SISTEMSKO PROGRAMIRANJE

OPERATORI, KONTROLA TOKA, PETLJE, ZNAKOVNI ULAZ/IZLAZ

1. OPERATORI

Operatori su simboli koji omogućavaju izvršavanje operacija nad vrednostima i promenljivima. Operatori mogu imati jedan, dva ili tri argumenta s tim što većina ima dva. Na primer, operator dodele ima dva argumenta – memorijsku lokaciju na levoj strani simbola = i izraz na desnoj strani. Ti argumenti se nazivaju operandi – vrednosti kojima se operiše. Operatori se mogu podeliti i prema broju operanada sa kojima rade. Većina operatora su binarni operatori što znači da se kombinuju dva izraza u jedan složeniji izraz. Postoje i unarni operatori koji vrše konverziju jednog izraza u drugi izraz.

- **Aritmetički operatori**

Aritmetički operatori su jednostavni, oni predstavljaju računske operacije.

Tabela 1. Aritmetički operatori

Operator	Opis	Primer
+	Sabiranje	a+b
-	Oduzimanje	a-b
*	Množenje	a*b
/	Deljenje	a/b
%	Moduo	a%b

Zadatak 1. Napisati program koji sa tastature čita dva cela broja i ispisuje njihov zbir, razliku, proziv, količnik i ostatak.

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{
   int a, b;
   printf("Unesite broj:", a);
   scanf("%d", &a);
   printf("Unesite broj:", b);
   scanf("%d", &b);
   printf ("a + b = %d\n", a+b);
   printf ("a - b = %d\n", a-b);
   printf ("a * b = %d\n", a*b);
   printf ("a / b = %f\n", (float)a/b); //cast operator
   printf ("a % b = %d\n", a%b);
   return 0;
}
```

Listing 1. Rešenje zadatka 1

- ***Cast* operator**

Svaki izraz daje neku vrednost čiji tip zavisi od tipova članova izraza. Kada su u nekom izrazu svi članovi istog tipa, tada je i vrednost izraza tog tipa.

```
int x = 3;
float y = x;
```

Listing 2. Automatska konverzija tipova

Kod sa listinga 2 je ispravan, iako se u drugom redu promenljivoj *y* dodeljunje tip *float* dodeljuje vrednost promenljive *x* koja je tipa *int*. Tu dolazi do automatske konverzije tipa *int* u *float*. U ovom slučaju ne dolazi do gubljenja podataka, dok bi za konverziju tipa *float* u *int* dovela do gubitka podataka, a to su cifre koje se nalaze iza decimalnog zareza.

- Implicitna konverzija

Razlikujemo dve situacije u kojima se primenjuje implicitna konverzija:

a) Pri izračunavanju vrednosti izraza kompajler automatski usklađuje tipove podataka koji se koriste u izrazu
b) Pri dodeljivanju vrednosti promenljivoj, vrednost se automatski konvertuje u tip promenljive

Osnovna ideja implicitne konverzije je da se ne gube informacije i da se sačuva preciznost podataka. Postoje pravila konverzije po kojima kompajler automatski izvršava implicitnu konverziju. Iplicitna konverzija se najčešće primenjuje kod numeričkih tipova podataka.

Zadatak 2. Izračunati vrednosti promenljivih x,y,z nakon izvršavanja naredbi.

```
int a = 15, b = 4, x;
double y,z, c = 15.0;
x = a / b;
y = a / b;
z = c / b;
```

Listing 2. Rešenje zadatka 2.

- Eksplicitna konverzija

Koristi se kod onih konverzija koje se ne mogu implicitno izvršiti. Eksplicitnom konverzijom programer, dodatnim kodom, od kompajlera zahteva traženu konverziju.
Eksplicitna konverzija se ostvaruje korišćenjem operatora *cast* ili korišćenjem klase *Convert*. Operator *cast* koristimo tako što se u zagredama navodi tip u koji se želi izvršiti konverzija vrednosti izraza koji sledei za operatorom *cast*.

Zadatak 3. Odrediti vrednosti promenljive x nakon izvršavanja naredbi.

```
int a = 11, b = 4;
double x, y;
x = a / b;
y = (double)a / b;
```

Listing 3. Rešenje zadatka 3

Objašnjenje:
U prvom delu *a* i *b* su tipa *int,* pri računanju vrednosti izraza *a / b* ne dolazi do implicitne konverzije i vrednost izraza je ceo broj 2, čime je došlo do gubitka decimalnog dela (2,75). Pri dodeli vrednosti izraza *a / b* tipa *int* promenljivoj *x* tipa *double,* dolazi do implicitne konverzije (*int* u *double*) i vrdnost promenljive *x* je 2,0.

U drugom delu u izrazu *a / b* je upotrebljen operator *cast x = (double)a / b;* Prvo će se izvršavati eksplicitna konverzija vrednosti promenljive *a* u tip *double*. Sada su operandi različitog tipa pa se vrši implicitna konverzija vrednosti promenljive *b* iz tipa *int* u tip *double*. Posle izvršenih konverzija, računa se vrednsot izraza i dodeljuje promenljivoj *x*. Po završetku operacije dodele, vrednost promenljive *x je* 2,75.

- **Relacijski i logički izrazi (operatori)**

Logički izrazi služe za poređenje različitih podataka i za druge logičke operacije. To su izrazi koji su sastavljeni od logičkih operanada povezanih logičkim operatorima. Logički operandi su relacijski izrazi ili logički izrazi.

Prioriteti logičkih operatora:

Logički operatori imaju veći prioritet od relacionih operatora.
< - manje,
> - veće,
<= - manje ili jednako,
>= - veće ili jednako,
== - dvostruko jednako (operator dodele),
!= - različito (! je operator negacije),
&& - logičko I (logički veznik),
|| - logičko ILI

Definisanje logičkih izraza

a) Logičke konstante *True* i *False* su logički izrazi
b) Logičke promenljive (tipa *Boolean*) logički izraz
c) Relacijski izrazi su logički izrazi
d) Pozivi logičkih funkcija su logički irazi
e) Ako su A1 i A2 logički izrazi, onda su logički izrazi i (*!A1*), (*!A2*), (*A1||A2*), (*A1 &&A2*)

- **Bitovni operatori**

Bitovni opeartori su operatori koji manipulišu sa celobrojnim tipovima podataka, na nivou bitova. Za potpuno razumevanje rada ovih operatora potrebno je poznavati način zapisivanja celih brojeva u potpunom komplementu. Postoji šest operatora za rad sa bitovima.

& - binarno I,
| - binarno ILI,
^ - binarno ekskluzivno ILI,
<< - levi shift,
>> - desni shift,
~ - unarna negacija

- **Operatori uvećavanja i umanjivanja**

U programskom jeziku C postoji dva operatora koja povećavaju i smanjuju za jedan. To su operatori *i*++ i *i*--. Operator inkrement ++ uvećava svoj operand za jedan, dok operator dekrement - - umanjuje svoj operand za jedan.
U prevodu, izraz *x = x+1* i *x*++ i *x = x-1* i *x*-- imaju isto značenje.
Isto tako, ti operatori se mogu koristiti i kao prefiksni *(*++*x*) i kao sufiksni (*x*++). Razlika je u tome što se u prefiksnom delu korišćenja vrednosti promenljive *x* uveća pre njegovog korišćenja, a u sufiksnom je obrnuto.

- **Operatori i izrazi dodeljivanja vrednosti**

Slično kao kod operatora inkremeta i dekrementa, postoje i skraćeni zapisi za operatore u kojima je promenljiva na levoj strani, a neki od operatora na desnoj strani, npr. *x = x+2*. Skraćeni zapis za ovaj izraz je *x+=2*.
Ovakvi skraćeni zapisi su mogući za sledeće operatore (+, -, *, /, %, <<, >>, &, ^, |).

2. KONTROLA TOKA

- ***if-else* iskaz**

Izvršavanje *if-else* naredbe je slično izvršavanju *if* naredbe, osim što u slučaju kada je vrednost logičkog izraza u zagradi jednaka *false*, izvršava se maredba 2 i preskače naredba1. U drugom slučaju, kaka je vrednost logičkog izraza jednaka *True*, izvršava se naredba1 i preskače se naredba2. Time se u oba slučaja izvršavanje *if-else* naredbe završava i program se nastavlja od naredbe koja sledi iza *if-else* naredbe.

Primer:

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{
 int a = 100;
 if( a < 20 )
 {
        printf("a je manje 20\n" );
 }
 else
 {
        printf("a nije manje 20\n" );
 }
 printf("Vrednost promenljive a je: %d\n", a);
 return 0;
}
```

Listing 4. Primer iskaza if-else

- ***if-else if-else* iskaz**

If-else naredba ima samo dve grane, odnosno kada je uslov ispunjen i kada uslov nije ispunjen. Međutim, nekada je potrebno preispitati više uslova u zavisnosti od njhovih vrednosti i izvršiti određene naredbe. U tim slučajevima se koriste *if-else if-else* iskaz.

Primer:

```
#include <stdio.h>
#include <stdlib.h>

int main ()
{
 int a = 100;
 if( a == 10 )
{
        printf("Vrednost promenljive a je 10.\n" );
 }
 else if( a == 20 )
```

```
 {
        printf("Vrednost promenljive a je 20.\n" );
 }
 else if( a == 30 )
 {
        printf("Vrednost promenljvie a je 30.\n" );
 }
 else
 {
        printf("Nijedan iskaz nije tacan.\n" );
 }
 printf("Tacna vrednost promenljive a je: %d\n", a );
 return 0;
}
```

Listing 5. Primer iskaza if-else if-else

- **Operator *switch***

Operator *switch* omogućava grananja u programu izborom jednog od više ponuđenih operatora. Programiranje korišćenjem *switch* operatora je dosta pogodnije neko korišćenje višestrukog *if else -- else if* iskaza.

Primer:

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
 char ocena = 'B';
 switch(ocena)
 {
          case 'A' :
          printf("Odlican!\n" ); break;
          case 'B' :
          case 'C' :
          printf("Vrlo dobar.\n" ); break;
          case 'D' :
           printf("Prosao si.\n" ); break;
          case 'F' :
           printf("Pokusaj ponovo.\n" ); break;
          default :
          printf(„Pogresna ocena.\n“ );
 }
 printf(„Tvoja ocena je %c\n“, ocena );
 return 0;
}
```

Listing 6. Primer iskaza switch

3. PETLJE (CIKLUS)

- ***For***

Petlja *for* je kontrolna struktura sa višestrukim ponavljanjem koja omogućava efikasno pisanje ciklusa koja treba da se izvrši tačno određen broj puta.

Zadatak 4. Napisati program koji štampa prvih pet prirodnih brojeva.

```
#include <stdio.h>
#include <stdlib.h>
int main ()
{
        int i;
        for(i=1; i<=5; i=i+1)
        printf("%d\n", i);
        return 0;

}
```

Listing 7. Rešenje zadatka 4.

Iskaz *for* ima sledeći oblik:

```
for (inicijalizacija; uslov; korak)
```

Listing 8. Oblik for iskaza

inicijalizacija – izraz u kojem se promenljivima dodeljuju početne vrednosti
uslov – logički izraz koji uslovljava izvršenje *for* petlje – petlja se izvršava sve dok je ovaj logički izraz tačan,
korak – izraz u kojem se vrši uvećanje brojača za jedan.

Iza svakog koraka u *for* petji nalazi se karakter „ **;** “, a iza cele petlje se ne nalazi karakter „**;**“. To je zato što je kraj komande tek u sledećem redu.

- ***while***

Petlja *while* se izvršava sve dok je neki uslov tačan. Provera uslova se vrši na početku bloka, ako je uslov tačan, blok se izvršava, inače se izlazi iz petlje. Sintaksa ovog iskaza je oblika:

```
while(uslov)
blok-iskaz
```

Listing 9. Sintaksa while iskaza

Zadatak 5. Rešiti prethodni zadatak korišćenjem *while* petlje.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
   int i=1;
   while(i<=5)
   {
      printf("%d\n", i);
      i++;
   }
return 0;
}
```

Listing 10. Rešenje zadatka 5.

- ***do-while***

Kao i *while* petlja, samo što se kod *do-while* petlje prvo izvršavaju naredbe pa se onda proverava uslov. Za razliku od *while* petlje, naredbe u *do-while* petlji će se izvršiti bar jedan put. Ova petlja ima sledeći oblik:

```
do
{
        Naredbe…
} while (uslov);
```

Listing 11. Oblik do-while petlje

Zadatak 6. Rešiti prethodni zadatak korišćenjem *do-while* petlje.

```
#include <stdio.h>
#include <stdlib.h>

int main()
{
int i=1;
do
{
   printf("%d\n", i);
   i++;
} while(i<=5);
return 0;
}
```

Listing 12. Rešenje zadatka 6

- ***break*** i ***continue***

Nekada je poželjno imati mogućnost izlaska iz petlje, pored glavnog uslova. Naredba *break* će osigurati izlaz, odnosno prekid kruženja bilo koje petlje. Koriteći ključnu reč *break* moguće je izaći iz petlje u bilo kom trenutku.
Naredba *continue* je usko povezana s naredbom *break.* Njen zadatk je da otpočne sledeću iteraciju petlje u kojoj se program vrti. Ona se koristi samo u petljama, za razliku od *break* koji se koristi i kod *switch* petlje.

Primer za iskaz *break*:

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
  int i;
  while (i < 10)
  {
    i++;
    printf("%d\n", i);
    if (i == 5) break;
  }
  return 0;
}
```

Listing 13. Primer zadatka sa iskazom break

Ključna reč *continue* se koristi kako bi se preskočio ostatak trenutnog prolaska kroz petlju i tako otpočeo novi prolaz.

Primer za *continue*:

#include <stdio.h>
#include <stdlib.h>

```
int main ()
{
   int a = 10;
   do
   {
      if( a == 15) {
      a = a + 1;
      continue;
   }
   printf("Vrednost promenljive a je: %d\n", a);
   a++;
 }while( a < 20 );
 return 0;
}
```

Listing 14. Iskaz continue

4. ZNAKOVNI ULAZ I IZLAZ

U standardnoj biblioteci, osim funkcija *printf* i *scanf* postoje funkcije koje služe za ulaz odnosno izlaz znakovnog tipa podataka. To su funkcije *getchar* i *putchar*.
Funkcija *getchar()* čita jedan karakter sa standarnog ulaza (*keyboard*) i kao rezultat vraća njegovu ASCII vrednost i čita jedan karakter sa tastature i njegovu vrednost dodeljuje promenljivoj c.

```
char c = getchar();
putchar (c);
```

Listing 15. Primer funkcije getchar, putchar

Funkcija *putchar (c)* štampa vrednost karaktera *c* na standardnom izlazu.
Kod unosa teksta, kao indikator za kraj unosa se koristi znak za kraj datoteke, koji se označava sa *EOF* (*end of file*).