UNIVERZITET METROPOLITAN BEOGRAD

CS130 - C/C++ PROGRAMSKI JEZIK

Uslovni iskazi i petlje, Funkcije

Lekcija 02

PRIRUČNIK ZA STUDENTE

CS130 - C/C++ PROGRAMSKI JEZIK

Lekcija 02

USLOVNI ISKAZI I PETLJE, FUNKCIJE

UVOD

Ova lekcija treba da ostvari sledeće ciljeve:

U okviru ove lekcije biće opisane konstrukcije koje postoje u C-u za upravljanje tokom programa tj. putanjom programa: if, if-else, if-else if i case-switch.

U C-u je moguće koristiti petlje u cilju ponavljanja izvršenja nekog segmenta koda. Stoga će biti opisane ***for***, ***while*** i ***do while*** petlje.

Na kraju lekcije ćemo opisati specifičnosti koje se odnose na korišćenje funkcija u C programu.

Poglavlje 1

Uslovni iskazi i grananja

TIPOVI USLOVNIH INSTRUKCIJA

Uslovne instrukcije zahtevaju od korisnika da specificira jedan ili više uslova koji će biti ispitani od strane programa, ali i sekvencu izraza koja će se izvršiti ukoliko je uslov zadovoljen

Instrukcije donošenja odluka (uslovne ili logičke instrukcije) zahtevaju od korisnika da specificira jedan ili više uslova koji će biti ispitani ili testirani od strane programa, a takođe i da specificira izraz ili sekvencu izraza koji će se izvršiti ukoliko je odgovarajući uslov zadovoljen i, opciono, ukoliko uslov nije zadovoljen.

C (C++) ima nekoliko različitih tipova logičkih instrukcija kao što su: ***if***, ***if else***, operator *?*, ***switch***. Na Slici 1.1 su prikazani tipovi mogućih uslovnih instrukcija u jeziku C(C++).

Iskaz	Opis
if iskaz	Jedan **if iskaz** se sastoji iz uslovnog izraza za kojim sledi jedan iskaz ili blok iskaza.
if...else iskaz	Jedan **if iskaz** može da prethodi opcionim else iskazom, čiji se blok izvršava ukoliko je uslovni izraz netačan.
ugnježđeni if iskaz	Mogu se koristiti jedan **if** ili **else if** iskaz unutar drugog ili više **if** ili **else if** iskaza.
switch iskaz	Iskaz **switch** omogućava ispitivanje vrednosti promenljive u odnosu na listu definisanih vrednosti.
ugnježđeni switch iskaz	Jedan **switch** iskaz može biti ugnježđen unutar drugog ili više **switch** iskaza

Slika 1.1 Osnovne uslovne instrukcije u C (C++) programskom jeziku [5]

Uslovni operator ?

Test operator tj. uslovni operator testira da li je neki izraz tačan ili ne, i onda izvršava jednu od dve date instrukcije u zavisnosti od rezultata testa. Sintaksa je sledeća:

```
(test-izraz) ? ako-tačno-uradi-ovo :
               ako-netačno-uradi-ovo ;
```

Primer korišćenja je u nastavku:

```
int num1 = 12345, num2 = 23456;
char znak;
printf("%d\n", num1);

(num1 < num2 ) ? printf("%d\n", num1)
               : printf("%d\n", num2);
printf("%d\n", num2);
```

Dakle, operator *?* je upotrebljen da izabere jedan od dva poziva *printf* u zavisnosti da li je broj num1 veći od broja num2 ili ne.

ISKAZI IF, IF-ELSE I IF-ELSEIF-ELSE

Ukoliko je izraz tačan izvršiće se blok koda u okviru if iskaza. U suprotnom, ukoliko je izraz netačan, izvršiće se prvi segment koda koji sledi po završetku if bloka

U programskom jeziku C (C++), if iskaz može biti konstruisan na sledeći način:

```
if(izraz)
{
   /* iskazi se izvrsavaju ako je uslov tačan*/
}
```

Pravila su ista kao i u Python jeziku. Ukoliko je izraz tačan (true), izvršiće se blok koda u okviru ***if*** iskaza.

Iskaz ***if*** može biti ugnježden unutar jednog ili više drugih ***if*** iskaza:

```
if(uslovni_izraz1)
{
   /*izvrsava se ako je uslovni_izraz1 tacan*/
   if(uslovni_izraz2)
   {
    /*izvrsava se ako je uslovni_izraz2 tacan*/
   }
}
```

Sintaksa iskaza if-else u programskom jeziku C (C++) ima sledeći oblik:

```
if(uslovni izraz)
{
   /* iskazi koji se izvrsavaju ako je uslov tacan*/
}
else
{
  /*iskazi koji se izvrsavaju ako je uslov netacan*/
}
```

Sintaksa ***if-else if-else*** iskaza u programskom jeziku C/C++ ima sledeći oblik:

```
if(uslovni_izraz1)
{
   /* Izvršava se kad je uslovni_izraz1 tačan */
}
else if(uslovni_izraz2)
{
   /* Izvršava se kad je uslovni_izraz1 tačan */
```

```
}
else
{
   /* izvršava se kada nijedan od */
   /* prethodnih uslova nije tačan */
}
```

LOGIČKI OPERATORI I USLOVNI ISKAZI

Logički operatori !, &&, || mogu biti korišćeni u cilju formiranja složenog logičkog izraza koji se ispituje korišćenjem if iskaza

- Operator && (Logičko "i" – "AND")

Primer korišćenja operatora &&: Ispitati da li je osoba državljanin i da li je starija od 18 godina. Ukoliko jeste, ima pravo glasa; u suprotnom nema pravo glasa:

```
if (age >= 18 && citizen == 1)
    printf("Dozvoljeno vam je da glasate");
else
    printf("Nije vam dozvoljeno da glasate");
```

- Operator || (Logičko "ili" - "OR")

Primer korišćenja operatora ||: Gradski prevoz je besplatan za sve osobe koje su mlađe od 12 ili starije od 65 godina:

```
if (age <= 12 || age >= 65)
    printf("Prevoz vam je besplatan");
else
    printf("Morate da platite prevoz");
```

- Operator *!* (Logičko "ne" – "NOT")

Primer korišćenja operatora *!*: Gradski prevoz je besplatan za sve osobe koje nisu starije od 12 godina i nisu mlađe od 65 godina:

```
if (!(age > 12 && age < 65))
    printf("Prevoz vam je besplatan");
else
    printf("Morate da platite prevoz");
```

ISKAZ SWITCH

Ovaj operator omogućava grananja u programu izborom jednog od više ponuđenih operatora.

Operator switch omogućava grananja u programu izborom jednog od više ponuđenih operatora. Operator ***switch*** ima sledeći oblik:

```
switch(izraz)
{
case konstanta1:
 operator1;
 ...
 break;
case konstanta2:
 operator2
 ...
 break;
default:
 operatorN1;
 ...
 break;
}
```

Nakon rezervisane reči ***switch*** se navodi *izraz* koji se naziva selektor. Vrednost selektora je uglavnom celobrojna ili znakovna.

Operatorom višestrukog izbora se izvršava ona grupa komandi ispred koje se nalazi konstanta čija je vrednost jednaka selektoru. U slučaju da vrednost selektora nije jednaka ni jednoj od navedenih konstanti, izvršiće se grupa komandi koja se definiše iza ***default*** opcije. U slučaju da se izostavi ***default*** opcija, neće se ništa izvršiti.

UPOTREBA ISKAZA “SWITCH”

Programiranje korišćenjem switch operatora je nekada znatno efikasnije i pogodnije nego korišćenje višestrukog if else – else if uslovnog operatora

Pretpostavimo da imamo zadatak da u zavisnosti od ocene koju je đak dobio na testu, treba odštampati uspeh. Zadatak možemo korišćenjem ***switch*** instrukcije uraditi na sledeći način:

```
#include <stdio.h>
int main ()
{
   char grade = '5';
   switch(grade)
   {
   case '5' :
    printf("Odlicno!\n" );    break;
   case '4' :
    printf("Vrlo dobro\n" );     break;
   case '3' :
    printf("Dobro\n" );     break;
   case '2' :
    printf("Dovoljno\n" );    break;
```

```
   case '1' :
    printf("Nedovoljno, pokusajte ponovo\n" );      break;
   default :
    printf("Nevazeca ocena\n" );
   }
   printf("Vasa ocena je %c\n", grade );
   return 0;
}
```

Rezultat prethodnog programa je:

```
Vrlo dobro
Vasa ocena je 4
```

Ugnježdeni “switch” iskaz

C vam dozvoljava da imate ***switch*** iskaz unutar nekog spoljašnjeg ***switch*** iskaza. Bez obzira da li ***case*** konstante unutrašnjeg i spoljašnjeg ***switch*** iskaza sadrže iste vrednosti, neće doći do konflikta i program će se normalno izvršiti. U nastavku je primer gde u oba ***switch*** izraza imamo proveru nad konstantom *'A'*:

```
switch(ch1) {
   case 'A':
    printf("Ovo A je deo spoljasnjeg switch" );
    switch(ch2) {
        case 'A':
            printf("Ovo A je deo unutrasnjeg switch" );
            break;
        case 'B': /* kod koji odgovara ovom case */
    }
    break;
   case 'B': /* kod koji odgovara ovom case */
}
```

Poglavlje 2

Ciklusi (petlje)

TIPOVI CIKLUSA (PETLJI)

Postoji veliki broj situacija gde je potrebno da se određen skup operacija izvrši više puta. Takav niz operacija koji će se izvršiti više puta naziva se ciklus ili petlja

U zavisnosti od položaja izlaznog kriterijuma u odnosu na telo ciklusa, ciklusi ili petlje mogu biti sa preduslovom (***while***) i sa postuslovom (***do while***). Ciklus ***while*** se skraćeno može zapisati pomoću operatora ciklusa ***for***. U nastavku je dat spisak i kratak opis petlji u C-u:

Tip petlje	Opis
while petlja	Ponavljanje iskaza ili grupe iskaza sve dok je ispunjen uslov. Uslov se ispituje pre izvršavanja tela petlje.
for petlja	Višestruko izvršavanje grupe iskaza uz skraćeni zapis promenljivih koje kontrolišu izvršavanje petlje.
do...while petlja	Kao i while petlja, s tim što se uslov ispituje tek na kraju tela (bloka) petlje
ugneždene petlje	Korišćenje petlji unutar bilo koje druge while, for ili do..while petlje.

Slika 2.1 Osnovni tipovi ciklusa [5]

Sintaksa petlje while ima sledeći oblik:

```
while(uslov)
{
   iskaz (grupa iskaza);
}
```

U nastavku je data osnovna sintaksa ***do-while*** petlje:

```
do
{
   iskaz(i);
}while( uslov);
```

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

PETLJA FOR

Ciklus for je kontrolna struktura sa višestrukim ponavljanjem koja omogućava efikasno pisanje petlje koja treba da se izvrši tačno određen broj puta

Operator ciklusa (petlje) for ima sledeći oblik:

```
for ( inicijalizacija; uslov; korekcija)
{
   skup iskaza;
}
```

Tok instrukcija u ***for*** petlji je sledeći:

- *Inicijalni korak se izvršava prvi, i samo jednom. Ovaj korak omogućava deklaraciju i inicijalizaciju kontrolne promenljive (promenljiva koja kontroliše petlju). Inicijalizacija nije neophodna, dovoljno je navesti oznaku **"tačka zarez".***
- *Zatim se ispituje uslov. Ukoliko je uslov tačan izvršava se telo petlje. Ukoliko je netačan ne izvršava se telo petlje a tok program se pomera na prvu liniju koja sledi posle tela **for** petlje.*
- *Nakon izvršavanja tela **for** petlje, dolazi se do iskaza korekcije. Ovaj iskaz omogućava korekciju bilo koje od kontrolnih promenljivih. Takođe može ostati prazan, ali je neophodno navesti **"tačka zarez"** nakon uslova.*

Ponovo se ispituje uslov i ako je tačan proces se ponavlja korak po korak (telo petlje, iskaz korekcije, i ponovo uslov) sve dok uslov ne postane netačan, kada dolazi do okončanja petlje.

Primer korišćenja for petlje:

```
#include <stdio.h>
int main ()
{
   int a;

   /* for loop execution */
   for( a = 10; a < 20; a = a + 1 ){
      printf("value of a: %d\n", a);
   }
   return 0;
}
```

Rezultat:

```
value of a: 10
value of a: 11
value of a: 12
value of a: 13
value of a: 14
```

```
value of a: 15
value of a: 16
value of a: 17
value of a: 18
value of a: 19
```

UGNJEŽDENI CIKLUSI

U programskom jeziku C moguće koristi petlje unutar bilo koje for, while ili do while petlje

U programskom jeziku C (C++) moguće je koristi petlje unutar bilo koje ***for***, ***while*** ili ***do-while*** petlje. U nastavku su dati primeri za ugnježdene ***for***, ***while*** i ***do-while*** petlje:

Ugnježdena ***for*** petlja:

```
for ( inicijalizacija; uslov; inkrement )
{
   for ( inicijalizacija; uslov; inkrement )
   {
    iskaz(i);
   }
   iskaz(i);
}
```

Ugnježdena ***while*** petlja:

```
while(uslov)
{
   while(uslov)
   {
    iskaz(i);
   }
   iskaz(i);
}
```

Ugnježdena ***do-while*** petlja:

```
do
{
   iskaz(i);
   do
   {
    iskaz(i);
   }while(uslov);

}while(uslov);
```

KONTROLNI ISKAZ BREAK

Instrukcija break se koristi da se izađe iz instrukcije case unutar instrukcije switch, ali može i da se koristiti da se izađe iz bilo koje for, while ili do-while petlje

Instrukcija break je već korišćena da se izađe iz ***case*** bloka unutar ***switch*** konstrukcije. Ona se može takođe koristiti da se izađe iz petlje. Instrukcija ***break*** može se koristiti u kombinaciji sa uslovnim testom ***if***. Kada je testirajući izraz ***if*** nađeno da je tačan, petlja se završava (nema više iteracija). Sintaksa je:

```
if(test_izraz) break;
```

U nastavku je dat primer korišćenja ***break*** instrukcije za prekid rada ***while*** petlje.

```
int a = 10;
while( a < 20 )
{
    printf("vrednost od a je: %d\n", a);
    a++;
    if( a > 15)
    {   /* prekinite petlju pomocu break iskaza */
        break;
    }
}
```

KONTROLNI ISKAZ CONTINUE

Naredba continue se primenjuje sa ciljem da se preskoči jedna od iteracija u okviru petlje i najčešće je kombinovana sa uslovnim izrazom if

Naredba continue se primenjuje sa ciljem da se preskoči jedna od iteracija u okviru petlje. Naredba ***continue*** može se upotrebiti unutar nekog bloka u petlji, kombinovana sa uslovnim izrazom ***if***. Kada se uslov u okviru ***if*** pokaže kao tačan, tekuća iteracija se odmah prekida, ali sledeća iteracija će se nastaviti. Sintaksa je:

- ***if*** *(test_izraz)* ***continue****;*

U nastavku je da primer korišćenja ***continue*** instrukcije u cilju preskakanja jednog koraka ***while*** petlje:

```
#include <stdio.h>
void main ()
{
   int a = 10;
```

```
   do
   {
    if( a == 15)
    {  /* preskoci ovu iteraciju */
        a = a + 1;
        continue;
    }
    printf("vrednost od a je: %d\n", a);
    a++;
   }while( a < 20 );
}
```

Poglavlje 3

Funkcije u C-u

DEFINICIJA FUNKCIJE

Funkcija je skup ili grupa iskaza koji zajedno kao jedna celina obavljaju neki predviđeni zadatak

Svaki C (C++) program se sastoji iz bar jedne funkcije a to je funkcija *main*, ali je moguće definisati neograničen broj funkcija u programu. Osnovni oblik definicije funkcije u programskom jeziku C je:

```
povratni_tip naziv_funkcije( lista parametara )
{
   telo funkcije;
}
```

Definicija funkcije se sastoji iz ***zaglavlja*** (header) i ***tela*** (body) funkcije, pri čemu se zaglavlje funkcije sastoji iz povratne vrednosti, naziva funkcije i liste parametara (argumenata):

- Povratna vrednost - return type: može biti tipa ***void*** ili bilo kog validnog tipa podatka (*int*, *double*, itd).

- Naziv funkcije: Ime funkcije zajedno sa listom parametra predstavlja potpis funkcije.
- Parametri (argumenti): Kada se funkcija poziva, njoj se prosleđuju vrednosti kroz listu parametara. Ove vrednosti predstavljaju stvarne parametre ili argumente funkcije. Funkcija može i da ne sadrži nijedan parametar.

Telo funkcije se sastoji iz skupa instrukcija koji definišu šta funkcija radi.

U nastavku je primer funkcije *max()* koja preuzima dva parametra *num1* i *num2*, i kao rezultat vraća veći broj:

```
int max(int num1, int num2)
{
    int result;

    if (num1 > num2)
        result = num1;
    else
        result = num2;

    return result;
}
```

Veoma često se koristi deklaracija funkcije, naročito pri radu sa više fajlova.

Deklaracija funkcija se sastoji iz sledećih delova:

```
povratni_tip naziv_funkcije( lista parametara);
```

Za prethodno navedenu funkciju *max*, deklaracija se izvodi na sledeći način:

```
int max(int num1, int num2);
```

POZIV FUNKCIJE

Poziv funkcije u C-u se ostvaruje jednostavnim navođenjem naziva funkcije iza koga sledi lista stvarnih parametara

Poziv funkcije u C-u se ostvaruje jednostavnim navođenjem naziva funkcije iza koga sledi lista stvarnih parametara. Ukoliko funkcija vraća vrednost onda je neophodno da postoji promenljiva koja će da prihvati povratnu vrednost funkcije. U nastavku je dat prost primer poziva funkcije *max* iz glavne funkcije *main*:

Ova lekcija sadrži video materijal. Ukoliko želite da pogledate ovaj video morate da otvorite LAMS lekciju.

```
#include <stdio.h>

/* deklaracija funkcije */
int max(int num1, int num2);

int main ()
{
   /* definicija lokalnih promenljivih */
   int a = 100;
   int b = 200;
   int ret;

   /* poyivanje funkcije da se odredi maksimum dva broja */
   ret = max(a, b);

   printf( "Maksimalna vrednost je : %d\n", ret );

   return 0;
}
```

OPSEG PROMENLJIVE

Opseg promenljive je region programa u kome je definisana promenljiva i u kome živi, i van koga ne može biti vidljiva niti dostupna.

Postoje tri mesta na kojima promenljiva može biti deklarisana kada je u pitanju C programski jezik:

- *Unutar bloka ili funkcije, i takva promenljiva se naziva lokalna (local) promenljiva,*
- *Izvan svih funkcija (pa i funkcije **main**), i naziva se globalna (global) promenljiva.*
- *U okviru liste parametara funkcije (tj. u zaglavlju funkcije), i takva promenljiva se naziva **formalni parametar funkcije***

U nastavku je primer korišćenja lokalnih promenljivih u ugnježdenim blokovima.

```
int main()
{ // spoljasni blok
    int nValue = 5;

    if (nValue >= 5)
    { // ugneždeni unutrašnji blok
        int nValue = 10;
    } // ugneždena nValue unistena

    // nValue se sada odnosi na promenljivu iz spoljasnjeg bloka
    return 0;
} // spoljasna nValue unistena
```

Treba primetiti da možemo deklarisati promenljive koje imaju ista imena u dva međusobno ugnježdena bloka.

Globalne promenljive su definisane izvan funkcija, i najčešće na početku programa (izvornog koda).U C programu dozvoljeno je da globalna i lokalna promenljiva imaju isto ime. Međutim, u okviru funkcije gde je deklarisana lokalna promenljiva, ona će imati prednosti u odnosu na globalnu promenljivu (koja će biti skrivena tokom izvršavanja lokalnog bloka). U nastavku je primer:

```
#include <stdio.h>
int g = 20; /* deklaracija globalne promenljive */
int main ()
{
  int g = 10; /* deklaracija lokalne promenljive */
  printf ("vrednost od g je g = %d\n",  g);
  return 0;
}
```

Rezultat će biti:

```
value of g = 10
```

Parametri funkcije, odnosno formalni parametri, se tretiraju kao lokalne promenljive koje žive sve dok postoji i funkcija koja ih koristi, i imaju prednost u odnosu na globalne promenljive sa istim nazivom.

REKURZIJA

Rekurzija je korisna u slučajevima kada je potrebno da se problem reši na način da se on prvo razloži na podproblem sa istim svojstvima ali manje dimenzije od originalnog

Rekurzija je slučaj kada funkcija poziva samu sebe. Takve funkcije se zovu rekurzivne funkcije, i C/C++ dozvoljava njihovo korišćenje. U nastavku je dat prost primer definicije i poziva rekurzivne funkcije iz glavnog programa.

```
void recursion()
{
   recursion(); /* funkcija poziva samu sebe */
}

void main()
{
   recursion();
}
```

Rekurzivne funkcije su veoma korisne za rešavanje mnogih matematičkih problema, kao što su Faktorijel broja ili Fibonačijeva serija brojeva.

POZIV FUNKCIJE PO VREDNOSTI

Kod poziva po vrednosti izvorni kod unutar funkcije radi lokalno i ne može da izmeni vrednost argumenata koji se prosleđuju funkciji

U programskom jeziku C, **poziv po vrednosti je podrazumevani način da se argumenti proslede funkciji**. Ovo znači da kod unutar funkcije radi lokalno i ne može da izmeni vrednost argumenata koji se prosleđuju funkciji. Neka je definisana funkcija ***swap*** na sledeći način.

```
void swap(int x, int y)
{
   int temp;

   temp = x; /* sačuvaj vrednost od x */
   x = y;   /* ubaci iz y u x */
   y = temp; /* ubaci iz temp u y */
```

```
   return;
}
```

Nakon izvršavanja programa na desnoj strani, koji poziva funckiju *swap*, dobiće se sledeći rezultat:

```
Pre razmene, vrednost od a :100
Pre razmene, vrednost od b :200
Nakon razmene, vrednost od a :100
Nakon razmene, vrednost od b :200
```

Na osnovu rezultata vidimo da iako je izvršena lokalna zamena vrednosti parametra unutar funkcije, to nije uticalo na promenu vrednosti stvarnih argumenata koji su prosleđeni funkciji.

Poziv funkcije *swap*, kojoj smo prosledili stvarne parametre iz glavnog programa, može biti ostvaren na sledeći način:

```
#include <stdio.h>

void swap(int x, int y); /* deklaracija funkcije */

int main ()
{
   int a = 100;
   int b = 200;

   printf("Pre razmene, vrednost od a : %d\n", a );
   printf("Pre razmene, vrednost od b : %d\n", b );

   swap(a, b); /* poziv funkcije koja zamenjuje vrednosti */

   printf("Nakon razmene, vrednost od a : %d\n", a );
   printf("Nakon razmene, vrednost od b : %d\n", b );

   return 0;
}
```

POZIV FUNKCIJE PO ADRESI

Unutar funkcije se koristi adresa da bi se pristupilo stvarnom parametru, što znači da se izmene vrše nad stvarnim argumentom

U programskom jeziku C se koriste pokazivači da bi se vrednost funkciji prosledila po adresi. Poziv po adresi u C-u se drugačije naziva i poziv po pokazivaču. Stoga je neophodno deklarisati funkciju tako da kao argumente prihvata pokazivačke promenljive. U nastavku je data izmenjena definicija funkcije *swap*, kojoj se prosleđuju adrese tako da se u njoj menjaju vrednosti dva cela broja na koja pokazuju argumenti funkcije.

```
void swap(int *x, int *y)
{
   int temp;
   temp = *x;   /* sačuvaj vrednost sa adrese x u temp */
   *x = *y;     /* ubaci iz y u x */
   *y = temp;   /* ubaci iz temp u y */

   return;
}
```

Izmenićemo način deklaracije funkcije *swap* kojoj prosleđujemo adrese kao parametar

```
void swap(int *x, int *y);
```

i izmenićemo način na koji se vrši pozivanje funkcije iz glavnog programa, tako da sada deo *main* funkcije iz koje se poziva *swap* izgleda ovako:

```
/* pozivanje funkcije koja razmenjuje vrednosti.
    * &a predstavlja pokazivač na a tj. adresu promenljive a,
    * &b predstavlja pokazivač na b tj. adresu promenljive b.  */
   swap(&a, &b);
```

Nakon izvršavanja prethodnog koda dobija se sledeći rezultat:

```
Pre razmene, vrednost od a :100
Pre razmene, vrednost od b :200
Nakon razmene, vrednost od a :200
Nakon razmene, vrednost od b :100
```

Vidimo da je sada izmenjen sadržaj promenljivih *a* i *b* nakon povratka iz funkcije *swap*.

3.1 Rezultat funkcije

REZULTAT FUNKCIJE JE VREDNOST

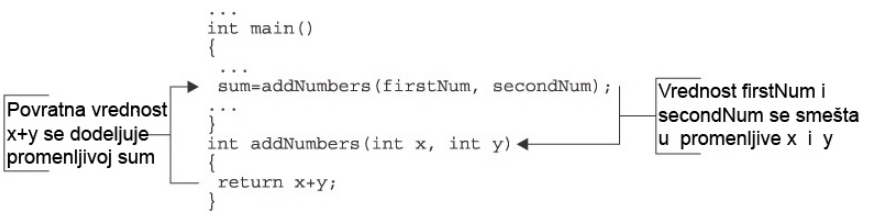
Kada se rezultat funkcija vraća po vrednosti, kopija rezultata se vraća pozivaocu, a ovaj metod funkcioniše na isti način kao poziv po vrednosti

Povratak po vrednosti je najprostiji i najbezbedniji povratni tip, odnosno tip rezultata funkcije. Kada se rezultat funkcije vraća po vrednosti, kopija rezultata se vraća pozivaocu. Druga prednost povratka po vrednosti je to što se mogu vratiti promenljive i izrazi u kojima učestvuju lokalne promenljive funkcije. Primer je dat u nastavku:

```
int DoubleValue(int nX)
{
```

```
    int nValue = nX * 2;
    return nValue; // Kopija promenljive nValue se ovde vraća pozivaocu
} // nValue ovde izlazi iz opsega
```

Na Slici 4.1 je detaljan opis postupka povratka po vrednosti.



Slika 3.1.1 Redosled izvršavanja pri pozivu i povratku iz funkcije [1]

REZULTAT FUNKCIJE JE ADRESA (POKAZIVAČ)

Povratak po adresi može vratiti samo adresu promenljive, a nikako literal ili izraz. Povratak po adresi se najčešće koristi da se vrati novo-alocirana memorijska adresa

Povratak po adresi predstavlja vraćanje adrese pozivaocu. Povratak po adresi može vratiti samo adresu promenljive, a nikako literal ili izraz. Povratak po adresi je veoma brz, brži od povratka po vrednosti, i kod njega se ne mogu se vratiti lokalne promenljive funkcije. Primer je u nastavku:

```
int* DoubleValue(int nX)
{
    int nValue = nX * 2;
    // ovde se vraća nValue po adresi:
    return &nValue;
} // nValue ovde izlazi iz opsega
```

Kao što vidimo *nValue* izlazi iz opsega nakon ***return*** poziva i povratka iz funkcije. To znači da pozivalac dobija adresu nealocirane memorije, što će izazvati problem.

Povratak po adresi se najčešće koristi da se vrati novo-alocirana memorijska adresa (o operatoru *malloc* i dinamičkom alociranju će biti više reči u narednim lekcijama, kada će biti data opširna objašnjenja).

Poglavlje 4

Program sa više fajlova

UPOTREBA VIŠE FAJLOVA U C PROGRAMU

Sa porastom obima i složenosti programa postoji potreba da se kod podeli u više različitih fajlova u cilju što bolje organizacije

Kako programi postaju obimniji i složeniji, postoji opravdana potreba da se kod bolje organizuje i podeli u nekoliko različitih fajlova.

Pretpostavimo da imamo dva fajla, jedan je glavni ***main.c*** fajl, a drugi je pomoćni fajl ***add.c*** u kome se nalazi definicija neke funkcije za sabiranje dva broja:

add.c :

```
int add(int x, int y)
{
    return x + y;
}
```

main.c :

```
#include <stdio.h>
int add(int x, int y); // deklaracija unapred, koriscenjem prototipa funkcije
int main()
{
    printf("Zbir brojeva 3 i 4 je : %d", add(3, 4));

    return 0;
}
```

Na ovaj način smo definiciju funkcije *add* smestili u poseban fajl i uprostili smo ***main.c*** fajl. Sličan princip koristimo ako imamo desetine ili stotine funkcije koje će po potrebi u datom trenutku biti pozvane iz glavnog programa aplikacije.

Treba napomenuti da u programskom jeziku C ne morate navesti deklaraciju funkcije *add* u fajlu sa glavnom *main* funkcijom. C kompajler će i bez navođenja deklaracija iskompajlirati program.

PISANJE SOPSTVENIH HEADER FAJLOVA

Fajlovi zaglavlja koje sami napravimo se u program uključuju direktivom #include iza koje sledi ime fajla uokvireno znakovima “ i “ (#include “file.h“)

Pisanje sopstvenih fajlova je iznenađujuće lak posao. Header fajl se sastoji iz dva dela. Prvi deo je rezervisan za pretprocesorske naredbe, a drugi deo je sadržaj header fajla. U drugom delu se nalazi deklaracija svih funkcija koje želimo da budu vidljive i korišćene od strane delova koda iz drugih fajlova. Svi korisnički napravljeni header fajlovi treba da budu sa ekstenzijom *.h*. U nastavku je dat primer korišćenja header fajla ***add.h*** :

```
int add(int x, int y); // prototip funkcije za add.h
```

U fajlu ***add.h*** se nalazi deklaracija funkcije, dok se u sledećem fajlu, ***add.c*** , nalazi njena definicija:

```
int add(int x, int y)
{
    return x + y;
}
```

U cilju uključivanja ovog fajla u glavni fajl, neophodno je da ga uključimo odgovarajućom ***#include*** naredbom. Fajl ***main.c*** koji uključuje ***add.h*** header fajl će imati sledeći oblik:

```
#include <stdio.h>
#include "add.h" // ovako ukljucujemo deklaraciju funkcije add()

void main()
{
    printf("Zbir brojeva 3 i 4 je %d",add(3, 4));
}
```

Dakle, u opštem slučaju, korisnički definisan *.h fajl se uključuje direktivom:

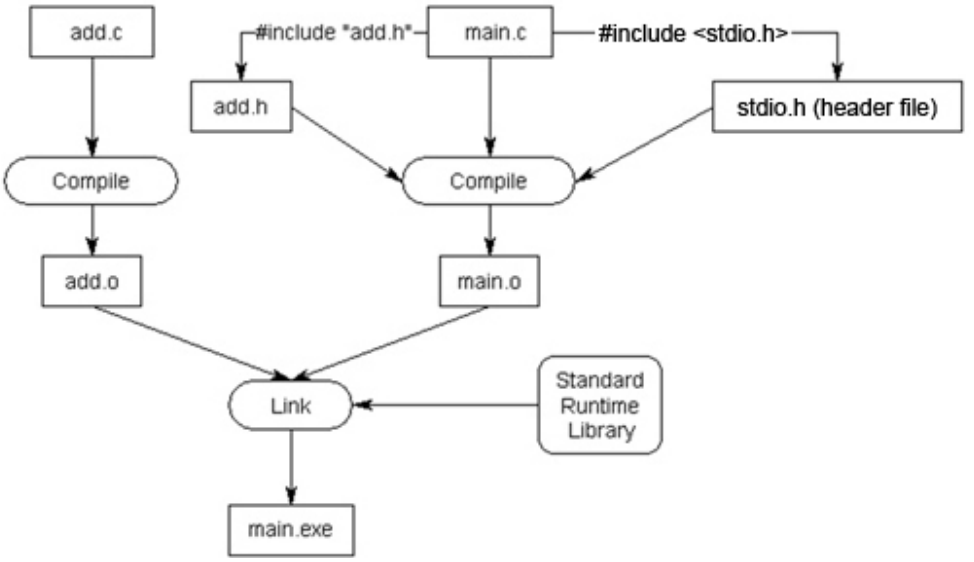
```
#include "file"
```

Na ovaj način se ukazuje kompajleru da pretraži direktorijum, u kome su i ostali fajlovi projekta, u potrazi za datotekom *“file”*.

FAZE U RAZVOJU PROGRAMA SA VIŠE FAJLOVA

Pri radu sa više fajlova kompajler prevodi svaki pojedinačni fajl, pri čemu se može desiti da prilikom prevođenja jednog fajla treba potražiti informaciju definisanu u drugom fajlu

U nastavku je dat klasičan primer faza u razvoju složenog C programa sa više fajlova, gde se koriste fajlovi standardne biblioteke i fajlovi kreirani od strane samih programera, Slika 4.1:



Slika 4.1 Primer složenog C programa sa više fajlova. Uključivanje sopstvenih i standardnih fajlova zaglavlja u program, za kojima sledi prevođenje i povezivanje programa u izvršnu verziju [7]

Poglavlje 5

Funkcije standardne biblioteke

TIPOVI MATEMATIČKIH FUNKCIJA

Standardne matematičke funkcije se nalaze u biblioteci <math.h> pa je potrebno je da se na početku programa naredbom # include ona uključi u program

U sastavu jezika C su definisane standardne biblioteke koje sadrže skup često korišćenih matematičkih funkcija. Da bi koristili neke od tih funkcija potrebno je da se na početku programa naredbom ***#include*** saopšti ime biblioteke u kojoj se funkcija nalazi. Standardne matematičke funkcije se nalaze u biblioteci <*math.h*>. U nastavku je naveden spisak nekih od često korišćenih funkcija:

- sin(x) - sinus
- cos(x) - cosinus
- tan(x) - tangens
- exp(x) - e^x
- log(x) - ln x
- log10(x) - $\log_{10} X$
- pow(x,y) - x^y
- sqrt(x) - kvadratni koren od x, x > 0
- fabs(x) - apsolutna vrednost od X
- ceil(x) - zaokrugljuje na prvi veći ceo broj
- floor(x) - zaokrugljuje na prvi manji ceo broj

U nastavku je dat primer određivanja površine trougla korišćenjem funkcije *pow* (eng. power) pomoću koje se računa kvadrat broja.

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

void main(void)
{
   double radius, area;
   printf("Unesi poluprecnik kruga: ");
   scanf("%lf",&radius);
   area = 3.14159 * pow(radius, 2);
   printf("Povrsina kruga je %lf\n",area);
}
```

Ulaz i izlaz prethodnog programa bi mogao biti:

```
Unesi poluprecnik kruga: 6
Povrsina kruga je113.097
```

GENERISANJE SLUČAJNIH BROJEVA

U tu svrhu se uglavnom koriste funkcije rand() i srand()

U nastavku ćemo obraditi jednu temu koja može da ima široku i interesantnu primenu, pogotovo u video igrama: generisanje slučajnih brojeva. U tu svrhu se uglavnom koriste funkcije *rand()* i *srand()*.

Funkcija *rand()* se koristi da bi se generisao slučajan broj. Prototip funkcije *rand* se nalazi u biblioteci *<stdlib.h>*. Sintaksa funkcije je:

```
int rand(void):
```

i ona vraća slučajan broj iz opsega od 0 do RAND_MAX. RAND_MAX: je konstanta čija podrazumevana vrednost varira među implementacijama ali je obično vrednosti 32767. Primer primene funkcije je u nastavku:

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

void main(void)
{
    // Ovaj program će kreirati istu sekvencu slucajnih
    // brojeva za svako sledeće pokretanje programa

    for(int i = 0; i<5; i++)
        printf(" %d ", rand());
}
```

Funkcija *rand()* generiše istu sekvencu brojeva iznova i iznova svaki put kada se ponovo startuje program. To možemo i videti na osnovu sledećih rezultata:

Izlaz 1:

```
453 1276 3425 89
```

Izlaz 2:

```
453 1276 3425 89
```

Izlaz 3:

```
453 1276 3425 89
```

Bacanje kockica.

Da bi demonstrirali kako radi funkcija *rand*, razvićemo program koji simulira 5 uzastopnih bacanja šestostrane kockice i prikazuje rezultat svakog bacanja. Da bi generisali proizvoljan ceo broj u intervalu 0 - 5, koristimo operator za određivanje ostatka pri deljenju (%) uz primenu funkcije *rand* :

```
rand() % 6
```

Ova operacija se zove ***skaliranje*** , a broj 6 predstavlja ***faktor skaliranja*** . Konačno, na rezultat dodajemo jedinicu i dobijamo broj u intervalu od 1 do 6.

FUNKCIJA SRAND()

Funkcija srand() postavlja početnu tačku za kreiranje sekvence, ili serija, pseudo-random celih brojeva.

Ako ne pozovemo *srand()* funkciju, sejač (seed) skupa slučajnih brojeva funkcije *rand()* je postavljen kao da smo pozvali *srand(1)* na početku programa. Svaka druga vrednost za seed postavlja generator na različitu početnu tačku. Sintaksa je:

```
void srand( unsigned seed );
//Seeds the pseudo-random number generator used by rand()
with the value seed.
```

Generator slučajnih brojeva treba da bude inicijalizovan samo jednom, pre nego što se prvi put pozove *rand()*, i uglavnom na početku programa. Nije poželjno da se stalno vrši inicijalizacija sejača (seed), ili da se reinicijalizuje sejač kada sledeći put poželite da generišete skup slučajnih brojeva.

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

void main(void)
{
    // Ovaj program će kreirati različitu sekvencu slučajnih
    // brojeva za svako sledeće pokretanje programa

    // KOristiti trenutno vreme za inicijalizaciju slučajnog generatora
    srand(time(0));

    for(int i = 0; i<5; i++)
        printf(" %d ", rand());
}
```

Uobičajena praksa je da se koristi rezultat funkcije *srand(time(0))* kao inicijalizator. Na ovaj način će funkcija *time()* svaki sledeći put vratiti različitu vrednost (koja je tipa ***time_t***) i stoga će se generisati različita sekvenca slučajnih brojeva za svako naredno izvršavanje programa.

S obzirom da smo primenili funkciju *srand*, rezultati izvršavanja navedenog programa biće:

Izlaz 1:

```
453 1432 325 89
```

Izlaz 2:

```
8976 21234 45 8975
```

Izlaz n:

```
563 9873 12321 24132
```

U kakvoj su vezi funkcije srand() i rand()?

srand() setuje (inicijalizuje) sejač koji će biti korišćen od strane funkcije *rand* za generisanje slučajnih brojeva. Ako ne pozovete *srand* pre prvog poziva funkcije *rand*, to je isto kao da ste pozvali *srand(1)* odnosno kao da ste setovali sejač (seed) na 1.

POSTUPAK ZA SKALIRANJE I POMERANJE SLUČAJNIH BROJEVA

Skaliranjem se definše širina opsega dok se pomeranjem postavlja početna pozicija opsega izgenerisanih slučajnih brojeva

Rekli smo da su brojevi koje generiše funkcija *rand()* u opsegu:

0 ≤ rand()≤ RAND_MAX

Takođe smo videli da sledeći kod simulira bacanje šestostrane kockice:

```
face = 1 + rand() % 6;
```

Ovaj iskaz uvek dodeljuje promenljivoj *face* celobrojnu vrednost u opsegu 1 ≤face ≤ 6. Širina ovog opsega (tj. broj uzastopnih celobrojnih vrednosti u opsegu) je 6, pri čemu je prvi broj jednak 1. Kao što možemo da vidimo, širina opsega je određena brojem koji se koristi da bi se skalirao *rand* korišćenjem operacije moduo, i početni broj iz opsega je jednak broju koji se dodaje na ***rand % 6***. Ovaj rezultat možemo da generalizujemo na sledeći način:

```
n = a + rand() % b;
```

gde je *a* vrednost pomeranja (što je jednako prvom broju u željenom opsegu uzastopnih brojeva) a *b* je faktor skaliranja (što je jednako širini željenog opsega uzastopnih celih brojeva).

U nastavku je kompletan primer:

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

```
void main( void )
{
    int i; /* brojac */
    unsigned seed; /* broj korišćen za inicijalizaciju generatora */

    printf( "Unesi seed: " );
    scanf( "%u", &seed ); /* %u za unsigned */

    srand( seed ); /* inicijalizuj generator slucajnih brojeva */

    for ( i = 1; i <= 10; i++ ) {

        /* izaberi slučajan broj od 1 do 6 i odštampaj ga */
        printf( "%10d", 1 + ( rand() % 6 ) );

        /* ako je counter deljiv sa 5, predji u novu liniju izlaza */
        if ( i % 5 == 0 ) {
            printf( "\n" );
        }
    }
}
```

Poglavlje 6

Vežbe

BROJ U OBRNUTOM PORETKU (10 MIN)

Invertovanje broja se izvodi korišćenjem operacija deljenja i nalaženja ostatka pri deljenju. Polazi se od cifre na desnoj strani i vrši izvlačenje jedne po jedne cifre iz broja, a zatim inverto

Napisati program koji unosi broj sa standardnog ulaza, formira broj sa ciframa u obrnutom poretku i ispisuje ga na standardni izlaz.

```
#include <stdio.h>
void main()
{
    int n,t=0;
    printf("Unesite broj\n");
    scanf("%d",&n);
    do
    {
        t=t*10+n%10;
        n/=10;
    }while(n);
    printf("Novi broj je %d\n", t);
}
```

Primer: Za broj n=1234 njegov inverzni je 4321.

Kako da izvučemo cifru 4 iz broja? Pa jednostavno primenimo 1234%10. Ok, ostaje nam da izvučemo i ostale cifre. Ali da bi smo izvukli cifru 3 primenom operacije % neophodno je da 3 bude na kraju broja. Zato početni broj 1234 delimo sa 10 i dobijamo n=123. Ako sada primenimo 123%10 rezultat će biti 3. Ako u petlji stalno primenjujemo operacije % i / moći ćemo da izvučemo sve cifre bilo kog broja (pa i onog sa deset i više cifara). Petlja će se prekinuti kada n bude jednako nula (naravno u poslednjem slučaju, imamo 1/10 = 0).

Drugi deo zadatka je da broj 4 koji je bio na kraju dovedemo na početak. Ako u tri ciklusa while petlje neki broj stalno množimo sa 10 dobićemo 4*10*10*10 = 4000. Slično treba da uradimo i za ostale cifre (2*10*10, 3 * 10, 1). Ovde koristimo i obrazac, takozvano Hornerovo pravilo, koje nam omogućava da predstavimo bilo koji broj:

$4321 = 4 * 10^3 + 3 * 10^2 + 2 * 10^1 + 1 * 10^0$

Drugi oblik koji je pogodniji za primenu u petlji je:

4321 = ((((4*10+3) * 10) + 2) * 10) + 1

i njega smo upravio i koristili u programu

PETLJE I KARAKTERI(10 MIN)

Pri radu sa karakterima moguće je koristiti funkcije getchar() i putchar(), za učitavanje odnosno štampu

Napisati program koji će tekst sa standardnog ulaza ispisati na standardni izlaz tako da se višestruke uzastopne pojave blanko znaka zamene jednim blankom (sažimanje).

```
#include <stdio.h>
#define GRANICA '0'
void main()
{
    int znak, preth;
    preth=GRANICA;
    while ((znak=getchar()) !=EOF)
    {
        if (znak !=' ' || preth != ' ')
            putchar(znak);
        preth=znak;
    }
}
```

Imamo while petlju koja učitava karakter po karakter sve dok se ne unese EOF (Ctrl+Z) karakter. Za uneti karakter znak proveravamo da li je praznina ili ne. Ukoliko su oba znak i preth praznine ništa se ne dešava. U suprotnom, ukoliko je samo znak praznina, ili je samo preth praznina štampa se znak

UPOTREBA OPERATORA BREAK U PETLJI (10 MIN)

Napisati program koji računa sumu unetih brojeva. Brojevi se unose redom sa tastature dok se ne unese negativan broj (koristiti petlju i break za prekid unosa brojeva)

```
// BreakDemo - Uneti niz brojeva.
// Nastavi da dodajes nove brojeve sumi
// sve dok korisnik ne unese negativan broj.
#include <stdio.h>
#include <stdlib.h>

int main()
{
 // input the loop count
 int accumulator = 0;
 printf("Ovaj program sumira vrednosti koje je uneo korisnik\n");
 printf("Prekinite petlju unosom negativnog broja\n");
```

```
 // loop "forever"
 for(;;)
 {
  // nastavi da unosis nove brojeve
  int value = 0;
  printf("Uneti sledeci broj: ");
  scanf("%d", &value);
  // ako je broj negativan...
  if (value < 0)
  {
  // ...onda izadji iz petlje
   break;
  }
 // ... u suprotnom dodaj broj na vrednost promenljive
 // accumulator
  accumulator = accumulator + value;
 }
 // sada kada smo izasli iz petlje
 // odstampaj akumuliranu vrednost sume
 printf("\nUkupna vrednost je %d\n", accumulator);

 // sačekaj neko vreme, da korisnik može da vidi rezultate
 // pre nego što se prekine program
 system("PAUSE");
 return 0;
}
```

Nakon opisa pravila korisniku (uneti negativan broj u cilju prekida, itd), program upada u ciklus koji izgleda kao beskonačna petlja. U toku te petlje se vrši unos brojeva sa tastature. Nakon unosa broja ispituje se da li broj zadovoljava izlazni kriterijum (da li je manji od nule). Ukoliko uneti broj nije negativan, program zaobilazi operator prekida ***break***, i dodaje unetu vrednost sumi (akomulira zbir). Nakon unosa negativnog broja i prekida izvršavanja ***for*** petlje, vrši se štampanje akumuliranog zbira na ekran.

ODREĐIVANJE PROSTIH BROJEVA (15 MIN)

Napisati program koji ispisuje prvih N prostih brojeva. Napisati funkciju koja određuje da li je broj prost

Rešenje:

Uključivanje biblioteka, deklaracija funkcija, i glavna main funkcija:

```
#include<stdio.h>
int prost(int);
void main()
{
    int n, i, br;
    printf("Unesite koliko prostih brojeva zelite da dobijete: \n");
    scanf("%d", &n);
```

```
    i = 0; br = 2;
    while(i < n)
    {
        if (prost(br))
        {
            printf("Broj %d je prost.\n", br);
            i++;
        }
        br++;
    }
}
```

Funkcija koja računa da li je broj prost:

```
int prost(int n)
{
 int prost,i;
 if (n==1) return 0;
 prost = (n%2!=0) || (n==2);
 i=3;
 while ((prost) && (i*i<=n))
 {
  prost = n%i != 0;
  i=i+2;
 }
 return prost;
}
```

Kao što znamo broj je prost ako je deljiv samo sa jedinicom i sa samim sobom, što upravo i proverava prethodna funkcija.

Poglavlje 7

Zadaci za samostalan rad

ZADACI ZA SAMOSTALNO VEŽBANJE

Na osnovu materijala sa predavanja i vežbi uraditi samostalno sledeće zadatke:

Zadatak 1. Napisati program koji ispisuje sumu svih neparnih brojeva manjih od 1000. (5 min)

Zadatak 2. Napisati program koji igra sa korisnikom igru ''Papir kamen makaze''. Koristite ugnježdene switch strukture. Pokušajte da napravite program tako da ne možete predvideti šta će računar izabrati (papir, kamen ili makaze). (10 min)

Zadatak 3. Napisati C program koji unosi ceo broj sa standardnog ulaza i ispisuje da li je uneti broj negativan, pozitivan ili jednak nuli. (10 min)

Zadatak 4. Napisati program koji izračunava apsolutnu vrednost celog broja. (10 min)

Zadatak 5. Napisati program koji određuje srednju vrednost N pozitivnih realnih brojeva – while petlja i for petlja. (10 min)

DODATNI ZADACI ZA SAMOSTALNI RAD

Koristeći materijal sa predavanja i vežbi, samostalno uraditi sledeće zadatke

Zadatak 1. Napisati program koji korišćenjem ugnježdene while petlje štampa po 10 X karaktera u 5 vrsta. (10 min)
Zadatak 2. Napisati program koji će izrazunati sume kvadrata prvih 5, odnosno prvih 25 brojeva, korišćenjem funkcije koja računa sumu kvadrata brojeva do N. Obe sume ispisati u zasebnim linijama. (10 min)
Zadatak 3. Napravite metodu koja će za unetu dužinu stranica pravougaonika prikazivati površinu pravougaonika sa duplo dužim stranicama. Metodi treba proslediti unete dužine iz prvog primera iz prethodne sekcije po adresi, i ona treba te vrednosti da duplira, i nakon toga se ponovo poziva funkcija iz prvog zadatka. (10 min)
Zadatak 4. Napisati funkciju u koju se unose dva broja, broj koji se stepenuje i stepen, a ona vraća rezultat stepenovanja. (10 min)
Zadatak 5. Kreirati C aplikaciju koja za unetu dužinu stranica pravougaonika ispisuje njegovu površinu i obim. Razdvojite izračunavanje u posebne funkcije. (10 min)

Poglavlje 8

Domaći zadatak

PRAVILA ZA DOMAĆI ZADATAK

Detaljno proučiti pravila za izradu domaćih zadataka

Svaki student dobija od asistenta sopstvenu kombinaciju domaćeg zadatka.

Onlajn studenti bi trebalo mejlom da se najave, kada budu želeli da krenu sa radom na predmetu i prikupljanjem predispitnih obaveza.

Odgovarajući Visual Studio (NetBeans ili CodeBlocks) projekat koji predstavlja rešenje domaćeg zadatka smestiti u folder CS130-DZ02-Ime-Prezime-BrojIndeksa. Zipovani folder CS130-DZ02-Ime-Prezime-BrojIndeksa poslati predmetnom asistentu (lazar.mrkela@metropolitan.ac.rs) u mejlu sa naslovom (subject)CS130-DZ02, inače se neće računati.

Studenti iz Niša predispitne obaveze predaju asistentima u Nišu (sofija.ilic@metropolitan.ac.rs, mihajlo.vukadinovic@metropolitan.ac.rs, i uros.lazarevic@metropolitan.ac.rs).

Student tradicionalne nastave ima 7 dana, od dana kada je dobio mail sa domaćim zadatkom, da uradi i pošalje rešenje za maksimalan broj poena. Ukoliko student pošalje domaći nakon tog roka, najviše može da ostvari 50% od maksimalnog broja poena.

Studenti onlajn nastave imaju rok da predaju rešene domaće zadatke 10 dana pre termina ispita u ispitnom roku u kome polažu CS130 C/C++ programski jezik.

Vreme izrade: 1,5h.

Zaključak

REZIME

Na osnovu svega obrađenog možemo da izvedemo sledeći zaključak:

Iskazi ***if***, ***if-else***, ***if-else if -else***, i ***switch*** se koriste da bi se struktuirao kod tako da se različiti segmenti koda izvršavaju u zavisnosti od donete odluke odnosno odgovarajućeg izbora.

Zatim je pokazano kako je moguće koristiti ***switch*** operator kao alternativa nekom ***if-else*** ili ***if-else if-else*** iskazu u programima gde se vrednosti logičkih izraza određuju korišćenjem logičkih operatora.

Opisano je korišćenje ***for, while***, i ***do-while*** petlje. Takođe su opisane ugnježdene petlje.

Na kraju je opisano je korišćenje operatora ***break*** u cilju prevremenog prekida petlje, kao i operatora ***continue*** u cilju preskakanja jedne iteracije petlje.

U C-u je moguće deo koda smestiti u programske celije koje se zovu funkcije. Informacije se mogu proslediti funkciji korišćenjem argumenata, i to se može u C-u uraditi po vrednosti i adresi.

Dok se argumenti koriste da se neka vrednost prosledi funkciji, povratne vrednosti (return value) se koriste da se rezultat izvršavanja funkcije vrati u blok iz kog je funkcija pozvana. Iz funkcije kao rezultat se može vratiti najviše jedna vrednost. Ukoliko funkcija ne vraća rezultat onda je ona tipa ***void***.

REFERENCE

Korišćena literatura:

[1] Jeff Kent , C++: Demystified: A Self-Teaching Guide, McGraw-Hill/Osborne, 2004.

[2] Nenad Filipović, Programski jezik C, Tehnički fakulet u Čačku, Univerzitet u Kragujevcu, 2003.

[3] Milan Čabarkapa, C - Osnovi programiranja, Krug, Beograd, 2003.

[4] Paul J Deitel, Harvey Deitel, C - How to program, 7th edition, Pearson, 2013.

[5] http://www.tutorialspoint.com/cprogramming/index.htm

[6] http://www.codingunit.com/category/c-tutorials