UNIVERZITET METROPOLITAN BEOGRAD

CS130 - C/C++ PROGRAMSKI JEZIK

Pokazivači

Lekcija 04

PRIRUČNIK ZA STUDENTE

CS130 - C/C++ PROGRAMSKI JEZIK

Lekcija 04

POKAZIVAČI

UVOD

Ova lekcija treba da ostvari sledeće ciljeve:

Pokazivače prati glas da su materija koja se veoma teško uči. Međutim, ukoliko odvojite dovoljno vremena u cilju njihovog razumevanja, pokazivači nisu toliko teški za učenje. Bilo da je teško ili ne, razumevanje rada pokazivača i sposobnost njihove primene u C i C++ jeziku je od izuzetnog značaja. Naime, neke C/C++ zadatke ćete mnogo lakše rešiti primenom pokazivača, dok neke druge zadatke, kao što je na primer dinamička alokacija memorije, jednostavno ne možete rešiti ni na jedan drugi način.

Da se prisetimo osnovnih operatora koje smo uveli u Lekciji L01 koje koristimo pri radu sa pokazivačima, Slika 8.1:

*	operator indirektnog (posrednog) pristupa
&	adresa

Primena: operator a

Slika-9.1: Operatori za rad sa pokazivačima [2]

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

Poglavlje 1

Osnovi pokazivača

OSVRT NA POKAZIVAČE

Svaka promenljiva je ustvari smeštena na nekoj memorijskoj lokaciji, a svaka memorijska lokacija ima svoju adresu koja je određena korišćenjem operatora &

Da se osvrnemo na sve što smo do sada naučili o pokazivačima. Opšti oblik deklarisanja pokazivača je:

```
tip *ime_pokazivača;
```

što znači: ***"ime_pokazivač je pokazivač na tip"*** . U ovoj deklaraciji ***tip*** označava tip podatka na koji pokazuje pokazivač *ime_pokazivača*.

U nastavku su primeri deklaracije pokazivača na različite tipove podataka:

```
int *ip;    /* pokazivac na ceo broj */
double *dp; /* pokazivac na double */
```

U narednom primeru se vrednost promenljive *num* menja 2 puta.

```
#include <stdio.h>
int main ()
{
   int num = 5;
   int* iPtr = &num;
   printf("Vrednost od num je %d\n", num);
   num = 10;
   printf("Vrednost od num nakon num = 10 je %d\n", num);
   *iPtr = 15;
   printf("Vrednost od num nakon *iPtr = 15 je %d\n", num);
   return 0;
}
```

Kod prve izmene se direktnom dodelom menja vrednost promenljivoj *num*, tako da je *num = 10*. Međutim druga promena je izvršena na novi način, korišćenjem indirektnog operatora:

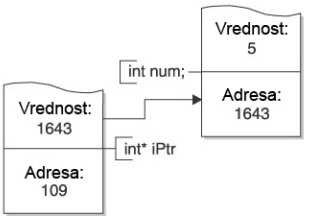
```
*iPtr = 15;
```

Adresu neke promenljive možemo odrediti tako što uz naziv promenljive primenimo operator *&* (ampersand):

```
int  var1;
printf("Adresa promenljive var1: %x\n", &var1);
```

Naredni primer opisuje korišćenje adresnog operatora da bi se adresa promenljive dodelila pokazivaču. Ovaj program takođe pokazuje da je vrednost pokazivača ista kao i adresa na koju pokazivač pokazuje, Slika 1.1.

```
int num = 5;
int* iPtr = &num;
printf("Adresa od x koriscenjem &num je %x\n",
        &num);
```



Slika 1.1 Pokazivač na adresu na kojoj se nalazi celobrojna vrednost [1]

NUL (NULL) POKAZIVAČI

Adresa označena sa 0 se često dodeljuje pokazivaču pri inicijalizaciji jer je to adresa rezervisana od strane sistema i nema opasnosti da loša upotreba pokazivača ugrozi vaš program

Uvek je dobra praksa da se pokazivaču dodeli *NULL* vrednost kada mu nije dodeljena konkretna adresa, odnosno kada on nema adresu na koju pokazuje. Ovaj pokazivač se obično naziva nul pokazivač. *NULL* pokazivač je konstanta čija je vrednost nula (*0*) i koja je definisana u nekoliko standardnih biblioteka C jezika. U nastavku je primer deklaracije pokazivača čija je vrednost *NULL*:

```
#include <stdio.h>
int main ()
{
   int  *ptr = NULL;

   printf("Vrednost od ptr je : %x\n", ptr  );

   return 0;
}
```

Rezultat prethodnog programa bi bio:

```
Vrednost od ptr je 0
```

Po ustaljenoj konvenciji, podrazumeva se da ako pokazivač pokazuje na *NULL* on ustvari ne pokazuje ni na kakvu adresu. Da bi se proverilo da li pokazivač ima dodeljenu vrednost ili ne pokazuje ni na šta, mogu se koristiti sledeći izrazi:

```
if(ptr)     /* izvrsava se ako p nije jednako null */
if(!ptr)    /* izvrsava se ako je p jednako null */
```

UPOTREBA POKAZIVAČA KOD NIZOVA

Korišćenjem pokazivača moguće je pristupiti članovima niza. Treba samo voditi računa pri deklaraciji da tip podatka niza bude isti tipu podatka pokazivača

Videli smo u prošloj lekciji da se preko pokazivača može posredno pristupati i članovima niza. U postupku deklarisanja vodi se računa da je tip podataka u nizu istovremeno i tip na koji pokazuje pokazivač tj. opšta sintaksa je:

```
tip ime_niza[];
tip *ime_pointera;
```

Inicijalizacija pokazivača se tada vrši bez upotrebe adresnog operatora & tj.

```
ime_pointera = ime_niza;
```

Direktno pristupanje članovima niza izvodi se uobičajeno preko indeksa. Ako je *i* indeks, onda je:

```
ime_niza[i]  - i+1 član zato što indeksi idu od 0.
```

Istom članu niza se može posredno pristupiti preko pokazivača uvećanog za indeks *i* :

```
*(ime_pointera + i) ili *(ime_niza + i)
```

Takođe, pokazivač se može inicijalizovati da pokazuje na član sa indeksom *i*:

```
ime_pointera = &ime_niza[i];
```

ili:

```
ime_pointera = ime_niza + i;
```

POKAZIVAČI, NIZOVI I ADRESE U MEMORIJI

Ime niza je istovremeno i konatantan pokazivač na prvi element niza

Neka je deklarisan sledeći niz:

```
int testScore[MAX] = {4, 7, 1};
```

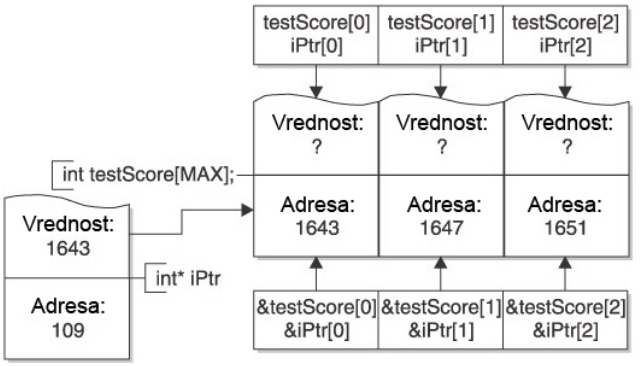
i neka je deklarisana i inicijalizovana promenljiva *iPtr* kao pokazivač na niz *testScore*, dodeljivanjem adrese niza *testScore* pokazivačkoj promenljivoj *iPtr*:

```
int* iPtr = testScore;
```

Moguće je napisati sledeći deo koda korišćenjem pokazivača na niz:

```
for (int i = 0; i < MAX; i++)
{
  printf("Adresa indeksa %d",i)
  printf(" niza iPtr je %x\n", & iPtr[i]);
  printf("Adresa indeksa %d", i)
  printf(" niza iPtr je %d\n", iPtr[i]);
}
```

Stoga će, kao što je prikazano na Slici 1.2, *iPtr[2]* i *testScore[2]* imati istu vrednost.



Slika 1.2 Promenljivi (iPtr) i konstantni (testScore) pokazivači korišćeni za pristupanje članovima niza [1]

Korišćenje izraza *(testScore + 4) je legalan način da se pristupi podatku 5. člana niza odnosno članu testScore[4]. To nam daje mogućnost da pristupamo članovima niza na sledeći način:

```
printf( "Vrednosti u nizu, preko pokazivaca\n");
for ( i = 0; i < 5; i++ )
{
    printf("*(iPtr + %d) : %f\n",  i, *(iPtr + i) );
}

printf( "Vrednosti u nizu, preko testScore kao adrese\n");
for ( i = 0; i < 5; i++ )
{
    printf("*(testScore + %d) : %f\n",  i, *(testScore + i) );
}
```

POKAZIVAČKA ARITMETIKA

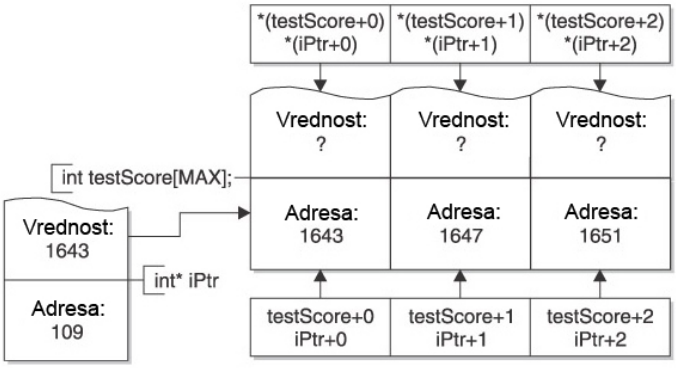
Uvećanjem pokazivača za jedan se ustvari dodaje pokazivaču vrednost veličine tipa podatka (u bajtovima) na koji pokazuje

Postoje četiri aritmetička operatora koja se mogu primeniti na pokazivačima: ++, --, + i -.

U nastavku je dat primer gde se vrši inkrementiranje pokazivačke promenljive u cilju pristupa susednim elementima niza:

```
#include <stdio.h>
#define MAX 3

int main ()
{
   int i;
   int testScore[MAX] = {4, 7, 1};
   int* iPtr = testScore;
   for (i = 0; i < MAX; i++, iPtr++)
   {
    printf("Adresa od indeksa %d niza testScore je %d\n",
    i,iPtr);
    printf("Vrednost na indeksu %d niza testScore je %d \n",
    i,*iPtr);
   }
   return 0;
}
```

Kao što se može videti na Slici 1.3, uvećanjem pokazivača *iPtr + 1* se na adresu na koju pokazuje *ptr* dodaje vrednost od 4 bajta, a isto će se desiti ako uradimo *testScore + 1*.



Slika 1.3 Efekat inkrementiranja pokazivača za 1 [1]

Stoga je 2. elementu niza moguće pristupiti na jedan od 4 sledeća načina:

```
testScore[1];
*(testScore + 1);
iPtr[1];
*(iPtr + 1);
```

Iste pretpostavke važe i u slučaju kada se vrši dekrementiranje pokazivača (umanjenje vrednosti za jedan), pri čemu se adresa na koju pokazuje umanjuje za onoliko bajtova kolika je vrednost tipa podatka na koji pokazuje.

RAZLIKE IZMEĐU NIZOVA I POKAZIVAČA

Ime niza kao konstantan pokazivač ne sme se koristiti u procesu inkrementiranja i dekrementiranja

Pokazivači i nizovi nisu do kraja u korelaciji što ilustruje sledeći primer:

```
#include <stdio.h>
#define MAX 3

int main ()
{
   int  var[MAX] = {10, 100, 200};
   int i;
   for (i = 0; i < MAX; i++)
   {
    *var = i;   // Ovo je korektna sintaksa
    var++;      // Ovo je nekorektno.
   }
   return 0;
}
```

Kao što se vidi iz primera, dopušteno je da se koristi pokazivački operator * nad nizom sa imenom *var* ali nije dozvoljeno da se modifikuje *var* vrednost. Razlog je taj što se ime niza *var* tretira kao konstantan pokazivač, i ne može mu se menjati vrednost korišćenjem operatora dodele (u ovom slučaju operator inkrementiranja).

Bez obzira što je ime niza ustvari konstantan pokazivač na prvi član niza, on ipak može biti korišćen u izrazu u pokazivačkoj notaciji sve dok mu se vrednost ne menja. Na primer, u nastavku je dat validan iskaz koji članu *var[2]* dodeljuje vrednost 500:

```
*(var + 2) = 500;
```

Prethodni iskaz je validan i program će biti uspešno iskompajliran jer promenljiva *var* u ovom slučaju nije modifikovana.

Poglavlje 2

Pokazivači i višedimenzionalni nizovi

NIZOVI POKAZIVAČA

C podržava korišćenje nizova pokazivača. Najčešće se nizovi pokazivača koriste kao zamena za višedimenzionalne nizove

Mogu postojati situacije gde je neophodno da imamo niz koji će kao vrednosti imati pokazivače na ***int***, ***char*** ili neki drugi tip podatka. Deklaracija niza pokazivača na celobrojne vrednosti može imati sledeći oblik:

```
int *ptr[MAX];
```

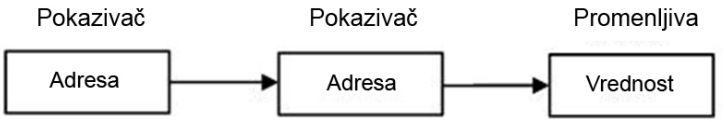
U prethodnom iskazu je deklarisana promenljiva *ptr* kao jedan niz koji u sebi sadrži *MAX* broj pokazivača na celobrojnu vrednost. To znači da svaki element niza *ptr* sadrži pokazivač na ceo broj. U nastavku je dat primer koji demonstrira korišćenje niza pokazivača, pri čemu se nizu pokazivača dodeljuju adrese odgovarajućih članova celobrojnog niza *var*:

```
#include <stdio.h>
#define MAX 3

int main ()
{
    int  var[] = {10, 100, 200};
    int i, *ptr[MAX];

    for ( i = 0; i < MAX; i++)
    {
        ptr[i] = &var[i]; /* dodeliti adresu celobrojnog podatka. */
    }
    for ( i = 0; i < MAX; i++)
    {
        printf("Vrednost od var[%d] = %d\n", i, *ptr[i] );
    }
    return 0;
}
```

POKAZIVAČI NA POKAZIVAČE

Pokazivač na pokazivač se može posmatrati kao lanac pokazivača. U tom slučaju prvi pokazivač sadrži adresu drugog pokazivača, a drugi pokazivač sadrži adresu neke promenljive

Pokazivač na pokazivač se posmatrati kao lanac pokazivača. Pokazivač sadrži adresu promenljive. Kada definišemo pokazivač na pokazivač, to u stvari znači da prvi pokazivač sadrži adresu drugog pokazivača, a drugi pokazivač pokazuje na lokaciju na kojoj se nalazi stvarna promenljiva, što je prikazano na Slici 2.1:



Slika 2.1 Lanac pokazivača kao pokazivač na pokazivač [6]

Promenljiva koja je pokazivač na drugi pokazivač mora biti deklarisana kao takva. Ovo se ostvaruje tako što se postavi dodatna zvezdica (asterisk) ispred naziva pokazivačke promenljive. Na primer, ako želimo da deklarišemo pokazivač na pokazivač na tip ***int***, to radimo na sledeći način:

```
int **var;
```

Ako želimo u prethodnom slučaju da indirektno pristupimo konkretnoj vrednosti korišćenjem pokazivača na pokazivač, pristup promenljivoj zahteva da se operator * primeni dva puta, kao što je to opisano u narednom primeru:

```
#include <stdio.h>

int main ()
{
   int  var;
   int  *ptr;
   int  **pptr;

   var = 3000;

   /* uzeti adresu od var */
   ptr = &var;

   /* uzeti adresu od ptr koriscenjem operatora & */
   pptr = &ptr;

   /* pristupi vrednosti preko pptr */
   printf("Vrednost od var = %d\n", var );
   printf("Vrednost dostupna preko *ptr = %d\n", *ptr );
   printf("Vrednost dostupna preko **pptr = %d\n", **pptr);
```

```
   return 0;
}
```

Rezultat prethodnog programa je:

```
Vrednost od var = 3000
Vrednost dostupna preko *ptr = 3000
Vrednost dostupna preko **pptr = 3000
```

POKAZIVAČI NA VIŠEDIMENZIONALNE NIZOVE

Višedimenzionalni niz je u memoriji smešten kao sekvenca tako da je pokazivačem na njega moguće pristupiti svim njegovim elementima

Dvodimenzionalni nizovi su u memoriji smešteni po vrstama pri čemu zauzimaju susedne memorijske lokacije. U slučaju da imamo 2D niz koji deklarišemo na sledeći način:

```
int a[2][3];
```

Njegovi elementi će u memoriji biti raspoređeni u sledećem poretku:

```
a[0][0] a[0][1] a[1][0] a[1][1] a[2][0] a[2][1]
```

što znači da se prvo menja desni indeks pa tek onda levi. Deklarišimo sada pokazivačku promenljivu na tip ***int***, na sledeći način:

```
int *pointer1;
```

i izvršimo dodelu:

```
pointer1 = a;
```

Prethodnom dodelom se definiše pokazivačka promenljiva *pointer1* da pokazuje na element 2D niza *a* koji se nalazi u nultoj vrsti i nultoj koloni.

Ono što treba znati kod 1D i 2D nizova je to da ime niza predstavlja pokazivač (adresu) prvog elementa niza, tj. važi jednakost:

```
pointer1 = &a[0][0];
```

Na osnovu prethodnog možemo da zaključimo da *pointer1+1* pokazuje na drugi element 2D niza odnosno na *a[0][1]*, i tako redom, pa stoga važe sledeće jednakosti:

```
pointer1    = &a[0][0];
pointer1 +1 = &a[0][1];
pointer1 +2 = &a[1][0];
pointer1 +3 = &a[1][1];
pointer1 +4 = &a[2][0];
pointer1 +5 = &a[2][1];
```

U nastavku su dati još neki primeri korišćenja pokazivača na dvodimenzionalne nizove:

```
int myMatrix[ ][ ] 2 4 = { {1,2,3,4} , {5,6,7,8} };
// Indeksiranje: myMatrix[i][j] je isto kao i
*(myMatrix[i] + j)
(*(myMatrix + i))[j]
*((*(myMatrix + i)) + j)
*(&myMatrix[0][0] + 4*i + j)
```

UPOTREBA POKAZIVAČA NA 2D NIZOVE

Ako je 2D niz u C-u definisan kao a[n][n] onda se a[i] može posmatrati kao pokazivač na i-tu vrstu 2D niza

Pošto se 2D niz opisuje kao niz nizova, to je u primeru sa nizom *a[2][3]*, *a* u stvari ime dvodimenzionalnog niza, dok su *a[0]*, *a[1]* i *a[2]* u stvari imena jednodimenzionalnih nizova koji predstavljaju vrste matrice. Stoga možemo da napišemo sledeće jednakosti:

```
a[0] = &a[0][0];
a[1] = &a[1][0];
a[2] = &a[2][0];
```

U narednom primeru imamo matricu *a* koju prosleđujemo funkciji koja računa aritmetičku sredinu vrsta.

```
#include <stdio.h>
float sredina(float a[], int n)
{
    int i;
    float suma = 0;
    for (i = 0; i < n; i++)
        suma += a[i];
    suma /= n;
    return suma;
}

void main()
{
    float a[2][3] = {1.0, 2.0, 3.0, 4.0, 5.0, 6.0};
    int i;
    for (i = 0; i < 2; i++)
    {
        printf("Aritmetička sredina vrste %d je %lf\n",
            i, sredina(a[i],3));
    }
}
```

Ukoliko je 2D niz parametar funkcije pri njenom pozivu je dovoljno da se kao stvarni argument navede ime niza jer ono predstavlja adresu njegovog nultog elementa.

Koristan video o primeni pokazivača na 2D nizove možete naći na sledećem linku:

https://www.youtube.com/watch?v=sHcnvZA2u88

POREĐENJE POKAZIVAČA I VIŠEDIMENZIONALNIH NIZOVA

Ključna prednost niza pokazivača nad dvodimenzionalnim nizom je činjenica da vrste na koje pokazuju ovi pokazivači mogu biti različite dužine

Razmotrimo razlike između dvodimenzionalnih nizova i nizova pokazivača. Ako su date deklaracije

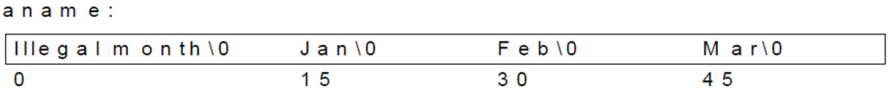
```
int a[10][20];
int *b[10];
```

tada su i *a[3][4]* i *b[3][4]* sintaksno ispravna referisanja na pojedinačni ***int***. Ali *a* je pravi dvodimenzionalni niz: 200 lokacija za podatak tipa ***int*** je rezervisano i uobičajena računica *20 * v + k* se koristi da bi se pristupilo elementu *a[v][k]*. Za niz *b*, međutim, deklaracija alocira samo 10 pokazivača i ne inicijalizuje ih - inicijalizacija se mora izvršiti eksplicitno, bilo statički ili dinamički. Pod pretpostavkom da svaki element niza *b* zaista pokazuje na niz od 20 elemenata, u memoriji će biti 200 lokacija za podatak *a* tipa ***int*** i još dodatno 10 lokacija za pokazivače. Ključna prednost niza pokazivača nad dvodimenzionalnim nizom je činjenica da vrste na koje pokazuju ovi pokazivači mogu biti različite dužine. Tako, svaki element niza *b* ne mora da pokazuje na niz od 20 elemenata - neki mogu da pokazuju na niz koji ima 2 elemenata, neki na niz od 50 elemenata, a neki mogu da budu NULL i da ne pokazuju nigde.

Razmotrimo primer niza koji treba da sadrži imena meseci. Jedno rešenje je zasnovano na dvodimenzionalnom nizu.

```
char aname[][15] = { "Illegal month", "Jan", "Feb", "Mar" };
```

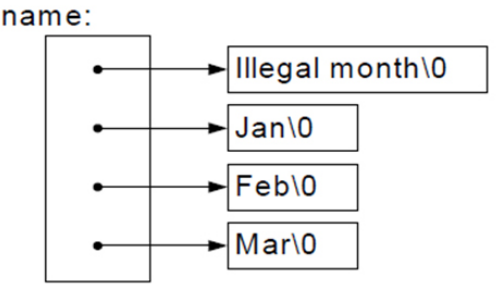
U memoriji računara ovaj niz može biti predstavljen kao (Slika 2.2):



Slika 2.2 Grafička ilustracija niza aname [4]

Pošto meseci imaju imena različite dužine, bolje rešenje je napraviti niz pokazivača na karaktere i inicijalizovati ga da pokazuje na konstantne nizove karaktera, smeštene u segmentu podataka (primetimo da nije potrebno navesti broj elemenata niza pošto je izvršena inicijalizacija) kao na Slici 2.3.

```
char *name[] = { "Illegal month", "Jan", "Feb", "Mar" };
```



Slika 2.3 Grafička ilustracija pokazivača name na niz stringova [4]

Poglavlje 3

Pokazivači i funkcije

POKAZIVAČI KAO ARGUMENTI FUNKCIJE

Izmena vrednosti koja je funkciji prosledjena preko pokazivača se reflektuje na stvarni argument odnosno promenljivu u bloku odakle je funkcija pozvana

Sintaksa na primeru prototipa funkcije je:

```
tip ime_funkcije(tip *ime_pointera);
```

U narednom primeru kao argument prosleđujemo funkciji pokazivač na tip ***unsigned long***, i menjamo vrednost u okviru funkcije koja se reflektuje na stvarni argument odnosno promenljivu u bloku odakle je funkcija pozvana:

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

void getSeconds(unsigned long *par);

int main ()
{
   unsigned long sec;
   getSeconds( &sec );

   printf("Broj sekundi je: %ld\n", sec );
   return 0;
}

void getSeconds(unsigned long *par)
{
   *par = time( NULL );
   return;
}
```

U lekciji o funkcijama smo videli odgovarajući primer korišćenja pokazivača.

Funkcija osim pokazivača na primitivne vrednosti, može i da prihvati ime niza u pokazivačkoj notaciji. U nastavku je dat primer gde funkciji prosleđujemo ime niza, pri čemu je u listi argumenata funkcije niz predstavljen kao pokazivač:

```
#include <stdio.h>
double getAverage(int *arr, int size);
void main ()
{
    int balance[5] = {1000, 2, 3, 17, 50};
    double avg;

    avg = getAverage( balance, 5 ) ;
    printf("Srednja vrednost je : %f\n", avg );
}

double getAverage(int *arr, int size)
{
    int   i, sum = 0;
    double avg;
    for (i = 0; i < size; ++i)
    {
        sum += arr[i];
    }
    avg = (double)sum / size;
    return avg;
}
```

POKAZIVAČI NA FUNKCIJE

Korišćenjem pokazivača na funkcije programeru je omogućeno da funkcije prosleđuje kao argumente drugih funkcija ili da im pristupa kao članovima niza u petlji

Pokazivači na funkcije pružaju veliku fleksibilnost u programiranju. Programeru je na taj način omogućeno da npr. funkcije prosleđuje kao parametre drugih funkcija ili da im pristupa kao članovima niza u petlji. Ovo se može videti iz narednog primera. Opšti oblik sintakse deklaracije je (tip odgovara tipu funkcije na koju pointer pokazuje):

```
tip (*ime_pointera)(tipovi_parametara);
```

U nastavku je dat konkretan primer deklaracije i korišćenja pokazivača na funkciju. Deklaracija je data na sledeći način:

```
double (*funcPtr)(double, double);
```

Prethodnom deklaracijom definisan je pokazivač na tip funkcije koja ima dva parametra tipa ***double***, i čija je povratna vrednosti takođe ***double***. Pokazivač na funkciju se nalazi u okviru zagrada čije je prisustvo obavezno. Bez navođenja zagrada koje uokviruju **funcPtr*, deklaracija bi imala sledeći izgled:

```
double *funcPtr(double, double);
```

što znači da bi imali prototip funkcije čija je vrednost pokazivač. Kad god je to neophodno, moguće je ime funkcije implicitno konvertovati u pokazivač na funkciju.

Stoga sledećim izrazima dodeljujemo adresu standardne funkcije *pow* pokazivaču označenom sa *funcPtr* pa zato možemo da pozivamo funkciju korišćenjem pokazivača:

```
double result;
// Neka funcPtr pokazuje na funkciju pow( ).
// Izraz *funcPtr sada poziva funkciju
// fukciju pow( ).
funcPtr = pow;

// Poziva funkciju koja je referencirana preko funcPtr.
result = (*funcPtr)( 1.5, 2.0 );
// Isti poziv funkcije, ali uz drugaciju sintaksu.
result = funcPtr( 1.5, 2.0 );
```

Kao što ilustruje poslednja linija prethodnog koda, kada se funkcija pozove korišćenjem pokazivača, ne mora da se navede indirektni operator (*) s obzirom da je levi operand pozivnog operatora funkcije tipa ***"pokazivač na funkciju"*** .

Moguće je pokazivače na funkcije smestiti u niz, a zatim pozvati funkciju korišćenjem indeksne notacije niza. Na primer, drajver za tastaturu može da koristi tabelu pokazivača na funkcije gde indeksi tabele predstavljaju odgovarajuću numeričku dugmad. Kada korisnik pritisne neko dugme iz programa se pozove odgovarajuća funkcija.

NIZ POKAZIVAČA NA FUNKCIJE

Niz pokazivača na funkcije omogućava jedan vid preklapanja (overloading) funkcija koji nije standardom podržan u C-u

Naredni program od korisnika zahteva da unese dva broja a zatim izvršava prosta izračunavanja nad njima.

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

double Add( double x, double y ) { return x + y; }
double Sub( double x, double y ) { return x - y; }
double Mul( double x, double y ) { return x * y; }
double Div( double x, double y ) { return x / y; }

double (*funcTable[5])(double, double)
        = { Add, Sub, Mul, Div, pow };

char *msgTable[5] = { "Zbir", "Razlika", "Proizvod", "Deljenje", "Stepenovanje" };

void main( )
{
```

```
  int i;
  double x = 0, y = 0;

  printf( "Uneti dva operanda:\n" );
  if ( scanf( "%lf %lf", &x, &y ) != 2 )
    printf( "Pogresan unos.\n" );

  for ( i = 0; i < 5; ++i )
    printf("%10s: %6.2f\n", msgTable[i], funcTable[i](x, y));
}
```

Matematičke funkcije su pozvane korišćenjem pokazivača koji su elementi niza *funcTable*.

Izraz *funcTable[i](x,y)* poziva funkciju čija adresa je smeštena u pokazivaču *funcTable[i]*. Već smo rekli da ime niza i okvirne zagrade *[ ]* ne moraju biti navedeni u okviru zagrada*()* jer pozivni operator funkcije *( )* i operator indeksiranja niza *[ ]* imaju veću prednost izvršavanja.

Još jednom da napomenemo, kompleksni tipovi kao što su nizovi pokazivača na funkcije su pogodniji za upotrebu u slučaju da se definiše prostiji tip korišćenjem ključne reči ***typedef***. Na primer, moguće je definisati niz *funcTable* na sledeći način:

```
// Tip funkcije je sada imenovan
// kao func_t.
typedef double func_t( double, double );

func_t *funcTable[5] = { Add, Sub, Mul, Div, pow };
```

VREDNOST FUNKCIJE JE POKAZIVAČ

Vrednost funkcije u ovom slučaju nikako ne sme da bude adresa lokalne promenljive definisane unutar bloka, osim ako je ta promenljiva deklarisana kao static

Sintaksa za prototip funkcije koja kao rezultat vraća pokazivač je:

```
tip *ime_funkcije(tipovi_parametara);
```

Na sličan način kao što se pokazivači prosleđuju funkciji, moguće je vratiti pokazivač iz funkcije kao vrednost funkcije.

Međutim, nije nikako dobra ideja da se kao rezultat funkcije vrati adresa lokalne promenljive u blok odakle je funkcija pozvana. Jedini način da se adresa lokalne promenljive vrati u glavni program je da se lokalna promenljiva deklariše kao statička ***static*** promenljiva.

Pogledajmo sada sledeću funkciju u kojoj se generiše 10 slučajnih brojeva a zatim se ti brojevi preko imena niza, koji predstavlja pokazivač na prvi element niza, vraćaju iz funkcije u blok odakle je funkcija pozvana. Naredni kod radi jer bez obzira što je oblast promenljive u okviru funkcije u kojoj je deklarisana, životni vek je korišćenjem ***static*** produžen sve dok se izvršava program.

Stoga, pokazivač vraćen u glavni program preko imena funkcije *getRandom*, pokazuje na lokalni niz koji živi, pošto je deklarisan kao ***static***, i nakon što je završen poziv funkcije *getRandom* i izvršen povratak u *main* funkciju.

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

int * getRandom( )
{
    static int  r[10];
    int i;

    srand( (unsigned)time( NULL ) );
    for ( i = 0; i < 10; ++i)
    {
        r[i] = rand();
        printf("%d\n", r[i] );
    }

    return r;
}

void main ()
{
    int *p;
    int i;

    p = getRandom();
    for ( i = 0; i < 10; i++ )
    {
        printf("*(p + [%d]) : %d\n", i, *(p + i) );
    }
}
```

Poglavlje 4

Pokazivači i stringovi

UPOTREBA POKAZIVAČA NA STRINGOVE

Pokazivači na C stringove rade po istom principu kao i pokazivači na nizove opštih brojnih vrednosti

Stringovi predstavljaju nizove karaktera, a na kraju tog niza se nalazi karakter binarne nule odnosno nul karakter. Pokazivači na C stringove, s obzirom da su to ustvari nizovi karaktera, rade po istom principu kao i pokazivači na nizove opštih brojnih vrednosti. U nastavku je dat jedan primer gde se koriste nizovi karaktera i pokazivači:

```
#include <stdio.h>
char stringA[40] = "Ovo je demonstracioni C-string";
char stringB[40];
void main(void)
{
    char *pA; /* prvi pokazivač na karakter*/
    char *pB; /* drugi pokazivač na karakter*/

    puts(stringA); /*prikaz stringa na ekranu*/
    pA = stringA;  /*pokazivač pA pokazuje na string A*/
    puts(pA);   /*prikaz onoga na sta pA pokazuje*/

    pB = stringB; /*pokazivač na string B*/
    putchar('\n'); /*prelaz u sledeci red*/

    while (*pA != '\0') /*dok ne stignemo do kraja stringa A*/
    {
        /*dodela svakom clanu stringa B clan stringa A*/
        *pB++ = *pA++;
    }
    *pB = '\0';  /*dodela oznake za kraj stringa B*/

    puts(pB); /*prikaz stringa B na ekranu*/
}
```

U ovom primeru smo prvo definisali dva niza karaktera od po 40 karaktera. U glavnom programu se definišu dva pokazivača *pA* i *pB*. Oni na početku ne pokazuju ni na šta određeno već se samo deklarišu kao pokazivači na tip ***char***. U okviru ***while*** petlje se vrši prepisivanje sadržaja na koji pokazuje pokazivač *pA* u sadržaj na koji pokazuje pokazivač *pB*. Desni postfiksni inkrementator uvećava adrese na koje trenutno pokazuju *pA* i *pB*. Petlja ***while*** se završava kada pokazivač *pA* pokaže na null karakter odnosno na poslednji član niza karaktera. Posle izlaska iz petlje neophodno je dodati vrednost null karaktera na kraj stringa

B odnosno vrednosti na koju trenutno pokazuje *pB*, jer smo rekli da je nulti karakter obavezan i označava kraj stringa.

RAZLIKE IZMEĐU POKAZIVAČA NA STRING I NIZA KARAKTERA

Kao i kod rada sa običnim numeričkim podacima, ime niza karaktera je konstantan pokazivač i uvek će pokazivati na prvi element niza karaktera

Ukoliko je pokazivač *pmessage* deklarisan kao:

```
char *pmessage;
```

onda iskaz:

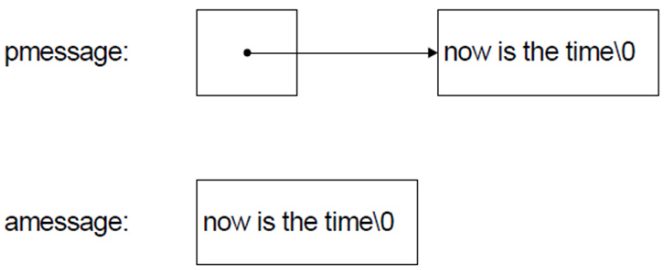
```
pmessage = "now is the time";
```

dodeljuju pokazivaču *pmessage* adresu prvog karaktera u nizu. Prethodni iskaz nije kopiranje niza pošto radimo sa pokazivačima. C jezik ne sadrži operacije koje ceo string, odnosno niz karaktera, tretiraju kao jedinstvenu celinu. Postoji značajna razlika među sledećim definicijama:

```
char amessage[] = "now is the time"; /* niz */
char *pmessage = "now is the time"; /* pokazivac */
```

Naime, *amessage* je niz, dovoljno veliki da se u njega smesti sekvenca karaktera, koja se završava sa nul karakterom***'\0'*** . Individualni karakteri u okviru niza mogu biti promenjeni ali *amessage* kao ime niza će biti konstantan pokazivač i uvek će pokazivati na prvi element.

Sa druge strane, *pmessage* je pokazivač, inicijalizovan da pokazuje na string konstantu; ovaj pokazivač može biti modifikovan tako da u nekom drugom vremenskom trenutku pokazuje negde drugde, ali u tom slučaju gubimo mogućnost promene niza karaktera na koji je pokazivač prethodno pokazivao, Slika 4.1.



Slika 4.1 Razlika između niza amessage i pokazivača pmessage [4]

Osim niza pokazivača na ceo broj, moguće je koristiti niz pokazivača na listu stringova. O tome smo ranije već pričali i opisali smo prednosti korišćenja niza pokazivača. Pogledajmo primer:

```
#define ARRAY_LEN 100
char *myStrPtr[ARRAY_LEN] =  // Niz pokazivaca na tip char
{
  "Marko",
  "Janko",
  "Slavko"
};
```

Primenom niza pokazivača na objekte (u ovom slučaju pokazivač na stringove) se alocira (zauzme) memorija samo za one objekte (tekstove) koji ustvari postoje. Nekorišćeni elementi niza će u tom slučaju biti nul pokazivači. Na taj način se vrši značajna ušteda memorije u odnosu na statički definisan niz podataka, npr. koji je dimenzija 100 X 256.

POKAZIVAČI I FUNKCIJE ZA RAD SA STRINGOVIMA

Funkcije za rad sa stringovima kao argumente koriste pokazivače na početne karaktere stringova kojima manipulišu. Jedino tako je moguće rezultat izmene videti u bloku pozivaoca

U primeru koji sledi koristi se standardna funkcija *strcat* da bi se dodao sadržaj stringa *str2* na kraj stringa *str1*.

```
#include <string.h>

char str1[30] = "Hajdemo";
char str2[ ] = " u Leskovac!";
/* ... */
strcat( str1, str2 );
puts( str1 );
```

Kao izlaz, prikazuje se novi sadržaj stringa *str1*:

```
Hajdemo u Leskovac!
```

Već znamo da su *str1* i *str2* u stvari pokazivači na prvi element niza karaktera, odnosno na prvi karakter u stringu.

Ono što je ovde bitno spomenuti je da funkcije za manipulaciju stringovima, kao što su *strcat* i *puts*, kao argumente prihvataju početne adrese stringova. Takve funkcije u stvari procesiraju prosleđeni string, karakter po karakter, sve dok se ne stigne do kraja stringa odnosno do nul karaktera ***'\0'*** . Uostalom, pogledajmo jednu od mogućih implementacija funkcije *strcat*:

```
char *strcat( char * s1, const char * s2 )
{
  char *rtnPtr = s1;

  // Pronalazi kraj stringa s1:
  while ( *s1 != '\0' )
    ++s1;
```

```
  // Spajanje stringova
  while (( *s1++ = *s2++ ) != '\0' );

  return rtnPtr;
}
```

Kao što možemo da vidimo, funkcija koristi pokazivače preko kojih se putuje kroz stringove, pronalazi kraj stringa i konačno vrši spajanje stringova.

Niz karaktera čija je adresa prvog elementa označena sa *s1* mora imati dovoljno veliku dužinu, tj. dužina treba da bude bar jednaka zbiru dužina oba stringa, plus jedan za nul karakter. Da bi ovo proverili, neophodno je da pre poziva funkcije *strcat* pozovemo funkciju *strlen* koja kao rezultat vraća dužinu stringa na koji pokazuje argument funkcije:

```
if ( sizeof(str1) >= ( strlen( str1 ) + strlen( str2 ) + 1 )
  strcat( str1, str2 );
```

Poglavlje 5

Pokazivači i konstante

KONSTANTAN POKAZIVAČ I POKAZIVAČ NA KONSTANTU

Kao i ostale promenljive, pokazivači mogu biti deklarisani kao konstante. Postoje dva načina da se kombinuju pokazivači sa ključnom rečju const, i često dolazi do mešanja ove dve kombinacije

Konstantan pokazivač

Da biste deklarisali konstantan pokazivač koristi se ključna reč ***const*** koja se stavlja između znaka asteriska (zvezdica) i naziva pokazivačke promenljive:

```
int nValue = 5;
int *const pnPtr = &nValue;
```

Kao i kod obične ***const*** promenljive, ***const*** pokazivač mora biti inicijalizovan po deklaraciji i njegova vrednost se kasnije ne može menjati. Ovo znači da će ***const*** pokazivač uvek pokazivati na istu adresu. U prethodnom primeru *pnPtr* će uvek pokazivati na adresu promenljive *nValue*. Pošto promenljiva *nValue* nije deklarisana kao konstantna, moguće je menjati vrednost koja se nalazi na toj adresi korišćenjem ***const*** pokazivača:

```
*pnPtr = 6; // dozvoljeno, pnPtr pokazuje na nekonstantni int
```

Pokazivač na konstantu

Takođe je moguće deklarisati pokazivač na konstantu korišćenjem ključne reči ***const*** pre tipa podatka promenljive:

```
int nValue = 5;
const int *pnPtr = &nValue;
```

Može se primetiti da pokazivač na konstantu u stvari ne pokazuje na promenljivu koja je deklarisana kao tip ***const***.

Nasuprot tome, ovo znači da pokazivač na konstantu tretira neku promenljivu kao konstantu samo ako joj se pristupa preko pokazivača. Stoga je moguće napisati sledeće:

```
nValue = 6; // nValue nije konstantno
```

ali nije moguće napisati:

```
*pnPtr = 6; // pnPtr tretira svoju vrednost kao konstantu
```

U ovom slučaju, pošto pokazivač nije konstantan, on može biti promenjen u smislu da pokazuje na drugu vrednost, pa možemo napisati sledeće:

```
int nValue = 5;
int nValue2 = 6;

const int *pnPtr = &nValue;
pnPtr = &nValue2; // ovo je u redu!
```

ZAKLJUČNA RAZMATRANJA

Korišćenje pokazivača i konstanti može često da bude konfuzno i zato je neophodno dobro poznavati pravila da ne bi došlo da nepredviđenih okolnosti u vašem programu

U slučaju da prethodno deluje konfuzno, sumirajmo zaključke na osnovu svega navedenog:

- *Običan pokazivač može biti preusmeren sa jedne lokacije na drugu.*
- *Konstantni pokazivač uvek pokazuje na istu adresu i ova adresa na može biti promenjena.*
- *Pokazivač na nekonstantnu vrednost može da menja vrednost na adresi na koju pokazuje.*
- *Pokazivač na konstantu vrednost tretira promenljivu kao konstantu (iako je promenljiva možda deklarisana kao nekonstantna) pa stoga ne može da menja vrednost na koju pokazuje.*

Konačno, moguće je deklarisati konstantan pokazivač na konstantnu vrednost:

```
const int nValue;
const int *const pnPtr = &nValue;
```

Konstantni pokazivač na konstantnu vrednost ne može biti preusmeren da pokazuje na druge adrese, niti može da menja vrednost na koju pokazuje.

Konstantni pokazivači se najčešće koriste kod prosleđivanja vrednosti funkcijama.

Poglavlje 6

Vežbe

NIZOVI I POKAZIVAČI (10 MIN)

U narednom primeru je pokazano korišćenje nizova i pokazivača gde se štampanje nizova može vršiti direktnim ili posrednim pristupom pojedinačnim članovima nizova

```
#include <stdio.h>
void main(void)
{
    float a,b,c, fniz[5] = {0.01, 0.1, 0.5, 1., 10.};
    float *p_fniz;
    char tekst[ ] ={"Ovo je znakovni niz\n"};
    char *p_tekst;
    int i;

    p_fniz=fniz; /* p_fniz= & fniz[0] */
    a=*p_fniz; /* a=fniz[0] */
    b=*(p_fniz+2); /* b=fniz[2] */

    p_fniz=&fniz[2];

    /* niz[4]=fniz[0]+fniz[3] */
    *(p_fniz+2)=*(p_fniz-2)+ *(p_fniz+1);
    c=*(fniz+4); /* c=fniz[4] */

    printf("a=%f b=%f c=%f\n",a,b,c);

    /* direktan pristup clanovima niza tekst */
    for(i=0; tekst[i]!='\0'; ++i)
        putchar(tekst[i]);
    /* pristup preko pokazivaca clanovima niza tekst */
    for(p_tekst=tekst; *p_tekst!='\0'; ++p_tekst)
        putchar(*p_tekst);
}
```

Iz primera se vidi da kada se u deklaraciji inicijalizuje niz (***char*** *tekst[]*) ne mora se navoditi eksplicitno broj članova niza. Kada se pokazivaču dodeli vrednost adrese nekog od članova niza, onda se članovima niza pristupa posredno preko te adrese (ta adresa je reper). Štampanje nizova se može takođe vršiti direktnim ili posrednim pristupom pojedinačnim članovima nizova. U konkretnom primeru to je demonstrirano na štampi tekstualnog niza u

for petlji. Kao uslov za broj prolaza u petlji iskorišćena je osobina da se svi znakovni nizovi (stringovi) u C-u završavaju znakom**"\0"** .

POKAZIVAČ NA STRING KAO ARGUMENT FUNKCIJE (10 MIN)

Napisati funkciju koja računa broj slova u tekstu, a koja kao argument ima pokazivač na string

```
#include <stdio.h>
int duzina(char *p_string);

void main(int argc, char *argv[])
{
    char *p_tekst={"Ovo je znakovni niz\n"};
    int i;

    for(i=1; i<argc; i++)
        printf("argument%d %s\n", i, argv[i]);

    i=duzina(p_tekst);
    printf("duzina stringa je %d", i);
}

int duzina(char *p_string)
{
    int l=0;
    char *p;
    p=p_string;

    while(*p != '\0')
    {
        ++l;
        ++p;
    }
    return l;
}
```

U ovom primeru na dva mesta je pokazan način zajedničke upotrebe pokazivača kao parametra funkcija. Prvo se pokazivač *p_string* vidi kao argument funkcije u prototipu funkcije *duzina*. Zatim se u funkciji kao argument *main* pojavljuje niz pokazivača *argv[]*. Prvi slučaj je tipičan način upotrebe pokazivača kao parametra funkcije. Funkcija ima zadatak da izbroji slova u stringu koji se prenosi preko pokazivača. U samoj fukciji se deklariše pomoćni pokazivač da po povratku u glavni program ne bi bio promenjen pokazivač koji je prosleđen. Prebrojavanje se vrši u petlji dok pokazivač ne dođe na karakter *'\0'*.

POKAZIVAČ NA FUNKCIJU KAO ARGUMENT FUNKCIJE (10 MIN)

Napisati C program koji izračunava sumu kvadrata, kubova i dvostrukih vrednosti od 1 do n

Funkcije za određivanje kvadrata i kuba broja, kao i funkcija za dupliranje broja:

```
#include <stdio.h>

int kvadrat(int n)
{
    return n*n;
}

int kub(int n)
{
    return n*n*n;
}

int parni_broj(int n)
{
    return 2*n;
}
```

Funkcija koja vrši sumiranja, i glavni *main* program:

```
int sumiraj(int (*f) (int), int n)
{
    int i, suma=0;
    for (i=1; i<=n; i++)
        suma += (*f)(i);
    return suma;
}

void main()
{
    printf("Suma kvadrata brojeva od jedan do 3 je %d\n",
        sumiraj(&kvadrat,3));
    printf("Suma kubova brojeva od jedan do 3 je %d\n",
        sumiraj(&kub,3));
    printf("Suma prvih pet parnih brojeva je %d\n",
        sumiraj(&parni_broj,5));
}
```

NIZ POKAZIVAČA NA FUNKCIJU (15 MIN)

Napisati program koji izračunava zbir i razliku odgovarajućih podataka korišćenjem funkcija i pokazivača na funkcije

```
#include <stdio.h>

int zbir(int *, int *);
int razlika(int *, int *);

void main(void)
{
    int a=10, b=5, r_zbir, r_razlika;
    int (*pointer[2])(int *, int *);

    pointer[0]=zbir;
    pointer[1]=razlika;

    r_zbir=(*pointer[0])(&a, &b);
    r_razlika=(*pointer[1])(&a, &b);

    printf("zbir=%d razlika=%d", r_zbir, r_razlika);
}

int zbir(int *a, int *b)
{
    return *a+*b;
}

int razlika(int *a, int *b)
{
    return *a-*b;
}
```

Prethodni primer pokazuje kako se deklarišu i koriste pokazivači na funkcije na primeru niza pokazivača *(*pointer[2])(int *, int *)*. Kao što se vidi, parametri funkcija su takođe pokazivači, a same funkcije su elementarne: zbir i razlika celih brojeva. Treba primetiti način na koji se inicijalizuju pokazivači na funkcije; sledi da su nazivi funkcija istovremeno i pokazivači.

Zadatak za samostalan rad

Napisati i funkcije za proizvod, količnik moduo(ostatak pri deljenju), i eksponent broja (e^x), i izvršiti primenu ovih funkcija u glavnom programu.

Poglavlje 7

Zadaci za samostalni rad

ZADACI ZA SAMOSTALNO VEŽBANJE

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

Zadatak 1. Napisati funkciju *f* sa povratnim tipom ***void*** koja ima samo jedan argument tipa ***int*** i vraća ga udvostručenog. (5 min)

Zadatak 2. Napisati funkciju *f* sa povratnim tipom ***void*** koja ima tačno jedan argument, a koja prosleđenu celobrojnu vrednost vraća sa promenjenim znakom. (10 min)

Zadatak 3. Napisati funkciju koja ima povratni tip ***void***, aistovremeno vraća dve vrednosti - količnik i ostatak dva data broja. Implementirati funkciju korišćenjem pokazivača. (10 min)

Zadatak 4. Napisati funkciju sa tipom povratne vrednosti ***void*** koja služi za izračunavanje zbira i razlike svoja prva dva argumenta. (10 min)

Zadatak 5. Napisati funkciju koja u nizu određuje najdužu seriju elemenata koji zadovoljavaju dato svojstvo. Svojstvo dostaviti kao parametar funkcije. Iskoristiti je da bi se našla najduža serija parnih kao i najduža serija pozitivnih elemenata niza. (10 min)

DODATNI ZADACI ZA SAMOSTALNI RAD

Koristeći materijal sa predavanja i vežbi, rešiti sledeće zadatke

Zadatak 1. U okviru vežbi lekcije L03 smo opisali način kako je moguće samostalno napisati osnovne funkcije za rad sa stringovima.

Napisati funkcije strstr, strlen, strchr korišćenjem pokazivača. (10 min)

Zadatak 2. Napisati program kojim se realizuje ciklično premeštanje vrednosti elemenata niza x za m mesta ulevo. Problem rešiti pomoću funkcije koja kao

argument koristi pokazivač na niz. (10 min)

Zadatak 3. Napisati funkciju koja korišćenjem pokazivača kao argumenta funkcije ispituje da li je string palindrom. (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-DZ04-Ime-Prezime-BrojIndeksa. Zipovani folder CS130-DZ04-Ime-Prezime-BrojIndeksa poslati predmetnom asistentu (lazar.mrkela@metropolitan.ac.rs) u mejlu sa naslovom (subject)CS130-DZ04, 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 predstavljenog gradiva, možemo zaključiti sledeće:

Pokazivač je promenljiva ili konstanta čija je vrednost adresa druge promenljive ili konstante. Neki zadaci u C-u se mnogo lakše rešavaju korišćenjem pokazivača, dok je neke druge probleme nemoguće rešiti bez upotrebe pokazivača.

Uvek treba eksplicitno dodeliti neku vrednost pokazivaču pre njegovog korišćenja, jer se u suprotnom mogu javiti neke ozbiljne greške pri izvršavanju programa.

Vrednost konstante *NULL*, memorijska adresa 0, signalizira da pokazivaču nije dodeljena nijedna dostupna memorijska adresa, tako da su izbegnute eventualne greške da se pomoću pokazivača pristupi i menja sadržaj nekoj već postojećoj memorijskoj lokaciji.

Operator indirekcije se koristi kod pokazivača sa ciljem da se pristupi vrednosti promenljive ili konstante na koju pokazivač pokazuje. Za ovaj operator se drugačije kaže da dereferencira pokazivač.

Pokazivač može biti promenljiva ili konstanta. Ime niza je konstantan pokazivač koji uvek pokazuje na prvi član niza. Pokazivačka promenljiva, deklarisana bez ključne reči ***const***, može da pokazuje na različite promenljive i konstante u toku izvršavanja programa.

Pokazivač može biti uvećan (inkrementiran). Uvećanje pokazivača se koristi u petlji prilikom prolaska kroz članove niza. Uvećanje pokazivača za 1 označava uvećanje njegove vrednosti za veličinu (broj bajtova) njegovog tipa podatka.

Pokazivači mogu biti prosleđeni funkciji kroz listu argumenata. Ovo se naziva prosleđivanje po adresi. Pokazivačka notacija se ovde koristi sa ciljem da se naglasi da je argument funkcije pokazivač. Razlika u sintaksi između prosleđivanja po vrednosti i po adresi je u tome što se u prototipu ili zaglavlju funkcije koristi znak asterisk (*) za prosleđivanje po adresi, dok je prilikom poziva funkcije neophodno koristiti adresni operator (&).

Povratna vrednost funkcije može biti pokazivač. U tom slučaju pokazivač treba da pokazuje ili na dinamički kreiranu promenljivu ili na statičku (***static***), a nikako na lokalnu promenljivu koja je kreirana unutar bloka funkcije.

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] Filip Maric, Predrag Janicic, Osnove programiranja kroz programski jezik C, Univerzitet u Beogradu, 2014.

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

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

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

[8] http://www.learncpp.com/