OBJEKTNO PROGRAMIRANJE 1

Oznaka predmeta: OOP

Predavanje broj: 7

Nastavna jedinica: Operatori i izrazi.

Nastavne teme:

Aritmetički binarni operatori. Aritmetički unarni operatori. Prefiksni i postfiksni inkrement i dekrement. Operatori na nivou bita: operatori pomeranja. Logički operatori na nivou bita. Logički operatori. Relacioni operatori i operatori (ne)jednakosti. Operatori dodele. Operatori referenciranja i dereferenciranja. Operator uzimanja adrese &. Operatori za rad sa dinamičkom memorijom (new, delete). Operator indeksiranja []. Operator poziva funkcije (). Operator uslova ?:. Operator sekvence ,. Operator veličine sizeof. Operator eksplicitne konverzije (). Operator razrešavanja opsega važenja ::. Izrazi. Standardne konverzije.

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

Literatura:

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

Operatori i izrazi

- **Aritmetički operatori**
- U aritmetičke operatore svrstani su operatori koji realizuju binarne aritmetičke operacije:
    - sabiranja,
    - oduzimanja,
    - množenja,
    - deljenja
    - pomeranja,
    - unarne aritmetičke operacije:
      promene znaka, inkrementiranja i dekrementiranja.
- Operatori sabiranja (+) i oduzimanja (-) su binarni operatori koji grupišu sleva udesno.
- Operandi moraju biti aritmetičkog tipa ili tipa pokazivača.
- Za operande aritmetičkih tipova primenjuju se standardne aritmetičke konverzije.
- Rezultat ovih operacija nije lvrednost.

Aritmetički binarni operatori

- Operatori množenja (*), deljenja (/) i ostatka (%) su binarni operatori koji grupišu sleva udesno.
    - operandi operacija * i / moraju biti aritmetičkog tipa,
    - operandi operacije % moraju biti celobrojnog tipa.
- Standardne konverzije se primenjuju za operande ovih operacija i određuju tip rezultata. Rezultat nije lvrednost.
- Rezultat operatora * je proizvod operanada. Rezultat operacije / je količnik pri deljenju operanada, a rezultat operatora % je celobrojni ostatak pri deljenju operanada.
    - Ako je drugi operand operacija / i % jednak 0, rezultat nije definisan;
- Tip rezultata određen je tipovima operanada, prema specifikacijama standardnih konverzija. To je naročito važno za operator / ako su operandi celobrojni, i rezultat je celobrojan, što znači da se radi o operaciji celobrojnog deljenja.

```
int a=13, b=4;
int c=a/b;        // c dobija vrednost 3, celobrojno deljenje
c=c*b+a%b;        // c dobija vrednost 13
float d=13.0/b;   // d dobija vrednost 3.25
```

Aritmetički unarni operatori

- Aritmetički unarni operatori su operatori znaka:
  - operator +
  - operator -
  - Ovi operatori zahtevaju jedan operand koji se navodi iza operatora (prefiksni operator).
- Operator + zahteva operand aritmetičkog tipa ili tipa pokazivača. Rezultat ima vrednost operanda.
- Operator - zahteva operand aritmetičkog tipa a rezultat je negativna vrednost operanda.
  - Ako je operand neoznačen (unsigned), rezultat se računa oduzimanjem operanda od 2^n, gde je n broj bita u binarnoj predstavi tipa rezultata.
- Tip rezultata je tip koji se dobija primenom standardne konverzije. Rezultat nije lvrednost.

```
a=-b;
b=+c-(-a);
```

Prefiksni i postfiksni inkrement i dekrement

- Prefiksni operator inkrementiranja je unarni operator koji se upotrebljava ispred operanda.
    - Operand mora biti aritmetičkog tipa ili tipa pokazivača.
    - Operand mora biti promenljiva lvrednost.
  - Operand se uvećava za 1, a rezultat ima novu vrednost operanda. Rezultat jeste lvrednost. Izraz ++x je ekvivalentan izrazu x+=1.
- Analogno važi za prefiksni operator dekrementiranja (umanjuje operand za 1).

```
int x = 0;
int y = ++x;     // x postaje 1, kao i y;
++x = --y;       // ++x je lvrednost; x postaje 0, kao i y
```

- Postfiksni operator inkrementiranja je unarni operator koji se upotrebljava iza operanda. Operand mora biti aritmetičkog tipa ili tipa pokazivača. Operand mora biti promenljiva lvrednost.
  - Rezultat ima vrednost operanda, a posle izračunavanja rezultata, operand se uvećava za 1. Tip rezultata je isti kao i tip operanda. Rezultat nije lvrednost.
- Analogno važi za postfiksni operator dekrementiranja (umanjuje operand za 1).

```
for (;*q++ = *p++;);//kopira niz(kraj je 0) od lokcije na koju gleda p
```

- Inkrementiranje i dekrementiranje nije definisano za tipove nabrajanja.

Operatori na nivou bita: operatori pomeranja

- Operatori pomeranja bita su binarni operatori koji se označavaju sa << i >>. Vrednost operacije E1<<E2 odnosno E1>>E2 ,jednaka je vrednosti E1, interpretirane kao binarna reč, pomerene za E2 bita ulevo odnosno udesno.
- Ovi operatori grupišu sleva udesno. Operandi moraju biti celobrojnog tipa, a nad operandima se vrši celobrojna promocija . Tip rezultata je tip promovisanog levog operanda. Rezultat nije lvrednost. Rezultat nije definisan ako je desni operand negativan, ili ako je veći ili jednak broju bita promovisanog levog operanda.
- Pri pomeranju ulevo (<<) ispražnjeni bitovi zdesna popunjavaju se nulama. Pomeranje udesno (>>) je garantovano logičko (ispražnjeni bitovi sleva se popunjavaju nulama) ako je E1 neoznačenog tipa (unsigned), ili ako je pozitivan. U suprotnom je rezultat zavisan od implementacije prevodioca.
- Ovi operatori ne menjaju operande (samo daju rezultat):

```
int a = 7, b = 8, c = 3;
int d = a<<c, e = b>>c; // d ima vrednost 56, e ima vrednost 1,
                        // a, b i c ostaju nepromenjeni
```

- Množenje sa 2^n : `a=b<<3;  // n=3, isto kao a=b*2*2*2;`
- Deljenje sa 2^n važi za deljenje neoznačenih brojeva (značenje pomeranja udesno zavisno od implementacije prevodioca za označene brojeve).

Logički operatori na nivou bita

- Na nivou bita: za logičko I operator je &, za logičko ILI operator je |, za logičko ekskluzivno-ILI operator je ^, i za logičku negaciju operator je ~.
- Operator ~ je prefiksni unarni operator. Operand mora biti celobrojnog tipa. Rezultat je tipa promovisanog operanda i nije lvrednost. Rezultat predstavlja prvi komplement operanda (svaki bit operanda se komplementira).
- Operatori &, | i ^ su binarni operatori koji zahtevaju celobrojne operande, nad kojima se primenjuju standardne konverzije.
- Sledeća if naredba ispituje da li je bit 7 promenljive status jednak 1:

```
if (status & 0x80) //...
```

- Sledeći kod postavlja dva bita najmanje težine na nulu:

```
unsigned flag=0x07;
unsigned ones=~0U;     // promenljiva ones ima predstavu sve jedinice
flag=flag & (ones<<2); // postavljena su dva bita najmanje težine na
                       // nulu
```

- Sa navedenim operacijama uvek treba biti pažljiv, ukoliko se želi prenosivost programa.
    - treba konsultovati specifikacije jezika C++ i ne treba se osloniti na testiranje programa na jednoj mašini.

Logički operatori

- Logički operatori su operator logičke I operacije (&&), operator logičke ILI operacije (||) i operator logičke negacije (!).
  - Operandi pripadaju celobrojnom tipu ili tipu pokazivača.
- Ovi operatori su binarni i grupišu sleva udesno. Rezultat je tipa int i nije lvrednost. Operacija && daje rezultat 1 ako su oba operanda različita od nule, inače daje rezultat 0. Operacija || daje rezultat 1 ako je bar jedan operand različit od nule, inače daje rezultat 0.
- Ako je vrednost prvog operanda operacije || različit od nule, drugi operand se i ne izračunava, jer je rezultat sigurno 1 (analogno važi i za &&)
  - Kod operatora logičkih operacija nad bitima (&, | i ^) ne garantuje se redosled izračunavanja operanada i uvek se izračunavaju oba operanda.

```
if (1 || f())    //funkcija f neće biti pozvana
```

- Operator logičke negacije je unarni prefiksni operator. Zahteva operand celobrojnog tipa ili tipa pokazivača. Rezultat je tipa int i nije lvrednost. Rezultat je jednak 1 ako je operand jednak nuli, u suprotnom jednak je 0.
  - Preporuka: ređe su greške ako se ispituje pozitivan uslov.

Relacioni operatori i operatori (ne)jednakosti

- Relacioni operatori su operatori za upoređivanje vrednosti operanada: operator manje (<), operator veće (>), operator manje ili jednako (<=) i operator veće ili jednako (>=). Operator upoređivanja na jednakost je operator ==, a na nejednakost operator !=.
- Relacioni operatori su binarni operatori koji grupišu sleva udesno. Operandi moraju biti aritmetičkog tipa ili tipa pokazivača. Rezultat je tipa int i nije lvrednost. Rezultat je jednak 1 ako je odgovarajuća relacija između operanada ispunjena, inače je rezultat 0.
- Na operande se primenjuju standardne konverzije. Operand tipa pokazivača može se upoređivati sa izrazom čija je vrednost 0 i sa pokazivačem istog tipa.
  - Pokazivač na bilo koji tip može se upoređivati sa pokazivačem na tip void, s tim što se najpre konvertuje u tip void*.
  - Veći je onaj pokazivač koji ukazuje na element sa većim indeksom u nizu.
- Ako pokazivači ne pokazuju na elemente istog niza, rezultat relacione operacije nije definisan, što ne znači da je 0. Obratiti pažnju na sledeći kod:

```
int a[100];                int b[100];
int *pa=&a[50];            int *pb=&b[99];
if (pa>=pb) cout<<"pa>=pb"; else cout<<"pa<pb"; cout<<endl;
cout<<"pa:"<<hex<<pa<<endl; cout<<"pb:"<<hex<<pb<<endl;
```

Operatori jednakosti/nejednakosti

- Navedeni operatori grupišu sleva udesno, ali se ovi operatori vrlo retko navode jedan iza drugog. Naime, izraz a<b<c znači (a<b)<c, a ne (a<b)&&(a<c).
- Operator upoređivanja na jednakost označava se sa ==, a operator upoređivanja na nejednakost sa !=.
    - Rezultat operacije == je 1 ako su operandi jednaki, inače je 0.
    - Za operator != važi suprotno.
  - Sve napomene o tipovima operanada, rezultata i ostalim specifičnostima, navedene u prethodnom odeljku o relacionim operatorima, važe i za ove operatore. Ovi operatori jedino imaju niži prioritet od relacionih operatora.
- Primer, funkcija strcmp u narednom primeru upoređuje dva niza znakova koji su dostavljeni kao argumenti, i vraća 1 ako su oni sasvim jednaki (imaju isti broj znakova i svi su znaci podudarni), inače vraća 0.

```
int strcmp (const char *p, const char *q) {
  for (;*p && *q;)
    if (*p++ != *q++) return 0;     // razliciti su
  if (*p || *q) return 0;           // ako i p i q gledaju na 0, dalje
  else          return 1;           // jednaki su
}
```

Operatori dodele

- Operatori dodele su sledeći: =, *=, /=, %=, +=, -=, >>=, <<=, &=, ^=, |=.
- Operator = deluje tako što objekat na koji se odnosi levi operand dobija vrednost desnog operanda, a kao rezultat operacije vraća se ta dodeljena vrednost.
- Dejstvo ostalih operatora, u izrazu čiji je opšti oblik E1 *op*= E2, je ekvivalentno izrazu E1 = E1 *op* E2, osim što se E1 izračunava samo jedanput.

```
flag&=(ones<<2);  // isto kao: flag=flag & (ones<<2);
ones<<=2;         // isto kao: ones=ones<<2;
```

- U ovom primeru treba primetiti da u prvom redu operator << ne menja vrednost operanda ones, dok se u drugom redu ovaj operand menja pomeranjem bita.
- Svi ovi operatori su binarni i grupišu zdesna ulevo, tako da je:

```
a=b=c;   // isto kao: a=(b=c), pa i a i b dobijaju vrednost c
```

- Svi ovi operatori zahtevaju promenljivu lvrednost kao levi operand.
    - Tip rezultata jednak je tipu levog operanda.
    - Rezultat jeste lvrednost, koja referencira isti objekat koji referencira levi operand, tako da je moguće pisati:

```
(a=b)=c; // a dobija vrednost najpre b, pa onda c; b se ne menja
```

Operatori dodele

- U operaciji proste dodele (=), ako su operandi aritmetičkih tipova, desni operand se konvertuje u tip levog operanda pre dodele.
- Kako ne postoji implicitna konverzija u tip nabrajanja, ako je levi operand tipa nabrajanja, i desni operand mora biti istog tipa.
- Ako je levi operand tipa pokazivača, i desni operand mora biti tipa pokazivača, ili konstanti izraz koji ima vrednost 0;
  - Pre dodele se desni operand konvertuje u tip levog operanda.
  - Objekti tipa const T i volatile T mogu biti dodeljeni objektima tipa T ili volatile T.
- Pošto se levi operand, ako predstavlja izraz, izračunava samo jednom, izraz koji je napisan korišćenjem složenog operatora dodele (npr. E1+=E2) efikasniji je od izraza koji koristi običan operator dodele i neku operaciju (E1=E1+E2), osim ako prevodilac nije sposoban da u drugom slučaju prepozna postojanje istog operanda sa leve i desne strane znaka dodele.
- U slučajevima kada je E1 izraz koji daje drugu vrednost pri ponovnom izračunavanju, odnosno ima sporedni efekat, semantika ova dva izraza je različita.

```
++x += 2; // paziti, nije isto sto i: ++x = ++x + 2;
```

Operatori referenciranja i dereferenciranja

- Operator dereferenciranja pokazivača (*) je unarni prefiksni operator. On predstavlja indirekciju: operand mora uvek biti izraz koji daje rezultat tipa pokazivača, a rezultat ove operacije je objekat na koji taj pokazivač ukazuje.
  - Ako je operand tipa "pokazivač na T", rezultat je tipa T.
  - Rezultat jeste lvrednost, koja referencira objekat na koji ukazuje operand.
- Operator pristupa članu (.) npr. *izraz.ime*, desni operand *ime* predstavlja ime nekog člana objekta klase koji je predstavljen levim operandom *izraz*.
  - Levi operand mora biti objekat klase koji ima člana imenovanog desnim operandom. Rezultat je imenovani član objekta i jeste lvrednost samo ako je imenovani član lvrednost.
- Objekat dereferenciranja pokazivača i pristupa članu (posredni pristup članu, (->) koristi se na sledeći način:

  izraz->ime, desni operand *ime* predstavlja ime nekog člana objekta klase.
  - Na ovaj objekat ukazuje pokazivač predstavljen levim operandom *izraz*.
  - Levi operand mora biti pokazivač na objekat klase koji ima člana imenovanog desnim operandom. Rezultat je imenovani član objekta i jeste lvrednost ako je imenovani član lvrednost.
  - Izraz E1->E2 je potpuno ekvivalentan izrazu (*E1).E2

Operator uzimanja adrese &

- Operator uzimanja adrese (&) je unarni prefiksni operator. Rezultat primene ovog operatora na neki objekat je pokazivač koji ukazuje na taj objekat.
  - Operand može biti funkcija ili lvrednost.
  - Ako je operand tipa T, rezultat je tipa T*. Ako je operand tipa const T, rezultat je tipa const T*. Slično važi i za volatile. Rezultat je pokazivač na operand i nije lvrednost.

  Funkcija swap zamenjuje vrednosti objekata na koje ukazuju njeni argumenti:

```
void swap (int *x, int *y) {
  int temp=*x;
  *x=*y;
  *y=temp;
}
void main () {
  int a=1, b=2;
  swap(&a,&b);      // prenose se adrese a i b
}
```

- Dodela adrese početka niza i adrese funkcije odgovarajućim pokazivačima:

```
int a[10]; int *pi=&a;       // &a je tipa int* i ima vrednost &a[0]
void f();
void (*pf)()=&f;             // &f je tipa void(*)() i ukazuje na f
```

Operatori za rad sa dinamičkom memorijom

- Životni vek dinamičkog objekta nije ograničen oblašću važenja, niti je na bilo koji drugi način definisan pre izvršavanja programa.
  - Životni vek ovakvih objekata kontroliše jedino programer (kreiraju se kada tok programa u vreme izvršavanja dođe do mesta na kome se zahteva njihovo kreiranje a ukidaju kada isti naiđe na mesto na kome se zahteva njihovo uništavanje).
  - Smeštaju se u posebnu oblast, u dinamičku memoriju.
  - Dinamički objekti su bezimeni i pristupa im se preko pokazivača.
- Dinamički objekat kreira se operatorom new, a ukida operatorom delete.
- Operator new je prefiksni operator (new *tip*). Kreira se dinamički objekat tipa *tip* a rezultat vraća pokazivač koji ukazuje na taj objekat.

```
int *p=new int;    // kreira se dinamički objekat tipa int,
*p=3;              // na koga ukazuje p; ovde objekat postaje 3,
(*p)++;            // a ovde 4;
int i=*p;          // promenljiva i dobija vrednost 4
```

- Operacija new:
  - Odvaja prostor u dinamičkoj memoriji za smeštanje objekta datog tipa,
  - Inicijalizuje taj objekat (pozivom konstruktora za objekat klase),
  - Vraća pokazivač na kreirani objekat.

Operator new

- Objekat koji se kreira operatorom new može se inicijalizovati pri kreiranju navođenjem inicijalizatora unutar zagrada iza imena tipa:

```
int *pi=new int(3); // kreira se dinamički objekat tipa int
                    // i inicijalizuje na vrednost 3;
Osoba *po=new Osoba("Petar Petrovic",40);
```

- Operator new kreira objekat koji nema svoje ime i vraća pokazivač na kreirani objekat tako da je jedini način da se objektu pristupi preko ovog pokazivača.
- Primer "gubljenja" dinamičkog objekta (na njega ništa ne referencira):

```
int i=*new int(3);   // dinamički objekat je izgubljen!
int j= new int(4);   // greška: new vraća pokazivač!
```

U prvom redu prevodilac ne prijavljuje grešku, ali najčešće izdaje upozorenje.

  - Operator new kreira dinamički, bezimeni objekat i vraća pokazivač na njega. Rezultat operacije * primenjene na ovaj pokazivač je objekat na koji on ukazuje.
  - Promenljiva ***i*** inicijalizuje se na vrednost ovog objekta, ali predstavlja sasvim drugi objekat, potpuno nezavisan od onog bezimenog.
  - Kako pokazivač na dinamički objekat nije nigde zapamćen, ovom objektu se više ne može pristupiti.
  - On se ne može čak ni uništiti, pa ostaje "zauvek" da živi.

Operator new

- Navedeni tip iza operatora new mora biti tip objekta, ali ne funkcije. Pokazivači na funkcije su objekti koji se ne mogu kreirati na ovaj način.
- Za kreiranje dinamičkog objekta nekog izvedenog tipa (osim pokazivača na objekte i nizove) naziv tipa mora navesti unutar zagrada.

```
int i=0;
int **ppi=new int*(&i); // dinamički objekat je tipa int*, može ovako
int f();
int (**ppf)()=new ( int(*)() ) (&f);
            // dinamički objekat tipa int(*)(), tip mora u zagrade
```

- Dinamički nizovi se mogu kreirati operatorom new, tako što se kao tip navodi odgovarajući tip niza, uz specificiranje dimenzije niza. Kao rezultat operatora new vraća se pokazivač na prvi element dinamičkog niza:

```
int *pi=new int[10];              // new int[...] vraća tip int*
for(int i=0; i<10; i++) pi[i]=i;// i-ti element se inicijalizuje na i
```

- Ako se kreira dinamički višedimenzioni niz, svi izrazi koji specificiraju dimenzije, osim prve, moraju biti konstanti izrazi sa pozitivnom vrednošću.
  - Izraz koji specificira prvu dimenziju može biti proizvoljni izraz.
  - Ovaj izraz može imati i vrednost 0 što nema smisla (new vraća pokazivač na objekat, a uzastopni ovakvi pozivi vraćaju različite pokazivače).

Operator new

- Objekat se u dinamičkoj memoriji inicijalizuje kao što sledi:
  - Za objekat klase poziva se konstruktor za taj objekat.
  - Argumenti poziva tog konstruktora navode se u inicijalizatoru iza tipa.
  - Za konstruktor bez argumenata, inicijalizator nije potreban.
  - Ako se inicijalizator ne navede za objekat ugrađenog tipa, objekat ima nedefinisanu početnu vrednost.
- Kada se operatorom new kreira niz objekata, ne može se navesti inicijalizator.
  - Ako je u pitanju niz objekata klase, onda ta klasa mora biti ili bez konstruktora, ili sa konstruktorom koji se može pozvati bez argumenata.
  - Tada se konstruktor poziva redom za jedan po jedan element niza.
- Izvršavanje operacije new poziva funkciju (ugrađena u jezik) za odvajanje (alokaciju) potrebnog prostora u dinamičkoj memoriji.
  - Ova funkcija ima naziv *operator new*.
  - Prvi argument ove funkcije sadrži veličinu tipa objekta koji se kreira, kako bi se znalo koliki je prostor u memoriji potreban.
  - Ostali argumenti su opcioni i služe za dostavljanje dodatnih informacija o načinu smeštanja objekta. Značenje ovih argumenata zavisi od implementacije ove funkcije.

Operator new

```
class X{ public:   void* operator new(size_t size, int A1, int A2){ /*
    ... */ }};
int main (){  X* ptr = new(1,2) X;}
```

- Za dinamičke objekte ugrađenih tipova poziva se navedena ugrađena funkcija operator new.
- Ova funkcija se za korisničke tipove (klase) može predefinisati (preklopiti).
- Kada se kreira dinamički objekat korisničkog tipa, pozvaće se ova predefinisana funkcija, ako je ima; ako je nema, poziva se ugrađena (globalna) funkcija.
- Ugrađena funkcija se može ekplicitno pozvati iako je definisana preklopljena funkcija za korisnički tip, navođenjem operatora `::` ispred operatora new.
- Ako funkcija operator new ne nađe potreban prostor u dinamičkoj memoriji, vratiće vrednost 0.
- Inicijalizacija dinamičkog objekta vrši se samo ako je ova funkcija vratila nenultu vrednost (pokazivač). Inače je rezultat celog izraza new jednak 0.
- Međusoban redosled poziva funkcije operator new za alokaciju prostora i izračunavanja argumenata inicijalizatora nije definisan.
- Takođe nije definisano da li se argumenti inicijalizatora uopšte izračunavaju ako funkcija operator new vrati vrednost 0.

Operator delete

- Operator delete je prefiksni unarni operator, koristi se kao: delete *izraz*.
  - Navedeni *izraz* mora davati rezultat koji predstavlja pokazivač na dinamički objekat koji je kreiran operatorom new.
  - Ova operacija ukida (uništava) dinamički objekat na koji ukazuje operand.

```
int *pi= new int(3);
Osoba *po= new Osoba("Petar Petrovic",40);
delete pi;     // ukida se objekat na koji ukazuje pi
delete po;     // ukida se objekat na koji ukazuje po
```

- Rezultat operacije delete je tipa void.
  - Operand mora biti pokazivač na objekat koji je kreiran operatorom new.
  - Primena operatora delete na pokazivač koji ima vrednost 0 je dozvoljena i nema nikakvog efekta.
  - Objekat ukinut ovom operacijom prestaje da živi (nedefinisan je efekat pokušaja pristupa dinamičkom objektu preko njegovog pokazivača posle ukidanja tog objekta).
- Dinamički objekti se obavezno moraju ukinuti kada više nisu potrebni (inače dolazi do curenja memorije), a posle toga ne treba više dereferencirati pokazivač koji je na njega ukazivao.

Operator delete

- Pre oslobađanja dinamičke memorije koju je zauzimao objekat poziva se destruktor, ako se radi o objektu klase koja ima destruktor.
- Dinamički nizovi ukidaju se operatorom delete na sledeći način: delete [] *izraz*.
  - *izraz* mora imati rezultat pokazivač na niz koji je alociran operatorom new.
  - ako je u pitanju niz objekata klase koja ima destruktor, poziva se taj destruktor za jedan po jedan element niza.
- Efekat ukidanja dinamičkog niza operatorom delete bez navođenja srednjih zagrada [] nije definisan.
  - Programer je dužan da pazi da se dinamički nizovi ukidaju baš ovakvom notacijom. Isto važi i za ukidanje običnih objekata notacijom delete[].
- Pokazivač na konstanti objekat ne može biti argument operatora delete:

```
void g(const int *p) {  // obećanje da se *p neće menjati,
  delete p;             // pa zato ovo nije dozvoljeno!
}
void f() {
  int *p=new int(3);
  g(p);
}
```

Razlike u odnosu na jezik C

- Jezik C ima dinamičke objekte i dinamičku memoriju, ali nema new i delete.
- Za alociranje dinamičke memorije u biblioteci <cstdlib>postoji više funkcija.
  - malloc, funkcija koja uzima za argument veličinu memorijskog prostora koji treba alocirati, a kao rezultat vraća pokazivač na alocirani segment memorije, tipa void*.
  - calloc, funkcija koja služi za alociranje prostora za nizove, ima dva argumenta, prvi je broj elemenata niza, a drugi je veličina jednog elementa.
    - kao rezultat vraća pokazivač tipa void*.
- Za oslobađanje dinamičke memorije postoji funkcija free, koja uzima argument tipa void* (pokazivač na alocirani prostor).
- Navedene funkcije ne treba koristiti jer operatori new i delete obezbeđuju:
  - veličina potrebnog prostora automatski se određuje iz tipa operanda operatora new;
  - ovaj operator vraća pokazivač na tip operanda, a ne void*, pa je kontrola tipova obezbeđena;
  - operator new automatski poziva konstruktor, a delete destruktor objekta; ovi operatori se mogu preklopiti.
- NE kombinovati operatore new i delete sa funkcijama tipa malloc i free.

Operator indeksiranja []

- Operator indeksiranja [] je binarni operator.
- Značenje ovog operatora je indeksiranje niza.
- Jedan (bilo koji) operand mora biti tipa pokazivača na tip T, a drugi mora biti celobrojnog tipa.
  - Rezultat je tipa T.
  - Izraz E1[E2] je, po definiciji, identičan izrazu *((E1)+(E2)).
- Operator [] je komutativan, pa je **a[i]** isto kao **i[a]**.
- Celobrojni operand može biti i negativan, pod uslovom da rezultujući pokazivač ukazuje na element niza:

```
char *s="Zdravo!";
s+=6;
char c=s[-4];       // c ima vrednost *(s-4), odnosno 'r'
```

Operator poziva funkcije ()

- Operator poziva funkcije () zahteva operand ispred zagrada tipa "funkcija koja vraća tip T", "pokazivač na funkciju koja vraća tip T", ili "referenca na funkciju koja vraća tip T“.
    - Rezultat operacije poziva funkcije u tom slučaju je tipa T.
    - Rezultat operacije poziva funkcije je lvrednost samo ako funkcija vraća referencu.
- Između zagrada se navodi lista izraza razdvojenih zarezom. Ovi izrazi predstavljaju stvarne argumente.
  - lista stvarnih argumenata može biti i prazna.
- Formalni argumenti funkcije koja se poziva inicijalizuju se u trenutku poziva odgovarajućim stvarnim argumentima.
- Redosled izračunavanja stvarnih argumenata je nedefinisan.
- Svi sporedni efekti koje uključuju izrazi koji definišu stvarne argumente, završavaju se pre ulaska u telo funkcije.
- Pozivi funkcija se mogu proizvoljno ugnježđavati, što znači da su rekurzije dozvoljene.

Operator uslovne dodele ?:

- Često je u programima potrebno da se nekoj promenljivoj dodeli vrednost koja zavisi od nekog uslova.

  Primer, posmatrajmo funkciju fact koja izračunava faktorijel prirodnog broja koji je dostavljen kao argument, rekurzivno:

```
int fact (int n) {
  if (0<n) return n*fact(n-1);
  else return 1;
}
```

- U ovom primeru, funkcija vraća vrednost jednog od dva izraza iza naredbe return, u zavisnosti od uslova `(0<n)` .

```
int fact(int n) { return (0<n) ? (n*fact(n-1)) : 1; }//kompaktan kod
```

- Ovaj operator uslova **?:** zahteva tri operanda i koristi se na sledeći način:
  izraz **?** *drugi_izraz* **:** *treci_izraz ;*
  - prvi operand predstavlja izraz koji se tumači kao uslov.
    - ako je vrednost ovog izraza različita od nule ("tačno"), vrednost cele operacije jednaka je vrednosti drugog izraza.
    - ako je vrednost prvog izraza jednaka nuli ("netačno"), vrednost cele operacije jednaka je vrednosti trećeg izraza.

Operator sekvence ,

```
for (; *p && *q;) {
   p++; q++;
}
```

Pošto je operacija sekvence izraz, prethodna petlja se može napisati kompaktnije ovako:

```
for (; *p && *q; p++,q++) { }
```

- Operator sekvence (,) je binarni operator koji se upotrebljava kao: *izraz,izraz*. Pri izračunavanju ove operacije najpre se izračunava prvi operand, a potom drugi. Rezultat operacije je vrednost drugog izraza.

```
int a(1),b(2),c(3), z = (a++,b++,c++); cout<<z<<endl;
```

- Vrednost operacije je vrednost drugog izraza. Svi sporedni efekti uključeni u izračunavanje prvog operanda završavaju se pre izračunavanja drugog operanda. Tip rezultata je tip drugog operanda. Rezultat je lvrednost ako je drugi operand lvrednost.
- Ovaj operator grupiše sleva udesno, tako da je a,b,c isto što i (a,b),c.
- U kontekstu gde zarez ima posebno značenje, na primer u listi stvarnih argumenata, izraz sekvence mora se navesti između zagrada. Na primer, funkcija f je ovde pozvana sa tri argumenta, koji imaju vrednosti 1, 2 i 3:

```
f(1,(t=1,t+1),3);
```

Operator veličine sizeof

- Operator sizeof je unarni operator koji se upotrebljava za dobijanje veličine objekta nekog tipa u bajtovima, u smislu njegovog zauzeća memorije:
  - sizeof *izraz* , operand je izaz koji se ne izračunava, već se samo izračunava veličina objekta tipa operanda
  - sizeof (*tip*), između zagrada je ime tipa, a izračunava se veličina objekta tog tipa.
- Rezultat ove operacije je konstanta celobrojnog neoznačenog tipa (size_t u standardnom zaglavlju <cstddef>.) koji zavisi od implementacije prevodioca.
- Kao prvi argument poziva funkcije operator new prenosi se veličina tipa T koji je naveden iza operatora new. To znači da je stvarni argument izraz sizeof(T).
- Operand operacije sizeof ne može biti funkcija (ali može biti pokazivač na funkciju), bit-polje, nedefinisana klasa, tip void, niti niz nepoznatih dimenzija.
- Kada se primeni na klasu, ovaj operator daje veličinu objekta ove klase, uključujući i potrebne dopune članova praznim prostorom, koja je potrebna kada se objekti ove klase smeštaju u niz, radi poravnanja elemenata niza na memorijske reči (*allignement*).

```
int a[10];                  // kod nizova je to memorijski prostor
const int i = sizeof(a);    // koji zauzima niz, i je 10*sizeof(int)
```

Operator eksplicitne konverzije ()

- Eksplicitna konverzija () ostala je od C-a (ne bi je trebalo koristiti) može se zahtevati u dva oblika:
  - *tip*(*lista_izraza*)
  - (tip)*izraz*.
- Primer na ugrađenim tipovima:

```
double poeni;
int ocena=int(poeni+0.5)/10;
```

- U ovom primeru, izraz *poeni+0.5*, koji je tipa double, konvertuje se u tip int eksplicitnom konverzijom.
- Ova konverzija u konkretnom slučaju nije bila neophodna, jer postoji standardna konverzija koja bi objekat ocena inicijalizovala celobrojnom vrednošću.
- Eksplicitna *C* konverzija drugog oblika pogodna je za tipove koji nemaju prost naziv (identifikator).

```
int i=0;
const int *pi1=&i;
int *pi2=(int*)pi1;      // bez ove konverzije ne bi moglo
```

- Rezultat konverzije nije lvrednost osim ako je konverzija u referencu.
- Pokazivač se može konvertovati u celobrojni tip koji je dovoljno velik da primi njegovu vrednost.

Standardne konverzije

```
int a=8;
a*=.5;
```

- U operaciji množenja i dodele, oba operanda se najpre dovode na zajednički tip.
- Tako se vrednost a konvertuje najpre u tip double. Zatim se ova vrednost množi sa 0.5 i dobija se rezultat tipa double. Ovaj rezultat se konvertuje nazad u tip int da bi se dodelio promenljivoj a.

- U sledećem primeru promenljiva b će dobiti vrednost 0.0:

```
float b=8.0f;
b*=1/2;
```

  - Kako su oba operanda operacije deljenja tipa int (celobrojni literali), i rezultat ove operacije biće broj 0 tipa int. Tek u operaciji množenja i dodele ova vrednost se konvertuje u tip float.

- Racionalno deljenje konstanti može se zahtevati na jedan od sledećih načina:

```
float b=8.0f;
b*=1/2.0;
b*=1.0/2;
b*=1.0/2.0;
```

C++ operatori konverzije

- U C++u se koriste sledeći operatori promene tipa (casting operators):

`static_cast <novi_tip> (izraz)`

koristi se za sve dobro definisane konverzije (nema provere validnosti konverzije u vreme izvršavanja):

```
double result = static_cast<double>(4)/5;
```

`dynamic_cast <novi_tip> (izraz)`

konvertuje podatak iz jednog tipa u drugi tip pri čemu se vrši provera validnosti konverzije u vreme izvršavanja (ako su navedeni tipovi nekompatibilni vraća 0),

```
class Base{virtual void dummy(){} }; class Derived: public Base{int i;};
Base * pb = new Derived;    Derived * pd= dynamic_cast<Derived*>(pb);
```

`const_cast <novi_tip> (izraz)`

koristi se za uklanjanje const ili volatile osobine promenljive (odredišni tip mora biti isti kao i izvorni tip):

```
const double PI = 3.14;
double* pdouble = const_cast<double*>(&PI);
```

`reinterpret_cast <novi_tip> (izraz)`

menja jedan tip u drugi (koristi se za promenu tipa između pokazivača koji ukazuju na nekompatibilne tipove) i koristite ga kada pouzdano znate da na primer `void*` zaista ukazuje na `novi_tip`

```
Zaba zaba;      Baba* pBaba = reinterpret_cast<Baba*>(&zaba);
```

Zadatak

```
//_____________________________fajl IspisBroja.h _____________________________
#pragma once
//IEEE 754: float zauzima 4 bajta (na nizoj lokaciji je nizi bajt)
//bitovi(od najznacajnijeg)  znacenje
//1                          (s) predznak
//8                          (K) karakteristika K=E+127, E=K-127
//23                         (m) mantisa
//broj u zadatku se racuna kao: ((-1) na s)*(1.m)*(2 na E)
class IspisBroja
{
    float fbroj;
  public:
        IspisBroja();
    void PostaviFloat(float broj);
    void IspisiBinarnoFloatClan();
};
//Primer: -0.75
// s = 1 jer je broj negativan
// 0.75 binarno je 1/2 +1/4 odnosno 0.11
// kako je broj predstavljen kao 1.m, onda 1.m = 0.11*(2 na 1),
// te je korekcija za E suprotna tj. E=-1
// kako je E=K-127, K=126 tj. 01111110
// konacno 1 01111110 10000000000000000000000
```

Zadatak

```
//_____________________________fajl IspisBroja.cpp _____________________________
#include "IspisBroja.h"
#include <iostream>
#include <iomanip>
using namespace std;
     IspisBroja::IspisBroja()              {  fbroj = 0;   }
void IspisBroja::PostaviFloat(float broj){  fbroj = broj;}

void IspisBroja::IspisiBinarnoFloatClan()
{
   unsigned char *p = reinterpret_cast<unsigned char*>(&fbroj);
   cout<<"FLOAT BROJ -------- BINARNO PRIKAZAN --------"<<endl;
   cout<<"           s kkkkkkkk mmmmmmmmmmmmmmmmmmmmmmm"<<endl;
   cout<<"-----------------------------------------------"<<endl;
   cout<<setw(10)<<fbroj<<" ";
   for(int i=3; i>=0; i--)
      for(int j=7; j>=0; j--)
      {
         (p[i]&(1<<j))?cout<<"1":cout<<"0";
         if((i==3 && j==7)||(i==2 && j==7)) cout<<" ";
      }
   cout<<endl;
}
```

Zadatak

```
//_____________________________fajl gde je main _____________________________
#include <iostream>
using namespace std;
#include "IspisBroja.h"
int main(){
   IspisBroja ib;
   ib.PostaviFloat(-0.75f);
   ib.IspisiBinarnoFloatClan();
   system("pause");return 0;
}

//_____________________________IZLAZ _____________________________
FLOAT BROJ -------- BINARNO PRIKAZAN --------
           s kkkkkkkk mmmmmmmmmmmmmmmmmmmmmmm
----------------------------------------------
     -0.75 1 01111110 10000000000000000000000
Press any key to continue . . .
```

```
//3. cas, mozda da rese ovu minijaturu
#include <string.h>
#include <iostream>
using namespace std;

class Automobil{
   string naziv;
   double cena;
public:
   Automobil(string naziv, double cena):naziv(naziv),cena(cena){}
   void Ispisi() { cout << naziv.data() << " " << cena << std::endl; }
   double GetCena(){ return cena; }
};

int poredi(const void *a, const void *b){
   return static_cast<int>((*(Automobil*)a).GetCena() - (*(Automobil*)b).GetCena());
}

int main(void){
  Automobil salon[] = { { "Audi", 80000 },{ "BMW", 30000 },{ "Mercedes", 100000 } };
  qsort(salon, sizeof(salon)/sizeof(Automobil), sizeof(Automobil), poredi);
  for (int i = 0; i < sizeof(salon) / sizeof(Automobil); i++) (salon + i)->Ispisi();

  system("pause"); return 0;
}
```