=0;i<broj_modela;i++)
        if(strcmp(temp.ime_modela,niz[i].ime_modela)==0)
        {
                postoji=1;
                printf("Cena %s modela je %f.\n",niz[i].ime_modela,niz[i].cena);

                printf("Koliko komada zelite da kupite\n");
                scanf("%d",&kol);
                if(kol>niz[i].kolicina)
                {
                  printf("Mozete kupiti maksimalno %d aparata\n",niz[i].kolicina);
                  break;
                }
                racun=kol*niz[i].cena;

                printf("Vas racun iznosi %f dinara.\n",racun);
                niz[i].kolicina-=kol;   /*azurira se broj aparata posle prodaje*/
                if(niz[i].kolicina==0) /*brisu se podaci o modelu*/
                                       /*kog vise nema u ponudi*/
                {
                        broj_modela--;
                        for(j=i;j<broj_modela;j++)
                        {
                                strcpy(niz[j].ime_modela,niz[j+1].ime_modela);
```

```
                                        niz[j].cena=niz[j+1].cena;
                                        niz[j].kolicina=niz[j+1].kolicina;
                                }
                        }
                        if((pt=fopen("mob.bin","wb"))==NULL)
                        {
                                printf("Greska pri otvaranju datoteke za upis!\n");
                                exit(1);
                        }
                        if(fwrite(niz,sizeof(struct      mob),broj_modela,pt)!=broj_modela)
                        {
                                printf("Greska\n");
                                exit(1);
                        }
                        fclose(pt);
                        break;
                }
                if(postoji==0)
                        printf("Model %s nije trenutno u ponudi\n", temp.ime_modela);

                free(niz);
}
void unos()
{
        printf("Unesite naziv modela: ");
        scanf("%s",model.ime_modela);
        printf("\nUnesite broj aparata: ");
        scanf("%d",&model.kolicina);
        printf("\nUnesite cenu %s modela: ", model.ime_modela);
        scanf("%f",&model.cena);
        printf("\n");
}
```

Задатак 7.

Написати програм на језику *C* који ће отпратити једно такмичење у скијашким скоковима.

Скакаонице се деле у три групе: нормалне, високе и летаонице. Програм треба да обрађује само прве две.

Свака скакаоница има своју К-тачку. За нормалне скакаонице К-тачка је између 75 и 99 метара а за високе скакаонице је од 100 до 185 метара.

Број поена за сваки скок добија се као збир поена на основу дужине скока и поена на основу оцена судија.

Поени за дужину скока рачунају се по следећој формули:

(дужина скока – К-тачка) x коефицијент,

где је коефицијент за нормалне скакаонице 2.0 а за високе 1.8.

Поени за оцене судија добијају се на даље описан начин. Скок оцењује пет судија и сваки од њих даје оцену од 3 до 20. Од тих оцена одбацује се најмања и највећа а остале се сабирају. Сваки скакач скаче два пута изузев оних који су дисквалификовани после прве серије. Такмичар је дисквалификован уколико је имао неправилан скок или има негативне поене. Такмичар може бити дисквалификован и у другом скоку.

Сви подаци уносе се преко тастатуре. На почетку програма уносе се К-тачка и подаци о такмичарима по редоследу скакања. За сваког такмичара уносе се: име, презиме и земља за коју скаче.

У првој серији програм даје редом имена учесника, уноси се дужина скока и оцене судија за тај скок. Рачунају се бодови и даје се извештај на екрану о броју бодова и о тренутној позицији на којој се налази такмичар после изведеног скока.

У другој серији такмичари скачу обрнутим редоследом у односу на освојену позицију, односно, прво скаче последње пласирани а последњи скаче првопласирани после прве серије. Оцењује се и други скок, затим се сабирају поени из обе серије и на основу тога се одређује коначан пласман за сваког скакача.

На сваких десет скокова даје се ранг-листа оних који су до тада скочили а на крају сваке серије комплетна ранг-листа.

Решење 7.

```
/*Program koji prati takmicenje u skijaskim skokovima*/

#include <stdio.h>
#include <stdlib.h>
#define NH 2.0
#define LH 1.8
#define BROJ_SUDIJA 5
#define BROJ_SERIJA 2
#define MINIMALNA_OCENA 3
#define MAKSIMALNA_OCENA 20
#define KRITICNA_TACKA 185

typedef struct{
```

```
        char ime[15], prezime[15], zemlja [4];
        int startna_pozicija,izvedena_serija;
        float bodovi;
        }Skakaci;

int K_tacka;
float koef;

void ucitavanje (Skakaci [],int);
void stampanje (Skakaci [],int);
void Ktacka(void);
float skok(Skakaci *);
int plasman(Skakaci [],int,int);
void izvestaj(float,int);

main()
{
        Skakaci *takmicari;
        int i,broj_takmicara,trenutna_pozicija,izvedeni_skokovi;
        float broj_bodova;

        Ktacka();

        while(1)
        {
                printf("\nUkupan broj skakaca: ");
                scanf("%i",&broj_takmicara);
                if(broj_takmicara>0)
                        break;
                printf("\nNepravilno unesen broj skakaca\n");
        }

        takmicari=malloc(broj_takmicara*sizeof(Skakaci));
        if(takmicari==NULL)
        {
                printf("\nNema dovoljno raspolozive memorije !\n");
                exit(1);
        }

        ucitavanje(takmicari,broj_takmicara);

        printf("\nSKACE SE PRVA SERIJA\n\n");
        izvedeni_skokovi=0;
        for(i=0;i<broj_takmicara;i++)
        {
                broj_bodova=skok(takmicari+i);
                trenutna_pozicija=plasman(takmicari,i,broj_takmicara);
                izvestaj(broj_bodova,trenutna_pozicija);
                izvedeni_skokovi++;
                if(izvedeni_skokovi%10==0)
```

```
                {
                        printf("\nTrenutna rang lista prve serije\n");
                        stampanje(takmicari,i+1);
                }
        }

        printf("\nPlasman posle prve serije\n");
        stampanje(takmicari,broj_takmicara);

        printf("\nSKACE SE DRUGA SERIJA\n\n");
        izvedeni_skokovi=0;

        for(i=broj_takmicara-1;i>=0;i--)
        {
                if((takmicari+i)->izvedena_serija!=BROJ_SERIJA)
                {
                        broj_bodova=skok(takmicari+i);
                        trenutna_pozicija=plasman(takmicari,i,broj_takmicara);
                        izvestaj(broj_bodova,trenutna_pozicija);
                        i=broj_takmicara-1;
                        izvedeni_skokovi++;
                        if(izvedeni_skokovi%10==0)
                        {
                                printf("\nTrenutna rang lista prve serije\n");
                                stampanje(takmicari,izvedeni_skokovi);
                        }
                }
        }
        printf("\nKonacan plasman posle druge serije\n");
        stampanje(takmicari,broj_takmicara);
        printf("\n\n");

        free (takmicari);

        return 0;
}

/*unosi se K-tacka i na osnovu nje odredjuje koeficijent*/
void Ktacka(void)
{
        while(1)
        {
                printf("\nUnesite K-tacku skakaonice ");
                scanf("%i",&K_tacka);
                if(K_tacka<75 || K_tacka>185)
                        printf("\nNepravilno unesena K-tacka!\n");
                else
                        break;
        }
```

```
        if(K_tacka<100)
                koef=(float)NH;
        else
                koef=(float)LH;
}

/*unose se podaci o skakacima i odredjuje startna pozicija za prvi skok*/
void ucitavanje (Skakaci skakaci[],int broj_skakaca)
{
        int i;

        printf("\nUnosimo startnu listu:\n");
        for(i=0;i<broj_skakaca;i++)
        {
                printf("\n");
                printf("Ime: ");
                scanf("%s",skakaci[i].ime);
                printf("Prezime: ");
                scanf("%s",skakaci[i].prezime);
                printf("Zemlja: ");
                scanf("%s",skakaci[i].zemlja);
                skakaci[i].bodovi=0;
                skakaci[i].izvedena_serija=0;
                skakaci[i].startna_pozicija=i+1;
        }
}

/*prikazuje se na ekranu trenutna rang lista posle n skokova*/
void stampanje (Skakaci skakaci[],int n)
{
        int i;

        printf("\nPlasman posle %i. takmicara\n\n",n);
        for(i=0;i<n;i++)
        {
                printf("%2i. ",i+1);
                printf("%-10s ",skakaci[i].ime);
                printf("%-10s ",skakaci[i].prezime);
                printf("(%s)\t\t",skakaci[i].zemlja);
                if(skakaci[i].bodovi==0)
                        printf("-.-\n");
                else
                        printf("%.2f\n",skakaci[i].bodovi);
        }
}

/*ocenjivanje pojedinacnog skoka*/
float skok(Skakaci *skakac)
{
        int i,imax,imin;
```

```
float ocene[BROJ_SUDIJA],duzina_skoka,suma;

printf("\n%i. ",skakac->startna_pozicija);
printf("%s ",skakac->ime);
printf("%s ",skakac->prezime);
printf("(%s)\n",skakac->zemlja);

while(1)
{
        printf("\nUnesite duzinu skoka: ");
        scanf("%f",&duzina_skoka);
        if(duzina_skoka>=0 && duzina_skoka<KRITICNA_TACKA)
                break;
        printf("\nNepravilno unesena duzina skoka\n");
}

/*diskvalifikovani takmicar ima duzinu skoka 0 i broj bodova je 0*/
if(duzina_skoka==0)
{
        skakac->bodovi=0;
        skakac->izvedena_serija=BROJ_SERIJA;
}
/*unose se ocene sudija*/
else
{
        skakac->izvedena_serija++;
        printf("\nOcene sudija:\n");
        for(i=0;i<BROJ_SUDIJA;i++)
        {
                printf("%i. sudija: ",i+1);
                scanf("%f",&ocene[i]);
                if(ocene[i]<MINIMALNA_OCENA ||
                ocene[i]>MAKSIMALNA_OCENA )
                {
                        printf("\nNepravilno\n");
                        i=i-1;
                }
        }

        imax=BROJ_SUDIJA-1;
        imin=0;
        for(i=0;i<BROJ_SUDIJA;i++)
        {
                if(ocene[i]<ocene[imin])
                        imin=i;
                if(ocene[BROJ_SUDIJA-1-i]>ocene[imax])
                        imax=BROJ_SUDIJA-1-i;
        }
        /*odredjuje se ukupan broj poena*/
        suma=0;
```

```
		for(i=0;i<BROJ_SUDIJA;i++)
			if (i!=imax && i!=imin)
				suma=suma+ocene[i];

		suma=suma+(duzina_skoka-K_tacka)*koef;
		/*ukoliko su poeni negativni takmicar je diskvalifikovan*/
		if(suma<0)
		{
			skakac->bodovi=0;
			skakac->izvedena_serija=BROJ_SERIJA;
		}
		else
			skakac->bodovi=skakac->bodovi+suma;
	}

	return (skakac->bodovi);
}

/*odredjuje se trenutni plasman skakaca*/
int plasman(Skakaci skakaci[],int k,int broj_skakaca)
{
	int i;
	Skakaci temp;

	if(k==broj_skakaca-1 || skakaci[k].bodovi>skakaci[k-1].bodovi)
	{	for(i=k;i>0;i--)
			if(skakaci[i].bodovi>skakaci[i-1].bodovi)
			{	temp=skakaci[i];
				skakaci[i]=skakaci[i-1];
				skakaci[i-1]=temp;
			}
			else
				break;

		return i+1;
	}
	else
	{
		for(i=k;i<broj_skakaca-1;i++)
			if(skakaci[i].bodovi<skakaci[i+1].bodovi)
			{	temp=skakaci[i];
				skakaci[i]=skakaci[i+1];
				skakaci[i+1]=temp;
			}
			else
				break;
		return (i+1);
	}
}
```

```
void izvestaj(float broj_bodova, int plasman)
{
	printf("\nUkupan broj bodova je: %.2f\n",broj_bodova);
	if(broj_bodova!=0)
		printf("\nTrenutna pozicija: %i\n",plasman);
	else
		printf("\nTrenutna pozicija: bez plasmana\n");
}
```

Задатак 8.

Написати програм на језику *C* за вођење евиденције о резервацијама термина за хотел у току једне сезоне.

На екрану треба да буде стално понуђен основни избор:

1. *Kreiranje baze*
2. *Listanje Baze*
3. *Rezervacija sobe*
4. *Potvrda rezervacije*
5. *Otkazivanje rezervacije*

0 Izlaz

све док се не изабере опција 0 за излаз.

Пре првог појављивања основног екрана са тастатуре се уноси име хотела и сезона.

У опцији 1. креира се датотека хотела. Преко тастатуре редом уносе се за сваку собу број и тип а затим се ти подаци уписују у датотеку. Тип подразумева број кревета у соби или *apt* за апартман. Крај уноса је када се за тип собе унесе празан ред (*Enter*).

У опцији 2. читају се подаци из датотеке и штампају на екрану. Редом се даје извештај за сваку собу: број собе, тип собе, термини када је соба слободна и термини када је соба заузета и на које име је извршена резервација.

У опцији 3. врши се резервација собе. Преко тастатуре се уноси број собе. Ако је исправно унет број на екрану се појављује порука о датумима када је соба слободна. После тога, ако има слободних термина преко тастатуре се уноси име на које се врши резервација и то се уписује у датотеку хотела.

У опцији 4. преко тастатуре се уноси име а у бази хотела се проверава да ли постоји резервација на то име и у којим терминима.

У опцији 5. се преко тастатуре уноси име особе за коју се врши отказивање резервације. Уколико постоји резервација у датотеци се ослобађају термини који су били резервисани.

Предвидети приказ на екрану свих потребних извештаја о раду појединих делова програма.

Решење 8.

```
/*Program za evidenciju rezervacija soba u hotelu u toku jedne sezone*/

#include<stdio.h>
#include<stdlib.h>
#include<ctype.h>
#include<string.h>
#define KRAJ 0
#define PORUKE_MENIJA 7
#define OD_GODINE 2000
#define DO_GODINE 2015
#define MAX_IME 30
#define DANI 366
#define DUZINA 5
#define MESECI 12
```

```
int dani_u_mesecu[31];

char dani_u_godini[DANI][MAX_IME];

char *meseci_u_godini[]={ "januar","februar","mart","april","maj","juni","juli",
                          "avgust","septembar","oktobar","novembar","decembar"};

int broj_dana[]={31,28,31,30,31,30,31,31,30,31,30,31};

typedef struct {
        char broj_sobe[DUZINA];
        char tip_sobe[DUZINA];
        char dani[DANI][MAX_IME];
        } Hotel;

int prestupna(int);
void niz_datuma(int);
int meni(void);
int baza_hotela (char *);
int slobodni_termini(Hotel);
int pregled_baze(char *);
int rezervacija_sobe(char *);
int provera_datuma(int,int);
int potvrda_rezervacije(char *);
int otkazivanje(char *);
int rezervisano(Hotel);
void pisanje_poruka(int);

main()
{
        int i,izbor,status;
        char ime_hotela[MAX_IME],godina[DUZINA];
        char *ime_datoteke;

        for(i=0; i<31; i++)
                dani_u_mesecu[i] = i+1;

        printf("\nUnesite ime hotela: ");
        gets(ime_hotela);

        while(1)
        {
                printf("\nUnesite godinu (%i - %i):  ",OD_GODINE,DO_GODINE);
                gets(godina);
                if (OD_GODINE<=atoi(godina) && atoi(godina)<=DO_GODINE)
                        break;
                printf("\nGodina je van opsega \n");
        }
```

```
	ime_datoteke=malloc(strlen(ime_hotela)+strlen(godina)+2);
	if(ime_datoteke==NULL)
	{
		pisanje_poruka(1);
		exit(1);
	}

	strcpy(ime_datoteke,ime_hotela);
	strcat(ime_datoteke,"_");
	strcat(ime_datoteke,godina);

	niz_datuma(atoi(godina));

	do
	{
		printf("\n\n HOTEL %s - godina %s\n\n",ime_hotela,godina);
		izbor=meni();
		switch(izbor)
		{
			case KRAJ:	break;

			case 1:		status=baza_hotela(ime_datoteke);
					pisanje_poruka(status);
					break;

			case 2:		status=pregled_baze(ime_datoteke);
					if(status)
						pisanje_poruka(status);
					break;

			case 3:		status=rezervacija_sobe(ime_datoteke);
					if(status>=0)
						pisanje_poruka(status);
					break;

			case 4:		status=potvrda_rezervacije(ime_datoteke);
					if(status)
						pisanje_poruka(status);
					break;

			case 5:		status=otkazivanje(ime_datoteke);
					if(status)
						pisanje_poruka(status);
					break;

			default:	printf("\nNepoznat izbor\n");
					break;
		}
	}
	while(izbor!=KRAJ);
```

```
	printf("\n\nKRAJ PROGRAMA\n\n");
}

int meni(void)
{
	int i,izbor;
	char *poruke[]={	"1. Kreiranje baze hotela",
				"2. Listanje baze",
				"3. Rezervacija sobe",
				"4. Provera rezervacije",
				"5. Otkazivanje rezervacije",
				"0. Izlaz",
				"Vas izbor: "			};

	for(i=0;i<PORUKE_MENIJA-1;i++)
		puts(poruke[i]);
	printf("\n%s",poruke[PORUKE_MENIJA-1]);
	scanf("%i",&izbor);

	return izbor;
}

int prestupna(int god)
{
	int prestup;

	if(god%400==0) prestup=1;
	else if(god%100==0)
		prestup=0;
	else if(god%4==0)
		prestup=1;
	else
		prestup=0;

	return prestup;
}

/*formira se niz datuma, redom, od 1. januara do 31. decembra*/
void niz_datuma(int god)
{
	char *temp;
	int index=0,i,j;

	temp=malloc(MAX_IME+1);
	broj_dana[1]+=prestupna(god);

	for(i=0;i<MESECI;i++)
		for(j=0;j<broj_dana[i];j++)
		{
			itoa(dani_u_mesecu[j], temp, 10);
```

```
                        strcat(temp, ". ");
                        strcat(temp,meseci_u_godini[i]);
                        strcpy(dani_u_godini[index++],temp);
                }

        if(!prestupna(god)) strcpy(dani_u_godini[DANI-1],"--");
                free(temp);
}

/*u bazi hotela za svaku sobu postoji broj, tip i niz datuma formiran
  prethodnom funkcijom*/
int baza_hotela (char *datoteka)
{
        FILE *fptr;
        Hotel soba;
        int i;

        printf("\nIme datoteke je %s\n",datoteka);
        fptr=fopen(datoteka,"wb");
        if(fptr==NULL)
                return 2;
        fflush(stdin);

        while(1)
        {
                printf("\nUnesite broj sobe (Enter za kraj): ");
                gets(soba.broj_sobe);
                if(strlen(soba.broj_sobe)==0)
                        break;

                printf("\nUnesite tip sobe (1,2,3,4,apt): ");
                gets(soba.tip_sobe);

                for(i=0;i<DANI;i++)
                        strcpy(soba.dani[i],dani_u_godini[i]);

                if(fwrite(&soba,sizeof(Hotel),1,fptr)!=1)
                        {
                                fclose(fptr);
                                return 3;
                        }
        }
        fclose(fptr);
        return 0;
}

int pregled_baze(char *datoteka)
{
        FILE *fptr;
        Hotel soba;
```

```
	fptr=fopen(datoteka,"rb");
	if(fptr==NULL)
		return 2;

	fflush(stdin);
	while(1)
	{
		if(fread(&soba,sizeof(Hotel),1,fptr)!=1 && !feof(fptr))
		{
			fclose(fptr);
			return 4;
		}
		if(feof(fptr))
			break;
		printf("\nBroj sobe: %s",soba.broj_sobe);
		printf("\nTip sobe (broj kreveta): %s\n",soba.tip_sobe);
		slobodni_termini(soba);
		if(rezervisano(soba)==0)
			printf("\nNema rezervisanih termina\n");
		getchar();
	}
	fclose(fptr);
	return 0;
}

/*soba je rezervisana odredjenog dana ako se umesto datuma u nizu nalazi
  ime osobe na koju glasi rezervacija*/
int slobodni_termini(Hotel soba)
{
	int ima_termina,nadjen,pocetak,kraj,i;

	ima_termina=nadjen=0;
	printf("\nSoba je slobodna u sledecim terminima:\n");

	for(i=0;i<DANI;i++)
	{
		if(isdigit(soba.dani[i][0]))
			{
				ima_termina++;
				nadjen=1;
				if(ima_termina==1)
					kraj=pocetak=i;
				else
					kraj=i;
			}
		else if(ima_termina!=0)
			{
				printf("\n%s  -  %s\n",soba.dani[pocetak],soba.dani[kraj]);
				ima_termina=0;
```

```
                        }
        }
        if(kraj==DANI-1)
                printf("\n%s  -  %s\n",soba.dani[pocetak],soba.dani[kraj]);

        if(!nadjen) printf("\nNema slobodnih termina\n");
                return nadjen;
}

int rezervacija_sobe(char *datoteka)
{
        FILE *fptr;
        Hotel soba;
        char osoba[MAX_IME],br_sobe[DUZINA];
        int mesto_u_bazi,nadjena,index_pocetka,index_kraja,slobodno,i;
        int dan_pocetka,dan_kraja,mesec_pocetka,mesec_kraja;

        mesto_u_bazi=nadjena=0;

        fptr=fopen(datoteka,"rb+");
        if(fptr==NULL)
                return 2;
        printf("\nUnesite broj sobe ");
        scanf("%s",br_sobe);
        fflush(stdin);

        /*proverava se da li u bazi postoji soba sa unetim brojem*/
        while(1)
        {
                if(fread(&soba,sizeof(Hotel),1,fptr)!=1 && !feof(fptr))
                        {
                                fclose(fptr);
                                return 4;
                        };
                if(feof(fptr))
                        break;
                mesto_u_bazi++;
                if(strcmp(soba.broj_sobe,br_sobe)==0)
                        {
                                nadjena=1;
                                break;
                        }
        }
        if(!nadjena)
        {
                printf("\nNe postoji soba sa tim brojem\n");
                fclose(fptr);
                return -1;
        }
```

```
/*daje se izvestaj da li je i u kojim terminima je slobodna soba*/
if(!slobodni_termini(soba))
{       fclose(fptr);
        return -1;
}

 /*unosi se datum pocetka i kraja rezervacije i proverava se da li su
   podaci pravilno uneti*/
printf("\nUnesite datum pocetka rezervacije (d-m) ");
scanf("%i-%i",&dan_pocetka,&mesec_pocetka);
index_pocetka=provera_datuma(dan_pocetka,mesec_pocetka);

if(index_pocetka==-1)
{
        printf("\nPogresno unet datum\n");
        fclose(fptr);
        return -1;
}

printf("\nUnesite datum kraja rezervacije (d-m) ");
scanf("%i-%i",&dan_kraja,&mesec_kraja);
index_kraja=provera_datuma(dan_kraja,mesec_kraja);

if(index_kraja==-1)
{
        printf("\nPogresno unet datum\n");
        fclose(fptr);
        return -1;
}
if(index_kraja<index_pocetka)
{       printf("\nPogresno uneti termini\n");
        fclose(fptr);
        return -1;
}

fflush(stdin);
slobodno=0;
for(i=index_pocetka;i<=index_kraja;i++)
        if(!isdigit(soba.dani[i][0]))
        {
         slobodno=1;
         break;
        }
if(slobodno)
{
        printf("\nSoba nije slobodna u tom terminu\n");
        fclose(fptr);
        return -1;
}
```

```
	/*umesto datuma upisuje se ime osobe na koju ce glasiti rezervacija*/
	printf("\nUnesite ime na koje ce biti rezervacija :");
	gets(osoba);
	for(i=index_pocetka;i<=index_kraja;i++)
		strcpy(soba.dani[i],osoba);
	if(fseek(fptr,(mesto_u_bazi-1)*sizeof(Hotel),SEEK_SET))
	{
		fclose(fptr);
		return 3;
	}
	if(fwrite(&soba,sizeof(Hotel),1,fptr)!=1)
	{
		fclose(fptr);
		return 3;
	}
	fclose (fptr);
	return 0;
}

int provera_datuma(int dan, int mesec)
{
	char *temp;
	int i;

	temp=malloc(MAX_IME);
	if(mesec>0 && mesec<=MESECI && dan>0 && dan<=31)
	{
		/*brojevi od 1 do 31 pretvaraju se u niz znakova*/
		itoa(dani_u_mesecu[dan-1], temp, 10);
		strcat(temp, ". ");
		strcat(temp,meseci_u_godini[mesec-1]);
		for(i=0;i<DANI;i++)
			if(strcmp(dani_u_godini[i],temp)==0)
				{
					free(temp);
					return i;
				}
	}
	free(temp);
	return -1;
}

/*trazi u datoteci za koju sobu se umesto datuma nalazi navedeno ime*/
int potvrda_rezervacije(char *datoteka)
{
	char osoba[MAX_IME];
	FILE *fptr;
	int nadjen_u_bazi,nadjen,i,index_pocetka,index_kraja;
	Hotel soba;
```

```
	fflush(stdin);
	fptr=fopen(datoteka,"rb");
	if(fptr==NULL)
		return 2;

	printf("\nUnesite ime osobe ");
	gets(osoba);

	nadjen_u_bazi=nadjen=0;
	printf("\nNa ime %s je rezervisano:\n",osoba);

	while(1)
	{
		if(fread(&soba,sizeof(Hotel),1,fptr)!=1 && !feof(fptr))
		{
			fclose(fptr);
			return 4;
		}
		if(feof(fptr))
			break;
		for(i=0;i<DANI;i++)
		{
			if(strcmp(soba.dani[i],osoba)==0)
			{
				nadjen++;
				nadjen_u_bazi=1;
				if(nadjen==1)
					index_pocetka=index_kraja=i;
				else
					index_kraja=i;
			}
			else
			{
				if(nadjen!=0)
				{
					printf("\nSoba %s : ",soba.broj_sobe);
					printf("%s - ",dani_u_godini[index_pocetka]);
					printf("%s\n",dani_u_godini[index_kraja]);
				}
				nadjen=0;
			}

		}
	}
	 if(nadjen_u_bazi==0)
		printf("\nNema rezervacije na to ime\n");

	 fclose(fptr);
	 return 0;
}
```

```
/*umesto imena ponovo se upisuje datum*/
int otkazivanje (char *datoteka)
{
        FILE *fptr;
        Hotel soba;
        char osoba[MAX_IME];
        int nadjen,mesto_u_bazi,i;

        nadjen=mesto_u_bazi=0;
        fflush(stdin);

        fptr=fopen(datoteka,"rb+");
        if(fptr==NULL)
                return 2;

        printf("\nNa koje ime se otkazuje rezervacija ");
        gets(osoba);

        while(1)
       {
                if(fread(&soba,sizeof(Hotel),1,fptr)!=1 && !feof(fptr))
                        {
                                fclose(fptr);
                                return 4;
                        }
                if(feof(fptr)) break;
                mesto_u_bazi++;
                for(i=0;i<366;i++)
                if(strcmp(soba.dani[i],osoba)==0)
                {
                        nadjen++;
                        strcpy(soba.dani[i],dani_u_godini[i]);
                }
                if(fseek(fptr,(mesto_u_bazi-1)*sizeof(Hotel),SEEK_SET))
                {
                        fclose(fptr);
                        return 3;
                }
                if(fwrite(&soba,sizeof(Hotel),1,fptr)!=1)
                {
                        fclose(fptr);
                        return 3;
                }
                if(fseek(fptr,mesto_u_bazi*sizeof(Hotel),SEEK_SET))
                {
                        fclose(fptr);
                        return 3;
                }
        }
```

```
	if(nadjen==0)
		printf("\nNema rezervacije na ime %s\n",osoba);
	else
		printf("\nOtkazano %i rezervacija\n",nadjen);

	fclose(fptr);
	return 0;
}

int rezervisano (Hotel soba)
{
	int nadjen=0,rezervisano=0,i;

	printf("\nSoba je rezervisana u sledecim terminima:\n");
	for(i=0;i<366;i++)
	{	if(isalpha(soba.dani[i][0]))
		{
			nadjen++;
			rezervisano=1;
			if(nadjen==1)
				printf("\n%s -",dani_u_godini[i]);
			else if(strcmp(soba.dani[i],soba.dani[i-1])!=0)
			{
			    printf(" %s na ime %s\n",dani_u_godini[i-1],soba.dani[i-1]);
			    nadjen=0;
			    i=i-1;
			}
		}

		else if(nadjen!=0)
		{
			printf(" %s na ime %s\n",dani_u_godini[i-1],soba.dani[i-1]);
			nadjen=0;
		}
	}

	return rezervisano;
}

void pisanje_poruka(int broj_greske)
{
	char *poruke[]={
		"Uspesno",
		"Nema dovoljno memorije",
		"Neuspeh prilikom otvaranja datoteke",
		"Neuspeh prilikom upisivanja podataka u datoteku",
		"Neuspeh prilikom citanja podataka"			};
	puts(poruke[broj_greske]);
}
```

Евиденција урађених лабораторијских вежби

Вежба	Датум	Вежбу прегледао
1.		
2.		
3.		
4.		
5.		
6.		
7.		
8.		
9.		
10.		

Име и презиме студента: _____

Број индекса и година уписа: _____

Назив студијског програма: _____

Прилог

Табела 1. Службене речи језика *C*

auto	double	int	struct
break	else	long	switch
case	enum	register	typedef
char	extern	return	union
const	float	short	unsigned
continue	for	signed	void
default	goto	sizeof	volatile
do	if	static	while

Поред ових речи, језик *C++* има још 32 службене речи, од којих су у примерима програма ове збирке заступљене: class, private, protected, public, new и delete.

Табела 2. Контролне секвенце стандардног излаза језика *C* и *C++*

Секвенца	Значење
\n	Прелаз у нови ред (NewLine NL)
\t	Хоризонтални табулар (HorizontalTabulation HT)
\v	Вертикални табулар (VerticalTabulation VT)
\b	Повратак за једно место (BackSpace BS)
\r	Повратак на почетак реда (CarriageReturn CR)
\f	Прелаз на следећу страну (FormFeed FF)
\a	Аларм/Звоно (Alarm/Bell BELL)
\'	Апостроф
\"	Наводник
\\	Коса црта уназад (BackSlash)

Табела 3. Основни типови података у језицима *C* и *C++*

Тип	Ознака	Величина (бајтова)	Опсег
Character	char	1	-128 do 127
Short integer	short	2	-32768 do 32767(1)
Integer	int	2/4	(1)/(2)
Long integer	long	4	-2,147,483,648 до 2,147,438,647(2)
Unsigned character	unsigned char	1	0 do 255
Unsigned short integer	unsigned short	2	0 do 65535 (3)
Unsigned integer	unsigned	2/4	(3)/(4)
Unsigned long integer	unsigned long	4	0 до 4,294,967,295 (4)
Single-precision Floating_point	float	4	1.2E-38 до 3.4E38 (прецизност=7 цифара)
Double-precision Floating-point	double	8	2.2E-308 до 1.8E308 (прецизност=16 цифара)

Табела 4. Контроле за форматирање излаза у језику *C*

Ознака	Тип податка за који се користи
%c	char
%d %u %o %x (X)	int (децимални облик) unsigned int (децимални облик) int/unsigned int (октални облик) int/unsigned int (хексадецимални облик)
%f %e %g	float, double (облик са децималном тачком) float, double (облик са експонентом) float, double (најједноставнији облик)

Табела 5. Контроле за форматирање улаза у језику *C*

Ознака	Тип податка за који се користи
%c	char
%d %u %o %x (X) %i	int (децимални облик) unsigned int (децимални облик) int/unsigned int (октални облик) int/unsigned int (хексадецимални облик) int (било који од три горе поменута облика)
%f %e %g	float (облик са децималном тачком или експонентом) float (облик са децималном тачком или експонентом) float (облик са децималном тачком или експонентом)
%lf %le %lg	double (облик са децималном тачком или експонентом) double (облик са децималном тачком или експонентом) double (облик са децималном тачком или експонентом)

Табела 6. Приоритети оператора у језику *C*

Приоритет	Број операнада	Оператор
15	2	[ ] ( ) . ->
14	1	! ~ ++ -- + - * & (тип) sizeof
13	2	* / %
12	2	+ -
11	2	<< >>
10	2	< <= > >=
9	2	== !=
8	2	&
7	2	^
6	2	|
5	2	&&
4	2	||
3	3	?:
2	2	= += -= *= /= %= &= ^= |= <<= >>=
1	2	,

Језик *C++* има додатне операторе у односу на операторе из ове табеле, али су од њих у примерима ове збирке заступљени следећи: оператор досега :: (приоритет 16), оператори доделе и ослобађања меморије new и delete (приоритет 14) и нова значења оператора >> и << (оператори улаза и излаза, приоритет 11).

Табела 7. Стандардна библиотека *stdio.h*

scanf("format",a1,a2…)	Функција чита са тастатуре податке у задатим *formatu* и уписује их од задатих адреса у меморији *(a1,a2...)*
printf("format",izraz1,izraz2…)	Функција приказује на екрану, у задатом *formatu,* вредности задатих израза (*izraz1*, *izraz2*...)
getchar(void)	Функција чита са тастатуре један знак и враћа његову *ASCII* вредност
putchar(c)	Функција приказује на екрану знак чију *ASCII* вредност добија у аргументу
gets(s)	Функција чита са тастатуре један низ знакова, до знака за прелаз у нови ред и уписује га у стринг *s*
puts(s)	Функција приказује на екрану садржај стринга *s*
fopen(ime, mod)	Функција отвара датотеку *ime* у режиму *mod* и враћа показивач на структуру типа *FILE* или показивач *NULL*
fclose(fptr)	Функција затвара датотеку којој је придружен показивач *fptr*
fscanf(fptr, "format",a1, a2…)	Функција чита са из датотеке којој је придружен показивач *fptr,* податке у задатим *formatu* и уписује их од задатих адреса у меморији *(a1,a2...)*
fprintf(fptr, "format, izraz…)	Функција уписује у задатом *formatu,* вредност израза *izraz* у датотеку којој је придружен показивач *fptr*
fgetc(fptr)	Функција чита један знак из датотеке којој је придружен показивач *fptr* и враћа његову *ASCII* вредност
fputc(c, fptr)	Функција уписује знак чију *ASCII* вредност добија у аргументу у датотеку којој је придружен показивач *fptr*
fgets(s, max+1, fptr)	Функција чита из датотеке којој је придружен показивач *fptr* један низ знакова, првих *max* или до знака за прелаз у нови ред и уписује га у стринг *s*
fputs(s, fptr)	Функција уписује садржај стринга *s* у датотеку којој је придружен показивач *fptr*
fread(a, vel, n, fptr)	Функција чита *n x vel* бајтова из бинарне датотеке (показивач је *fptr*) и уписује у меморију од адресе *a*
fwrite(a, vel, n, fptr)	Функција уписује *n x vel* бајтова у бинарну датотеку (показивач је *fptr*), са адресе *a* у меморији
ftell(fptr)	Функција враћа вредност текуће позиције у датотеци којој је придужен показивач *fptr*
fseek(fptr, n, poz)	Функција помера текућу позицију у датотеци којој је придужен показивач *fptr,* од позиције *poz* за *n* бајтова
rewind(fptr)	Функција враћа текућу позицију у датотеци којој је придужен показивач *fptr,*на почетак датотеке
fflush(fptr)	Функција брише заостале карактере у улазном току из датотеке којој је придужен показивач *fptr*
feof(fptr)	Вредност функције је ≠ 0, ако је нађен крај датотеке којој је придужен показивач *fptr*

Табела 8. Стандардна библиотека *ctype.h*
Свака даље поменута функција библиотеке *ctype.h* враћа вредност типа *int* различиту од 0 (*true*), ако аргумент *c* задовољава описани услов (аргумент *c* је типа је *int* и чува *ASCII* кôд знака који се испитује).

isalnum(c)	Вредност функције је различита од 0, ако је вредност *c* ASCII кôд слова или децималног броја.
isalpha(c)	Вредност функције је различита од 0, ако је вредност *c* ASCII кôд слова.
isdigit(c)	Вредност функције је различита од 0 ако је вредност *c* ASCII кôд децималног броја.
islower(c)	Вредност функције је различита од 0 ако је вредност *c* ASCII кôд малог слова.
isupper(c)	Вредност функције је различита од 0 ако је вредност *c* ASCII кôд великог слова.
tolower(c)	Ако је вредност *c* ASCII кôд великог слова, вредност функције је ASCII кôд истог малог слова.
toupper(c)	Ако је вредност *c* ASCII кôд малог слова, вредност функције је ASCII кôд истог великог слова.

Табела 9. Стандардна библиотека *math.h*
У свим даље поменутим функцијама библиотеке *math.h* вредности *x* и *y* су типа *double* и све функције враћају вредност типа *double.*

cos(x)	Вредност функције је *cos x*
fabs(x)	Вредност функције је |x|
log(x)	Вредност функције је $\log_e x$, x>0.
log10(x)	Вредност функције је $\log_{10} x$, x>0.
pow(x, y)	Вредност функције је x^y
sin(x)	Вредност функције је sin x.
sqrt(x)	Вредност функције је $\sqrt{x}$
tan(x)	Вредност функције је tg x.

Табела 10. Стандардна библиотека *stdlib.h*

Даље су поменуте само неке коришћене функције библиотеке *stdlib.h*.

atof(s)	Функција враћа вредност типа *double*, добијену конверзијом цифара из низа знакова *s.*
atoi(s)	Функција враћа вредност типа *int*, добијену конверзијом цифара из низа знакова *s.*
atol(s)	Функција враћа вредност типа *long*, добијену конверзијом цифара из низа знакова *s.*
rand()	Функција враћа псеудо-случајну вредност типа *int*, у опсегу бројева од 0 до *RAND_MAX* (32767).
exit(status)	Функција прекида извршавање програма и прослеђује *status* оперативном систему: 1 (била је грешка) или 0 (није било грешке).
malloc(n)	Функција тражи динамичку доделу простора у меморији за *n* бајтова. Вредност функције је адреса првог бајта додељене меморије (ако је успела додела) или *NULL* у супротном.
realloc(∗a,n)	Функција тражи промену величине динамички додељеног простора у меморији од адресе ***a*** на *n* бајтова. Вредност функције је адреса првог бајта додељене меморије после измене (ако је успела) или *NULL* у супротном.
free(∗a)	Функција тражи ослобађање динамички додељеног простора у меморији од адресе ***a***.

Табела 11. Стандардна библиотека *string.h*

У свим даље поменутим функцијама библиотеке *string.h*, *s, s1* и *s2* су низови карактера, а *c* и *n* су вредности типа int.

strcat(s1, s2)	Функција дописује садржај низа *s2* на крај постојећег садржаја низа *s1* и враћа показивач на резултујући низ *s1.*
strcmp(s1, s2)	Функција пореди садржаје низова *s1* и *s2* и враћа вредност типа *int:* <0, ако је низ *s1* мањи по *ASCII* табели од низа *s2,* 0, ако је низ *s1* исти по *ASCII* табели као и низ *s2,* >0, ако је низ *s1* већи по *ASCII* табели од низа *s2.*
strcpy(s1, s2)	Функција копира садржај низа *s2* од почетка низа *s1* (укључује и \0) и враћа показивач на резултујући низ *s1.*
strlen(s)	Функција враћа дужину низа *s* (не укључује \0).

Табела 12. Стандардна библиотека *fstream.h*

f.open(ime, mod)	Функција отвара датотеку *ime* у режиму *mod* и придружује јој објекат *f*
f.close()	Функција затвара датотеку којој је придружен објекат *f*
f.get()	Функција чита један знак из датотеке којој је придружен објекат *f* и враћа његову *ASCII* вредност
f.get(s, max+1)	Функција чита један низ знакова из датотеке којој је придружен објекат *f*, првих *max* или до знака за прелаз у нови ред и уписује га у стринг *s*
f.get(s, max+1, c)	Функција чита један низ знакова из датотеке којој је придружен објекат *f*, првих *max* или до знака *c* или до знака за прелаз у нови ред и уписује га у стринг *s*
f.getline(s, max+1)	Функција чита један низ знакова из датотеке којој је придружен објекат *f*, првих *max* или до знака за прелаз у нови ред, и уписује га у стринг *s* (брише знак '\n')
f.put(c)	Функција уписује у датотеку којој је придружен објекат *f* знак чију *ASCII* вредност добија у аргументу
f.read(a, n)	Функција чита *n* бајтова из бинарне датотеке којој је придружен објекат *f* и уписује у меморију од адресе *a*
f.write(a, n)	Функција уписује *n* бајтова у бинарну датотеку којој је придружен објекат *f,* са адресе *a* у меморији
f.tellg()	Функција враћа вредност текуће позиције у датотеци отвореној за читање, којој је придружен објекат *f*
f.tellp()	Функција враћа вредност текуће позиције у датотеци отвореној за упис, којој је придружен објекат *f*
f.seekg(n, poz)	Функција помера текућу позицију у датотеци отвореној за читање, којој је придружен објекат *f,* од позиције *poz* за *n* бајтова
f.seekp(n, poz)	Функција помера текућу позицију у датотеци отвореној за упис, којој је придружен објекат *f,* од позиције *poz* за *n* бајтова
f.flush()	Функција брише заостале карактере у улазном току из датотеке којој је придружен објекат *f*
f.eof()	Вредност функције је ≠ 0, ако је нађен крај датотеке којој је придружен објекат *f*

Табела 13. Основна табела *ASCII* кôдова

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	000	NUL (null)	32	20	040	Space	64	40	100	@	96	60	140	`
1	1	001	SOH (start of heading)	33	21	041	!	65	41	101	A	97	61	141	a
2	2	002	STX (start of text)	34	22	042	"	66	42	102	B	98	62	142	b
3	3	003	ETX (end of text)	35	23	043	#	67	43	103	C	99	63	143	c
4	4	004	EOT (end of transmission)	36	24	044	$	68	44	104	D	100	64	144	d
5	5	005	ENQ (enquiry)	37	25	045	%	69	45	105	E	101	65	145	e
6	6	006	ACK (acknowledge)	38	26	046	&	70	46	106	F	102	66	146	f
7	7	007	BEL (bell)	39	27	047	'	71	47	107	G	103	67	147	g
8	8	010	BS (backspace)	40	28	050	(	72	48	110	H	104	68	150	h
9	9	011	TAB (horizontal tab)	41	29	051	)	73	49	111	I	105	69	151	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	74	4A	112	J	106	6A	152	j
11	B	013	VT (vertical tab)	43	2B	053	+	75	4B	113	K	107	6B	153	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	76	4C	114	L	108	6C	154	l
13	D	015	CR (carriage return)	45	2D	055	-	77	4D	115	M	109	6D	155	m
14	E	016	SO (shift out)	46	2E	056	.	78	4E	116	N	110	6E	156	n
15	F	017	SI (shift in)	47	2F	057	/	79	4F	117	O	111	6F	157	o
16	10	020	DLE (data link escape)	48	30	060	0	80	50	120	P	112	70	160	p
17	11	021	DC1 (device control 1)	49	31	061	1	81	51	121	Q	113	71	161	q
18	12	022	DC2 (device control 2)	50	32	062	2	82	52	122	R	114	72	162	r
19	13	023	DC3 (device control 3)	51	33	063	3	83	53	123	S	115	73	163	s
20	14	024	DC4 (device control 4)	52	34	064	4	84	54	124	T	116	74	164	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	85	55	125	U	117	75	165	u
22	16	026	SYN (synchronous idle)	54	36	066	6	86	56	126	V	118	76	166	v
23	17	027	ETB (end of trans. block)	55	37	067	7	87	57	127	W	119	77	167	w
24	18	030	CAN (cancel)	56	38	070	8	88	58	130	X	120	78	170	x
25	19	031	EM (end of medium)	57	39	071	9	89	59	131	Y	121	79	171	y
26	1A	032	SUB (substitute)	58	3A	072	:	90	5A	132	Z	122	7A	172	z
27	1B	033	ESC (escape)	59	3B	073	;	91	5B	133	[	123	7B	173	{
28	1C	034	FS (file separator)	60	3C	074	<	92	5C	134	\	124	7C	174	|
29	1D	035	GS (group separator)	61	3D	075	=	93	5D	135	]	125	7D	175	}
30	1E	036	RS (record separator)	62	3E	076	>	94	5E	136	^	126	7E	176	~
31	1F	037	US (unit separator)	63	3F	077	?	95	5F	137	_	127	7F	177	DEL

Литература

[1] Ласло Краус, Програмски језик *C* са решеним задацима, Академска мисао, Београд 2006.

[2] Ласло Краус, Програмски језик *C*++ са решеним задацима, Академска мисао, Београд 2004.

[3] Слободан Обрадовић, Вештина доброг програмирања, Виша електротехничка школа, Београд 2004.

[4] Милан Чабаркапа, *C* основи програмирања, Круг, Београд 1996.

[5] *Brian W. Kernighan, Dennis M. Ritchie, The C Programming Language - ANSI C, Prentice Hall Software Series*, 1988.

[6] *Augie Hansen,* Програмирање на језику *C* – потпуни водич за програмски језик *C*, Микро књига, Београд 1991.

[7] *Peter Aitken, Bradley Jones,Teach Yourself C in 21 Days,CD edition*,1998.

[8] *Chris H. Pappas, William H. Murray*, *C/C*++ Водич за програмере, Микро књига, Београд 1996.