OBJEKTNO ORIJENTISANO PROGRAMIRANJE

Oznaka predmeta: OOP

Predavanje broj: 3

Nastavna jedinica: Koncepti jezika C++ koji nisu objektno orijentisani .

Nastavne teme:

Pretprocesor. Direktive. Makrodefinicije i makrozamene. Uslovno prevođenje Upotreba makrodefinicija Operatori # i ##. Leksička analiza. Tokeni. Komentari. Ključne reči. Literali. Deklaracije i definicije. Organizacija programa i povezivanje. Organizacija programa po fajlovima. Fajlovi: *.h *.cpp. Standardna zaglavlja. Struktura programskog fajla. Glavna funkcija (main) i završetak programa.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

Dragan Milićev, “Objektno orijentisano programiranje na jeziku C++”, Mikro knjiga, Beograd, 2005.

Pretprocesor

- Pretprocesor (*preprocessor* , makroprocesor) je program koji kao ulaz prihvata programski izvorni tekst, vrši određene transformacije tog teksta te kao izlaz daje ovaj transformisani tekst programa.
  - Transformacije nad izvornim tekstom zadaju se posebnim linijama u programu koje se nazivaju *direktivama* pretprocesoru.
  - Direktive utiču samo na pretprocesor.
  - Pretprocesor daje "prečišćen" tekst koji se prosleđuje prevodiocu.
- Pretprocesor je skoro uvek integrisan sa prevodiocem.
- Iako prevodioci jezika C++ vrlo često rade pretprocesiranje i prevođenje u istom prolazu kroz program, rezultat je potpuno isti kao da je pretprocesor radio potpuno nezavisno, prolaskom kroz program u celini pre prevođenja.
- Glavne mogućnosti pretprocesora su:
  - makrozamene,
  - uključivanje teksta drugih fajlova u tekst programa
  - uslovno prevođenje

Direktive

- Direktive su neka vrsta naredbi pretprocesoru.
- Direktiva je linija teksta izvornog programa koja počinje znakom #.
  - Ispred znaka # može se naći proizvoljan broj blanko znakova ili tabulatora.
    - Za rastavljanje direktive na više redova (ne može stati u jedan) koristi se znak obrnuta kosa crta (\) koji se stavlja na kraj reda.

  Primer:

```
#define makro ova direktiva se nastavlja i u  \
        sledeci red. \
        I ovaj red je deo direktive, a tu je i kraj.
```

- Direktive nemaju nikakve veze sa ostatkom jezika C++.
  - One se mogu naći bilo gde u programu. Pretprocesor jednostavno ne prepoznaje ništa od sintakse jezika C++, već samo običan tekst programa.
  - Važenje direktive počinje od mesta na kome se ona nalazi i traje sve do kraja programskog fajla.

Makrodefinicije i makrozamene

- Jedna od najvažnijih mogućnosti pretprocesora je izvršavanje tzv. *makrozamene* (*macroexpansion*).
  - Direktivom **#define** se pretprocesoru definiše jedan *makro* (*macro*). To je reč čije sve pojave u izvornom tekstu programa pretprocesor treba da zameni specificiranim nizom reči:

    ```
    #define Pi 3.14
    ```

  - U ovoj direktivi (makrodefiniciji) reč Pi je makro. Ovom direktivom se nalaže da pretprocesor u tekstu programa iza direktive izvrši zamenu (tzv. makrozamena ili makroekspanzija) svih pojava reči Pi znacima 3.14. Pretprocesor razlikuje mala i velika slova.
- Pošto se direktive mogu naći bilo gde u programu njihovo važenje počinje od mesta gde se nalaze do kraja fajla (osim ako se ne ukinu):

  ```
  a=b+Pi;
  #define Pi 3.14
  c=a+Pi;
  ```

  pretprocesor će dati rezultat (izlazni tekst):

  ```
  a=b+Pi;
  c=a+3.14;
  ```

Sintaksa makrodefinicije

- Makrodefinicija ima sledeću sintaksu:

```
#define identifikator niz_reči
```

- *Identifikator* je reč koja predstavlja makro, a niz reči iza ovog identifikatora predstavlja telo makrodefinicije - tekst kojim će svaka pojava makroa u tekstu programa iza direktive biti zamenjena.

```
#define MAKRO ovo je tekst kojim ce rec MAKRO biti zamenjena
```

- Sledeća direktiva:

```
#define MAX 100
```

uzrokuje da tekst: `int a[MAX];`
pretprocesor pretvori u tekst: `int a[100];`

- Pretprocesor "slepo" zamenjuje makro nizom reči. Tako na primer direktiva:

```
#define IZRAZ a+b;
```

uzrokuje da tekst: `c=IZRAZ+c;`
pretvari u tekst: `c=a+b;+c;`

što verovatno nije bio cilj (dvotačka u makrodefiniciji je višak).

Sintaksa makrodefinicije

- Makrodefinicijom se može zadati i makrozamena sa argumentima.
  - Odmah iza prvog identifikatora (makro-a), bez razmaka između, ide znak otvorena zagrada, pa lista argumenata i zatvorena zagrada.

```
#define zbir(a,b) (a+b)
```

Reč *zbir* je makro, dok su a i b formalni argumenti makrodefinicije. Svaka pojava u tekstu reči *zbir* iza koje sledi par zagrada sa dva stvarna argumenta biće zamenjen odgovarajućim tekstom:

Tekst: `a=zbir(c+d,2);`
postaje: `a=(c+d+2);`

- U ovom primeru prvi stvarni argument je tekst c+d, a drugi je tekst 2, pa se umesto parametra **a** iz makrodefinicije pojavljuje c+d, a umesto parametra **b** tekst 2.

Paziti na dodavanje zagrada u tekstu zamene, npr:

```
#define zbir(a,b) a+b
```

onda bi tekst: `b=a*zbir(c,d);`
bio pretvoren u tekst: `b=a*c+d; //nije ono sto smo hteli`

Sintaksa makrodefinicije

- U tekstu zamene se ponovo traže identifikatori definisanih makroa tako da se i oni zamenjuju, osim makroa koji je upravo zamenjen.

```
#define MAKRO svaka pojava reci MAKRO
#define BezVeze #define MAKRO kraj
```

tekst `BezVeze` posle pretprocesiranja postaje:

```
#define svaka pojava reci MAKRO kraj
```

sada bi završetkom pretprocesiranja prevodilac dojavio grešku jer ne prepoznaje #define

- Definicija makroa može da se ukine direktivom #undef.

```
a=Pi+b;
#define Pi 3.14
c=Pi+d;
#undef Pi
e=Pi+f;
```

ovaj tekst posle pretprocesiranja postaje:

```
a=Pi+b;
c=3.14+d;
e=Pi+f;
```

Direktiva #include

- Direktiva:

  ```
  #include <ime_fajla>
  ```

  rezultujućem tekstu umesto sebe uključuje sadržaj fajla označenim kao ime_fajla.

  ```
  #include <iostream>
  ```

  prethodna direktiva nalaže pretprocesoru da u fajl u kome se direktiva nalazi, na njeno mesto, uključi ceo tekst iz fajla sa imenom iostream.
- Naznačeni fajl traži se na nizu mesta koja su specificirana implementacijom pretprocesora (najčešće neki direktorijum u kome su smešteni fajlovi samog prevodioca predviđeni za uključivanje (standardni direktorijum).
- Drugi oblik direktive #include je : `#include "ime_fajla"` radi kao i prethodna direktiva, ali se fajl traži najpre na nekom posebno definisanom mestu (tekući direktorijum), a ako se tu ne nađe, pretraživanje se nastavlja na istom onom mestu koje se pretražuje za direktivu prvog oblika.
- Direktive #include se mogu ugnežđavati. To znači da se u tekstu fajla koji se uključuje može naći nova direktiva #include.

Uslovno prevođenje

- Direktivama je moguće neki deo teksta programa proslediti prevodiocu pod određenim ispunjenim uslovom u trenutku pretprocesiranja.

```
#if konstantni_izraz
        if_deo
#elif konstantni_izraz
        elif_deo
#else
        else_deo
#endif
```

  - Iza direktive *#if* (važi i za *#elif*) sledi proizvoljni izraz koji se može izračunati u vreme prevođenja (ne obuhvata programske promenljive) i koji rezultuje celobrojnom konstantnom vrednošću.
  - Delova *#elif* može biti proizvoljno mnogo, a mogu se i izostaviti. Deo *#else* se može takođe izostaviti.
- Ako je vrednost izraza iza *#if* različita od 0, konačni tekst je *if_deo,* u suprotnom izračunava se sledeći izraz iza *#elif* i ako je njegova vrednost različita od nule, konačni tekst je *elif_deo.* Ako su vrednosti svih izraza nule, konačni tekst je *else_deo*.

Uslovno prevođenje

- U izrazima uslovnog prevođenja može se naći i operator *defined*. Njegova upotreba je:

```
defined identifikator
```

ili

```
defined (identifikator)
```

Vrednost ovakvog izraza je 1 ako je *identifikator* prethodno definisan kao makro direktivom #define, a nije ukinut direktivom #undef; inače je vrednost nula.

```
#define _DEBUG                  //obicno pocinju sa _NAZIV
...
#if defined _DEBUG              // moze i #ifdef _DEBUG
        stampaj();              // nešto štampa
#endif
```

- Direktiva:

```
#ifndef identifikator ili  #if !defined identifikator
```

znači suprotno: uslov je ispunjen ako *identifikator* nije definisan (*not defined*).

Upotreba makrodefinicija

- Pretprocesor jezika C++ vodi svoje poreklo iz jezika C gde su direktive za definisanje makroa korišćene su za uvođenje konstanti u program.
- Tada je postavljanje nove vrednosti konstante rađeno promenom jedne makrodefinicije. Identifikator konstante je pisan velikim slovima da bi se razlikovale od promenljivih pri čitanju programa.

```
#define MIN 2
#define MAX 99
...
if (i<MIN) ...
if (i>MAX) ...
```

- U jeziku C++ prethodnu upotrebu sasavim je potisnulo postojanje konstantnih objekata.
- Makroi sa argumentima su pozivani pozivom nalik na funkcije, ali bez suvišnih režija za poziv "prave" funkcije.

```
#define max(a,b) a>b?a:b
...
x=max(y,z);
```

Za ovu se u jeziku C++ koriste funkcije ugrađe u kôd (*inline functions*).

Operator

- U pretprocesoru, kao i u celom jeziku C++, niz znakova uokviren znacima navoda (") smatra se leksičkim elementom - tokenom (*token*) čiji se sadržaj u pretprocesoru ne analizira (ne vrši se zamena makroa unutar niza znakova uokvirenog navodnicima).
- Ako je potrebno da rezultat pretprocesiranja bude baš niz znakova koji predstavlja stvarni argument makrozamene uokviren znacima navoda, koristi se operator #.

```
#define makro(argument) #argument
#define Pera Detlic
```

Ako se makro pozove sa:

```
makro(Pera)    makro("Pera")
```

rezultat će biti:

```
Pera           "Pera"
```

- Rezultat se dalje ne analizira na zamenu, pa se Pera ne zamenjuje rečju Detlic.
- Paziti, ako je: `#define makro(argument) "argument"`
  poziv: makro(Pera) daje: `"argument"`

Operatori

- Operator ## u tekstu zamene makroa između dve reči znači spajanje tih reči (konkatenaciju).
  - Najpre se te dve reči u tekstu eventualno zamenjuju stvarnim argumentima ako predstavljaju formalne argumente, pa se rezultat spaja u jednu reč.
    - Samo tako spojena reč se dalje analizira kao makro za eventualnu zamenu.

  Primer:

```
#define promenljiva(i) Prom##i

promenljiva(1)=promenljiva(2)+promenljiva(3);
```

  rezultat je tekst: `Prom1=Prom2+Prom3;`

- Iz svega što je rečeno sledi da je pretprocesor jezika C++ orijentisan na tokene, a ne na znakove.
    - Pretprocesor najpre rastavlja tekst na tokene, pa tek onda tokene zamenjuje ako su definisani kao makroi.

Ostale direktive

- Direktiva:

  ```
  #line konstanta "ime_fajla"
  ```

  u predefinisanom (ugrađenom) makrou koji se naziva `__LINE__` dodeljuje se navedena konstanta.
  - Prevodilac će smatrati da je redni broj sledeće linije programa jednak toj konstanti (pomoć programeru u porukama o greškama u programu).
  - Sličan smisao ima i *ime_fajla*. Ovaj naziv (može i da se izostavi) dodeljuje se predefinisanom makrou `__FILE__`.
- Direktiva kojom prevodilac daje poruku o grešci sa navedenim tekstom

  ```
  #error tekst
  ```
- Direktiva: `#pragma tekst`
  ima značenje koje je definisano implementacijom prevodioca. Za uključivanje fajla samo jednom u program koristili smo direktivu:

  ```
  #pragma once
  ```
- Prazna direktiva: `#` nema nikakvog efekta.

Predefinisana imena

- U pretprocesor su uvek ugrađeni sledeći tzv. predefinisani makroi:

  `__LINE__` sadrži niz znakova koji predstavlja tekuću liniju programa;

  `__FILE__` sadrži niz znakova koji predstavlja naziv programskog fajla koji se trenutno prevodi;

  `__DATE__` sadrži niz znakova koji predstavlja datum prevođenja u obliku "*mm dd gggg*";

  `__TIME__` sadrži niz znakova koji predstavlja vreme prevođenja u obliku "*čč*:*mm*:*ss*";

  `__cplusplus` je definisan ako se prevodi C++ program (a ne C program).

- Ova imena se
  ne mogu ukinuti sa `#undef`
  niti redefinisati sa `#define`.

Leksička analiza

- Na početku analize programa, prevodilac rastavlja tekst programskog fajla na elementarne gramatičke jedinice - *tokene* (*token*).
- Posle ovog rastavljanja, pretprocesor transformiše izvorni tekst programa:
  - vrši makrozamene, uključivanje fajlova, uslovno prevođenje.
- Rezultat pretprocesiranja je tekst - niz tokena. Tokeni su elementi pomoću kojih prevodilac kasnije vrši sintaksnu analizu programa i generiše kôd.
- Faza rastavljanja teksta programa na tokene naziva se *leksičkom analizom*.

- ***Tokeni***
- Jezik C++ poseduje pet grupa tokena: ključne reči, separatore, identifikatore, operatore i literale. (`if x>5 y+=3.14;`)
- Blanko znaci, horizontalni i vertikalni tabulatori, znaci prelaska u novi red i na novu stranicu nazivaju se zajedničkim imenom *belinama* (*white space*). Beline su separatori tokena.
- Prelazak u novi red i blanko znak rastavljaju tokene.

Tokeni

- Na primer, deo programa:

```
if (a<b) if (c<d) for
                  (i=0; i<100
        ; i++)
        niz[i]=niz[i+1];
```

je identičan (za prevodioca) sledećem delu:

```
if (a<b)
  if (c<d)
    for (i=0; i<100; i++)
      niz[i]=niz[i+1];
```

- Na prevodiocu je zadatak da tekst rastavi na tokene.
- Prevodilac izdvaja redom jedan po jedan token iz teksta programa pri čemu uzima najduži niz znakova koji predstavljaju token.
  Kod: `a=b+++c;`
  znači isto što i: `a=b++ +c;`
  jer se iza znaka b, koji predstavlja token, uzima niz od dva znaka + koji mogu predstavljati token (operator ++).

Komentari

- Komentar je sve iza // do kraja reda:

```
a+b*c;   // komentar
```

- Komentara na proizvoljnom mestu u redu počinje znacima /* i završava znacima */

```
a=b+ /* Ovo je komentar: ova linija znači a=b+c; */ c;
/* i ovo je komentar */
```

- Znaci //, /* i */ nemaju nikakvo značenje unutar komentara koji počinju sa //, već pripadaju komentaru.
- Takođe, znaci // i /* nemaju nikakvo značenje unutar komentara koji počinje sa /*.
- Komentari se smatraju belinama.
- Izdvajanje tokena, pa i komentara kao belina, vrši se pre pretprocesiranja (pretprocesor je orijentisan na tokene - njegov ulaz je tekst rastavljen na tokene, a ne niz znakova).
- Komentari unutar direktiva koje definišu makroe se smatraju belinama, a ne delom teksta koji ulazi u telo makrodefinicije.

```
#define MAKRO ovo je definicija // a ovo nije
```

Identifikatori

- Identifikator je proizvoljno dug niz slova i cifara. Za prvi znak identifikatora uzmite slovo engleske abecede.
- C++ razlikuje velika i mala slova pa bi trebalo paziti.

- Setite se da se i znak donja crta (_) smatra slovom, pa se može naći i na početku identifikatora.
  - Identifikatori koji počinju jednim znakom donja crta najčešće se koriste u posebne svrhe, u bibliotekama koje se daju uz prevodioce.
  - Identifikatori koji počinju sa dva znaka donja crta (__)su rezervisani za upotrebu u implementacijama C++ prevodilaca, pa se izričito **ne** preporučuju za običnu upotrebu.

- Postoje, u osnovi, dva stila pisanja identifikatora koji se sastoje od više reči iz govornog jezika:

```
BankovniRacun
bankovni_racun
```

Identifikatori

- Ključne reči jezika C++ su rezervisane (ne mogu se upotrebljavati sa dugim značenjem osim onog propisanog u jeziku). Evo nekih:

asm	continue	float	new	signed	try
auto	default	for	operator	sizeof	typedef
break	delete	friend	private	static	union
case	do	goto	protected	struct	unsigned
catch	double	if	public	switch	virtual
char	else	inline	register	template	void
class	enum	int	return	this	volatile
const	extern	long	short	throw	while

- Znaci koji se u C++ programu koriste kao operatori ili znaci interpunkcije:

```
!  %  ^  &  *  (  )  -  +  =  {  }  |  ~  [  ]  \  ;  '  :  "
<  >  ?  ,  .  /
```

- Kombinacije znakova se tretiraju kao tokeni (to su operatori jezika C++):

```
->  ++  --  .*  ->*  <<  >>  <=  >=  ==  !=  &&
||  *=  /=  %=  +=  -=  <<=  >>=  &=  ^=  |=  ::
```

Literali

- Literali su tokeni koji u programu predstavljaju konstantne veličine nekog tipa ("konstante") i u jeziku C++ postoji četiri vrste literala:

1. Znakovne konstante (*character constants*),
2. Celobrojne konstante (*integer constants*), su nizovi cifara
   - Ako ne počinje nulom – decimalni broj, `16 //16`$_{10}$
   - Ako počinje sa 0 – oktalni broj (u bazi 8, znaci 0..7) `020 //16`$_{8}$
   - Ako počinje sa 0x ili 0X – heksadecimalni broj (baza 16, znakovi 0..9,A…F ili 0..9,a..f) `0x10//16`$_{16}$
3. Racionalne konstante (*floating constants*)
4. String-literali (*string literals*).

- Tip decimalne celobrojne konstante je neki od celobrojnih tipova: int (ceo broj), long int ("dugi" ceo broj, veći opseg brojeva), unsigned long int (neoznačeni "dugi" ceo broj)…
- Za oktalne i heksadecimalne konstante redosled izbora prevodioca je: int, unsigned int (neoznačeni ceo broj), long int, unsigned long int, long long, unsigned long long.

Literali

- Ako se niz cifara završava znakom l ili L, tip literala je prvi od long tipova kojim se može predstaviti dati broj.

```
120034L        // "dugi" celobrojni broj
0x1afU         // nepredznačeni heksadecimalni broj
```

- Za long long koristi se LL (ll) i za znak U (u).
- Znakovne konstante predstavljaju se pomoću jednog ili više znakova uokvirenih znacima ' (aspostrof), npr. 'a' ili 'abc'.
  - Znakovni literali sa jednim znakom su tipa char, a sa više znakova su tipa wchar_t.
  - Vrednost znakovne konstante sa jednim znakom jednaka je numeričkoj vrednosti znaka u skupu znakova date mašine,
  - Vrednost znakovne konstante sa više znakova zavisi od implementacije prevodioca.
- Znakovne konstante sa više znakova se retko koriste i treba ih izbegavati.

Tabela posebnih znakova

Opis znaka	Znak	Predstava	Primer
Novi red (*new line*)	NL(LF)	\n	'\n'
Horizontalni tabulator (*horizontal tab*)	HT	\t	'\t'
Vertikalni tabulator (*vertical tab*)	VT	\v	'\v'
Backspace	BS	\b	'\b'
Carriage return	CR	\r	'\r'
Nova strana (*form feed*)	FF	\f	'\f'
Zvono (*alert*)	BEL	\a	'\a'
Obrnuta kosa crta (*backslash*)	\	\\	'\\'
Znak pitanja (*question mark*)	?	\?	'\?'
Apostrof (*single quote*)	'	\'	'\''
Znaci navoda (*double quotes*)	"	\"	'\"'
Oktalni broj		\ooo	'\142'
Heksadecimalni broj		\xhhh	'\x3F0'

Literali

- Racionalne konstante predstavljaju se na način uobičajen u programskim jezicima.
  - Racionalna konstanta ima celobrojni deo, decimalnu tačku, razlomljeni deo, i eventualnu specifikaciju eksponenta koja počinje znakom e ili E, iza koga sledi opcioni znak eksponenta (+ ili -) i njegova celobrojna vrednost.
  - Svi brojevi su sa osnovom deset. Celobrojni ili razlomljeni deo (ali ne oba) se može izostaviti (podrazumeva se 0).
  - Tip racionalne konstante je double, osim ako se konstanta završava sufiksom f ili F, kada je tip float, ili sufiksom l ili L, kada je tip long double

Primer:

```
1.2       // 1.2 tipa double
.2        // 0.2
2.        // 2.0
.4e-3     // 0.4E-3
2.E4f     // 2.0E4 tipa float
```

Literali

- String literali se predstavljaju nizom znakova (objekata tipa char) uokvirenim znacima navoda unutar kojih se mogu se naći pokazani posebni znaci (iza \).
- Za svaki string literal prevodilac rezerviše statički prostor u memoriji koji inicijalizuje datim znacima (zavisi od implementacije).
  Na primer, za svaki od stringova "abc" i "c" prevodilac može odvojiti poseban prostor tako da se oni ne preklapaju, ali može odvojiti i samo prostor za prvi string, dok će drugi string biti podskup prvog.
- Stringovi koji su napisani neposredno jedan iza drugog se spajaju u jedan, ali tako da se znaci iz spojenih stringova održavaju odvojenim.

```
"\xA" "B"  // string koji sadrži dva znaka: '\xA' i 'B',
           // a ne samo jedan znak '\xAB‘
```

- Na kraju spajanja susednih stringova prevodilac dodaje znak '\0' za detekciju kraja stringa.

```
"abc"          // niz znakova dužine 4
```

- Znak " unutar string literala mora se navesti iza znaka \.

```
"\"Porsche\""  // ovo je string "Porsche"
```

Deklaracije i definicije

- Deklaracijom se u program uvodi jedno ili više imena i saopštava se prevodiocu:
  - Šta se sa imenom može raditi,
  - Kog je tipa: objekat koga ime predstavlja ili funkcija koju ime predstavlja,
  - Kako izgleda klasa koju ime predstavlja i slično.
- Ako deklaracija objekta sadrži kreiranje objekta, onda je to uvek i definicija.
- Ako deklaracija funkcije sadrži telo funkcije, onda je to definicija.
- U programu mora postojati jedna i samo jedna definicija svakog objekta, funkcije, klase i nabrajanja (*enumeration*).
- Deklaracije koje nisu definicije se mogu proizvoljno mnogo puta ponavljati u programu, dok definicije ne mogu.

```
class S;    // deklaracija koja samo govori da je S ime klase;
class S;    // može se ponavljati;
int a;      // definicija;
int a;      // greška: definicija se ne sme ponavljati;
```

Deklaracije i definicije

- U prethodnom primeru treba primetiti način deklaracije klase S.
- Ovom ("praznom") deklaracijom se prevodiocu samo saopštava da je S ime klase, ali to nije dovoljno za kreiranje objekta te klase:
  - da bi prevodilac znao kako da kreira i rukuje objektom klase S, mora znati deklaraciju te klase.
- Ovakva ("prazna") deklaracija se koristi za definisanje objekata tipa "pokazivač na tip S“, jer za formiranje objekta ovakvog tipa nije potrebno znati izgled klase S:

```
class S;
class P {
 public:
   S* ps;
};
```

- Ako se ovakvom pokazivaču želi dodeliti neka vrednost, tj. da bi on ukazivao na neki objekat, treba kreirati objekat klase S.
  - Za to je već potrebna deklaracija klase S.

Organizacija programa i povezivanje

- Program u jeziku C++ se sastoji iz više programskih modula koji se odvojeno prevode u tzv. objektni kôd (*object code*) a onda povezuju (*linking*) u konačni, izvršni program (*executable code*).
- Svaki programski modul čuva se u jednom fajlu.
- Prevodilac prevodi svaki programski fajl (skoro) *potpuno nezavisno*, ne znajući koji sve fajlovi čine program.
- Zbog potpuno nezavisnog prevođenja jednog fajla (prevodilac "ne vidi" ništa drugo osim onoga što se nalazi u fajlu koji trenutno prevodi)
- Svaki programski fajl treba da sadrži:

  odgovarajuće deklaracije svih imena koje se u fajlu koriste (ako se koristi objekat koji se kreira u drugom fajlu, onda je potrebna samo deklaracija tog objekta).
- Jedan objekat se može formirati samo na jednom mestu u programu, a koristiti na više mesta (pravilo jedinstvene definicije objekta), uz proizvoljno mnogo njegovih deklaracija.
- Deklaracije imena navode se u svim fajlovima u kojima se ta imena koriste.

Organizacija programa po fajlovima

- Kada prevodilac prevede fajl, u formiranom objektnom kodu su informacije o svim imenima koja su definisana u tom fajlu i namenjena za korišćenje u drugim fajlovima, kao i o imenima koja su korišćena u tom fajlu, ali nisu tu i definisana.
- Kada "pregleda" sve ovakve informacije iz svih objektnih fajlova koji čine program, povezivač (*linker*) ima zadatak da "razmeni" sve prikupljene informacije o korišćenju imena i da fajlove poveže u jedinstven izvršni program.
- Da bi program bio pregledan i lakši za održavanje:
  - organizacija programa je takva da se u odvojene fajlove smeštaju definicije različitih klasa.
  - definicije logički povezanih klasa se smeštaju u isti fajl (klasa sadrži član koji je objekat druge klase, a ta druga klasa se ne koristi na drugim mestima).
- Prethodno: u fajlu u kome se nalazi deklaracija klase S (class S;) klasa S se koristi samo za deklarisanje objekata tipa "pokazivač na S", pa nije potrebna cela deklaracija ove klase.

Fajlovi-zaglavlja

- U C++ programu čest je slučaj da se deklaracije imena nalaze na više mesta u programu, tj. u više programskih fajlova.
- Ako se deklaracija nekog imena (npr. klase) nalazi na više mesta u programu, onda i najmanja izmena deklaracije zahteva pažljivu analizu svih programskih fajlova, kako bi svi oni koji sadrže promenjenu deklaraciju bili ažurirani.
- Rešenje: sve potrebne deklaracije za korišćenje imena iz neke logičke celine programa, npr. deklaracije svih klasa i drugih imena definisanih u jednom programskom fajlu, a namenjenih za korišćenje u drugim fajlovima, grupišu u poseban fajl koji se naziva *zaglavljem* (*header file*).
- U fajlovima gde je potrebno koristiti imena čije se deklaracije nalaze u fajlu-zaglavlju navodi se samo direktiva pretprocesora #include kojom se fajl-zaglavlje uključuje u odgovarajući programski fajl.
- Ovim se u svim fajlovima gde se imena koriste nalaze sve potrebne deklaracije, ali se izmene mogu vršiti samo na jednom mestu.
- Fajlovi zaglavlja uobičajeno imaju ekstenziju .h, dok programski fajlovi uobičajeno imaju ekstenziju .cpp.

Fajlovi: *.h *.cpp

- Da bi se izbeglo da se zaglavlje koje može imati i definicije uključi više puta koriste se sledeće tehnike:
  - Stara tehnika:

```
// Fajl: osoba.h
// Sadrži deklaraciju klase osoba
#ifndef _Osoba
#define _Osoba
  class Osoba {
    protected:
       char* ime;
       int god;
    public:
       virtual void KoSi();
       Osoba(char* ime_i_prezime, int godine_starosti);
  };
#endif
```

  - Nova tehnika: direktiva prevodiocu **`#pragma once`**
- Ovim se može u fajlovima ponavljati direktiva za uključivanje fajla (`#include "Osoba.h"`) pri čemu ostaje i dalje samo jedna deklaracija.

Standardna zaglavlja

- Jezik C++ sve složenije operacije, kao što su ulaz/izlaz, matematičke funkcije, operacije nad nizovima znakova, pozive sistemskih usluga i slično prepušta bibliotekama funkcija i klasa.
- Deklaracije funkcija, klasa i drugih imena iz pojedinih biblioteka nalaze se u standardnim fajlovima-zaglavljima koji se nalaze u posebnom katalogu prevodioca. Uključuju se sa #include <ime>
- Neka zaglavlja koja se često koriste su:

<cfloat>	parametri za funkcije za rad sa racionalnim brojevima u pokretnom zarezu
<climits>	parametri okruženja, informacije o ograničenjima prevodioca, kao i opsege celobrojnih veličina
<cstddef>	nekoliko često korišćenih tipova podataka i makroa
<cstdarg>	makroi za čitanje argumenata funkcija koje mogu da prime promenljiv skup argumenata
<cstdlib>	deklaracije raznih često korišćenih funkcija za konverzije, sortiranje i slično
<iostream>	deklaracije standardnih ulazno/izlaznih funkcija i klasa
<cstring>	deklaracije funkcija za rad sa nizovima znakova
<cmath>	deklaracija matematičkih funkcija.

Glavna funkcija (main)

- Kako se svi programski fajlovi sastoje isključivo od deklaracija, sav izvršni kôd programa nalazi se raspoređen po definicijama funkcija.
- Funkcija koja se prvo poziva kada program počne sa izvršavanjem je funkcija nečlanica sa nazivom *main*.
  - Ovu funkciju definiše programer.
  - Svaki program mora imati funkciju koja se zove *main*.
- Tip funkcije main zavisi od implementacije prevodioca (svi prevodioci dozvoljavaju da tip povratne vrednosti ove funkcije bude void ili int).
- Ako je tip int, funkcija treba da vrati celobrojnu vrednost koja će biti prosleđena operativnom sistemu pri završetku rada programa.
- Najčešće se vraćena vrednost 0 tretira kao regularan završetak programa.
- Prevodioci dozvoljavaju da funkcija *main* ne prima nikakve argumente, ili da prima argumente po sledećoj deklaraciji:

```
int main (int argc, char* argv[]) {/*...*/}
```

Glavna funkcija (main)

- Vrednosti argumenata funkcije *main* postavlja operativni sistem pri pokretanju programa.
- Prvi argument *argc* sadrži broj argumenata sa kojima je program pozvan. Drugi argument *argv* predstavlja niz pokazivača na znakove gde će biti smešteni kao što sledi:
  - Element argv[0] sadrži ime kojim je program pozvan, ili prazan niz znakova (""), zavisno od prevodioca,
  - Ostali elementi sadrže reči koje su navedene iza imena programa pri pozivu.
  - Poslednji argument je u argv[argc-1].
  - Element argv[argc]=0.

  Primer, ako je program prozvan iz operativnog sistema sa:

  `primer argument1 argument2`

  onda će arc = 3 (3 reči je u pozivu), argv[0] = "**`primer`**", argv[1] ="**`argument1`**", argv[2] = "**`argument2`**“ , argv[3]=0.
- Funkcija main() se ne sme pozivati u programu, niti se sme uzimati njena adresa.

Završetak programa

- Program se može završiti naredbom return; u funkciji main() ako je povratni tip void ili uz povratnu vrednost ako je main() deklarisana da vraća tip int.
- Isti efekat može se postići pozivom funkcije

```
void exit(int status);  //EXIT_SUCCESS,EXIT_FAILURE
```

koja je deklarisana u standardnom zaglavlju `<cstdlib>`.

  - Argument ove funkcije biće prosleđen operativnom sistemu kao povratna vrednost pri završetku programa.
- Funkcija **`atexit`**, koja je deklarisana u istom zaglavlju, može se koristiti da se specificiraju korisničke funkcije koje će biti pozvane kada se program završava pozivom funkcije `exit`.

```
int atexit ( void (*function) (void) );
```

- Poziv funkcije

```
void abort();
```

završava program bez poziva korisničkih funkcija koje su specificirane pomoću *atexit*.

Zadatak

```
//FAJL: Tacka2D.h ----------------------------------------
#pragma once
class Tacka2D{
   private:
      double x;
      double y;
   public:
      Tacka2D();
      void PostaviXY(double novix, double noviy);
      double UzmiX();
      double UzmiY();
      double UdaljenostDoTacke(Tacka2D drugatacka);
      void IspisiPodatke();
};
//FAJL: Tacka2D.cpp ----------------------------------------
#include <iostream>
#include <cmath>
#include "Tacka2D.h"
Tacka2D::Tacka2D(){
   x = y = 0;
}
```

Zadatak

```
//nastavak fajla Tacka2D.cpp ---------------------------------
void Tacka2D::PostaviXY(double novix, double noviy)
{
   x = novix;
   y = noviy;
}

double Tacka2D::UzmiX() { return x; }
double Tacka2D::UzmiY() { return y; }

double Tacka2D::UdaljenostDoTacke(Tacka2D drugatacka)
{
   return sqrt( pow( drugatacka.UzmiX()-x ,2) +
                pow( drugatacka.y      -y ,2)    );
}
void Tacka2D::IspisiPodatke()
{
   using namespace std;
   cout<< "("<< x <<","<< y <<")";
}
```

Zadatak

```
//FAJL U KOME JE FUNKCIJA main ------------------------------
#include <iostream>
using namespace std;
#include "Tacka2D.h"
#define _ISPISI_PODATKE
int main(){
   cout<<"\"Primer mix\""<<endl;
   Tacka2D A;          // da bude A=(Ax,Ay)=(0.0,0.0) t.j. (0,0)
   Tacka2D B;          // da bude B=(Bx,By)=(0.0,0.0) t.j. (0.0)
   B.PostaviXY(3.0,4.0);
#if defined _ISPISI_PODATKE
   cout<<"A";
   A.IspisiPodatke();
   cout<<endl;;
   cout<< "B("<< B.UzmiX() <<","<< B.UzmiY() <<")"<<endl;
#endif
   cout<<"Udaljenost od A do B iznosi "
       << A.UdaljenostDoTacke(B) << endl;
   cout<<"Linija koda je "<<__LINE__<<endl;
```

Zadatak

```
    int a=100, b=10;
    cout<<"a="<<a<<", b="<<b<<endl;
    cout<<"a+++b="<<a+++b<<endl;
    cout<<"a="<<a<<"\n\n\n";
    system("pause");
    return 0;
}
//_____________ I Z L A Z   I Z   P R O G R A M A : _________
"Primer mix"
A(0,0)
B(3,4)
Udaljenost od A do B iznosi 5
Linija koda je 18 //bice napisano koja je to kod Vas linija koda
a=100, b=10
a+++b=110
a=101


Press any key to continue . . .
```