Lekcija 04

Pokazivači

Miljan Milošević

POKAZIVAČI

	01	02	03	04
Uvod	**Uvod u pokazivače**	**Pokazivači i nizovi**	**Pokazivačka aritmetika**	**Pokazivači i višedimenzionalni nizovi**
	➢ *Osnovi o pokazivačima* ➢ *Šta su pokazivači?* ➢ *Kako koristiti pokazivače* ➢ *Indirektni operator i dereferenciranje* ➢ *Primer korišćenja pokazivača* ➢ *Nul (NULL) pokazivači* ➢ *Poređenje pokazivača*	➢ *Upotreba pokazivača kod nizova* ➢ *Pokazivači, nizovi i adrese u memoriji* ➢ *Posredan pristup članovima niza* ➢ *Korelacija nizova i pokazivača*	➢ *Osnovno o pokazivačkoj aritmetici* ➢ *Operator uvećanja kod pokazivača (inkrementiranje)* ➢ *Primer inkrementiranja pokazivača* ➢ *Operator umanjenja kod pokazivača (dekrementiranje)* ➢ *Razlike između nizova i pokazivača*	➢ *Nizovi pokazivača* ➢ *Pokazivači na pokazivače* ➢ *Pokazivači na višedimenzionalne nizove* ➢ *Upotreba pokazivača na 2D nizove* ➢ *Poređenje pokazivača i višedimenzionalnih nizova*

POKAZIVAČI

05

Pokazivači i funkcije

- *Pokazivači kao argumenti funkcije*
- *Upotreba pokazivača kao argumenata funkcije*
- *Upotreba pokazivača kao argumenata funkcije*
- *Pokazivači na funkcije*
- *Niz pokazivača na funkcije*
- *Vrednost funkcije je pokazivač*

06

Pokazivači i stringovi

- *Upotreba pokazivača na stringove*
- *Niz pokazivača na listu stringova*
- *Nizovi pokazivača na stringove*

07

Pokazivači i funkcije za rad sa stringovima

- *Razlike između pokazivača na string i niza karaktera*
- *Pokazivači i funkcije za rad sa stringovima*
- *Definisanje funkcije strcpy() korišćenjem pokazivača*

08

Pokazivači i konstante

- *Konstantan pokazivač i pokazivač na konstantu*
- *Pokazivači i konstante – zaključak*

09

Složene deklaracije

- *Proste i složene deklaracije*
- *Interpretacija složenih deklaracija*
- *Primeri interpretacije složenih deklaracija*

POKAZIVAČI

10

Vežbe

- *Primer. Osnovi upotrebe pokazivača*
- *Primer. Pokazivačka aritmetika*
- *Pokazivač kao argument funkcije – Primer1*
- *Pokazivač kao argument funkcije – Primer2*
- *Primer. Veza između pokazivača i nizova*
- *Primer. Nizovi i pokazivači*

UVOD

Ova lekcija treba da ostvari sledeće ciljeve:

U okviru ove lekcije studenti se upoznaju sa pojmovima u vezi pokazivača u programskom jeziku C:

- ❑ Deklaracija i korišćenje pokazivača
- ❑ Pokazivači i nizovi
- ❑ Pokazivačka aritmetika
- ❑ Pokazivači i višedimenzionalni nizovi
- ❑ Pokazivači i stringovi
- ❑ Pokazivači i funkcije za rad sa C-stringovima
- ❑ Pokazivači i konstante
- ❑ Složene deklaracije

Pokazivač (eng. pointer) je tip promenljive koji pokazuje na drugu promenljivu ili konstantu. Vrednost pokazivača je adresa promenljive ili konstante na koju pokazuje. Na sličan način mogu da se ponašaju i ljudi. Ukoliko vas neko pita za adresu na kojoj neka osoba živi i ukoliko to nije dovoljno blizu da se fizički pokaže rukom, onda ćete vi verovatno navesti adresu pomoću koje će kuća na kojoj ta osoba živi biti lako locirana.

Pokazivači imaju takvu reputaciju kod studenata kao materija koja se veoma teško uči. Međutim, mnogi smatraju da je ta reputacija prenaduvana. Naime, pokazivači nisu teški za učenje ukoliko im se posveti dovoljno vremena u cilju boljeg razumevanja. Bilo teško ili ne, učenje pokazivača je veoma bitno. Neki C zadaci će biti mnogo lakše rešeni korišćenjem pokazivača, dok drugi zadaci, kao što je na primer dinamička alokacija memorije, jednostavno ne mogu biti obavljeni bez korišćenja pokazivača.

Uvod u pokazivače

pokazivač, adresa, promenljiva, memorija, poređenje

- *Osnovi o pokazivačima*
- *Šta su pokazivači?*
- *Kako koristiti pokazivače*
- *Indirektni operator i dereferenciranje*
- *Primer korišćenja pokazivača*
- *Nul (NULL) pokazivači*

01

OSNOVI O POKAZIVAČIMA

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

Svi programski jezici interno, na nivou prevodioca, koriste podatke koji predstavljaju adrese (pozicije u memoriji računara) svih tipova podataka zastupljenih u jeziku. Upravo preko adrese podacima je moguće pristupati - očitavati ili menjati vrednosti. Iako se interno u jeziku radi sa adresama podataka, u većini jezika nije omogućen, na programerskom nivou, pristup podacima koji predstavljaju adrese. Ovakva restrikcija smanjuje fleksibilnost pri programiranju. Sa druge strane, na ovaj način eliminisane su brojne i teške greške koje mogu nastati pri upotrebi ovih podataka. Programski jezik C (i C++) omogućava i afirmiše upotrebu ove vrste podataka koji predstavljaju specijalni elementarni tip podataka i naziv im je pokazivači (pointeri). U deklaraciji pokazivača pojavljuje se operator posrednog pristupa *. Opšti oblik deklarisanja pokazivača je:

```
tip *ime_pokazivača;
```

i treba ga interpretirati na sledeći način: "ime_pokazivač je pokazivač na tip". U ovoj deklaraciji tip označava tip podatka na koji pokazuje pokazivač ime_pokazivača. Promenljiva ime_pokazivača dobija vrednost inicijalizacijom (izjednačavanje sa adresom nekog podatka) ili sabiranjem i oduzimanjem (drugih) pokazivača i celobrojnih vrednosti.

Kao što vam je već poznato, svaka promenljiva je ustvari smeštena na nekoj memorijskoj lokacija, a svaka memorijska lokacija ima svoju adresu koja je određena korišćenjem operatora & (ampersand). U nastavku je dat primer u kome se vrši štampanje adrese definisanih promenljivih u programu:

```
#include <stdio.h>

int main ()
{
   int  var1;
   char var2[10];

   printf("Address of var1 variable: %x\n", &var1
);
   printf("Address of var2 variable: %x\n", &var2
);

   return 0;
}
```

Rezultat će biti:

```
Address of var1 variable: bff5a400
Address of var2 variable: bff5a3f6
```

ŠTA SU POKAZIVAČI?

Pokazivač je promenljiva čija vrednost predstavlja adresu druge promenljive, tj. direktnu adresu memorijske lokacije

Pokazivač je promenljiva čija vrednost predstavlja adresu druge promenljive, tj. direktnu adresu memorijske lokacije. Kao i pri radu sa ostalim promenljivama i konstantama, pokazivač je neophodno deklarisati pre upotrebe. Kao što smo već spomenuli opšti oblik deklaracije pokazivača je:

```
type *var-name;
```

pri čemu, type predstavlja osnovni tip pokazivača, i mora biti validan C/C++ tip podatka, dok var-name predstavlja ime pokazivačke promenljive. Znak zvezda (asterisk *) se koristi sa ciljem da se ukaže kompajleru da se radi o pokazivačkoj promenljivoj. U nastavku su dati primeri deklaracije pokazivača na različite tipove podataka:

```
int    *ip;    /* pointer to an integer */
double *dp;    /* pointer to a double */
float  *fp;    /* pointer to a float */
char   *ch     /* pointer to a character */
```

Stvarni tip podatka svih prethodno deklarisanih pokazivača, bilo da pokazuju na tip int, float, char ili neki drugi tip podatka, je ustvari veliki heksadecimalni broj koji predstavlja adresu u memoriji. Jedina razlika između pokazivača na različite tipove podataka je ustvari tip promenljive ili konstante na koju pokazivačka promenljiva pokazuje.

KAKO KORISTITI POKAZIVAČE

Koraci u radu sa pokazivačima su: deklaracija, dodela vrednosti i pristup vrednosti na memorijskoj adresi na koju pokazivač pokazuje

Postoji nekoliko bitnih koraka koji su praksa pri radu sa pokazivačima: **(a)** definisanje pokazivačke promenljive, **(b)** dodela adrese promenljive pokazivaču i **(c)** konačno pristup vrednosti na memorijskoj lokaciji korišćenjem pokazivača. Pristup vrednosti na memorijskoj adresi se obavlja korišćenjem operatora zvezda (*) uz ime pokazivačke promenljive, koji ustvari vraća vrednost promenljive smeštene u delu memorije na koju pokazivač ukazuje. U nastavku je dat primer korišćenja ovih operacija:

```
#include <stdio.h>

int main ()
{
   int  var = 20;   /* actual variable declaration */
   int  *ip;        /* pointer variable declaration */

   ip = &var;  /* store address of var in pointer variable*/

   printf("Address of var variable: %x\n", &var  );

   /* address stored in pointer variable */
   printf("Address stored in ip variable: %x\n", ip );

   /* access the value using the pointer */
   printf("Value of *ip variable: %d\n", *ip );

   return 0;
}
```

Rezultat prethodnog programa je:

```
Address of var variable: bffd8b3c
Address stored in ip variable: bffd8b3c
Value of *ip variable: 20
```

INDIREKTNI OPERATOR I DEREFERENCIRANJE

Osnovna svrha upotrebe pokazivača je da pristupe adresi i, ako je potrebno, da promene vrednost promenljive na koju pokazuju

Posmatrajmo sledeći primer gde se vrednost promenljive num menja dva puta.

```
#include <stdio.h>
int main ()
{
   int num = 5;
   int* iPtr = &num;
   printf("The value of num is %d\n", num);
   num = 10;
   printf("The value of num after num = 10 is %d\n",
                num);
   *iPtr = 15;
   printf("The value of num after *iPtr = 15 is
%d\n",
                num);
   return 0;
}
```

Prva izmena je naravno dobro poznata gde 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;
```

Indirektni operator je asterisk(*), isti asterisk koji se koristi za deklaraciju pokazivača ili za množenje. U prethodnom iskazu, asterisk ne služi ni za deklaraciju ni za množenje već se koristi u kontekstu kao indirektni operator.

Dodavanjem indirektnog operatora ispred naziva pokazivačke promenljive se vrši takozvano deferenciranje pokazivača. Vrednost pokazivača nakon dereferenciranja nije adresa, već ustvari vrednost na toj adresi, odnosno vrednost promenljive na koju pokazivač pokazuje. Na primer, u prethodnom programu vrednost pokazivača iPtr je adresa promenljive num. Međutim, vrednost dereferenciranog pokazivača je ustvari vrednost promenljive num. Stoga, naredna dva iskaza imaju isti efekat, jer oba menjaju vrednost promenljive num:

```
num = 25;
 *iPtr = 25;
```

Na sličan način, dereferencirani pokazivač može biti korišćen u aritmetičkom izrazu umesto promenljive na koju pokazuje. Stoga će naredna dva iskaza imati isti rezutat:

```
   num *= 2;
   *iPtr *= 2;
```

U ovim primerima, promena vrednosti promenljive korišćenjem indirektnog operatora se čini dosta komplikovana u odnosu na direktnu promenu, Međutim, u primerima koji će biti opisani u okviru ove (pokazivači i nizovi) ili neke od narednih lekcija (dinamičko alociranje memorije) pokazaćemo da je korišćenje pokazivača jednostavniji ili čak jedini način da se uopšte izvrše zadaci.

PRIMER KORIŠĆENJA POKAZIVAČA

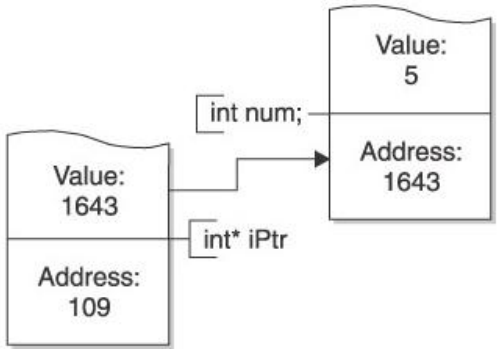
Vrednost pokazivača je veliki heksadecimalni broj koji predstavlja adresu u memoriji one promenljive na koju pokazuje

U nastavku je dat još jedan primer koji opisuje korišćenje adresnog operatora da bi se adresa promenljve dodelila pokazivaču. Ovaj program takođe pokazuje da je vrednost pokazivača ista kao i adresa na koju pokazivač pokazuje.

```
#include <stdio.h>
int main ()
{
   int num = 5;
   int* iPtr = &num;
   printf("The address of x using &num is %x\n",&num);
   printf("The address of x using iPtr is %x\n",iPtr);
   return 0;
}
```

Rezultat programa je:

```
The address of x using &num is 0012FED4
The address of x using iPtr is 0012FED4
```



Slika-1 Pokazivač na adresu na kojoj se nalazi celobrojna vrednost

NUL (NULL) POKAZIVAČI

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

Uvek je dobra praksa da se pokazivaču dodeli nul vrednost kada mu nije dodeljena konkretna adresa, odnosno kada on nema adresu na koju pokazuje. Ovo se najčešće radi u trenutku kada se vrši deklaracija pokazivača. Pokazivač kome je dodeljena vrednost NULL 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 dat primer deklaracije pokazivača čija je vrednost NULL:

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

   printf("The value of ptr is : %x\n", ptr  );

   return 0;
}
```

Rezultat prethodnog programa bi bio:

```
The value of ptr is 0
```

Kod skoro svih operativnih sistema, programima nije dozvoljeno da pristupe memorijskoj lokaciji sa adresom 0, jer je taj deo memorije rezervisan od strane operativnog sistema. Međutim, memorijska lokacija sa adresom nula ima specifično značenje: ona ukazuje da pokazivač nema nameru da pokazuje na dostupnu memorijsku adresu. Po usvojenoj 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)     /* succeeds if p is not null */
if(!ptr)    /* succeeds if p is null */
```

POREĐENJE POKAZIVAČA

Pokazivači se mogu upoređivati korišćenjem operatora kao što su ==, <, i >, i poređenje ima smisla ako su promenljive na koje pokazuju uređene u odgovarajućem odnosu

Moguće je izvršiti poređenje pokazivača korišćenjem operatora poređenja kao što su ==, <, i >. Ukoliko dva pokazivača p1 i p2 pokazuju na promenljive koje su u odgovarajućem odnosu kao što je slučaj sa elementima niza, onda ima smisla porediti pokazivače p1 i p2.

U nastavku je dat modifikovani kod prethodnog primera gde je u okviru for petlje promenjen uslov tako da se pokazivačka promenljiva ptr uvećava sve dok je adresa na koju ptr pokazuje manja ili jednaka od adrese poslednjeg elementa niza, što je u ovom konkretnom slučaju &var[MAX - 1]:

```
#include <stdio.h>
const int MAX = 3;
int main ()
{
   int  var[] = {10, 100, 200};
   int  i, *ptr;

   /* let us have address of the first element in
pointer */
   ptr = var;
   i = 0;
   while ( ptr <= &var[MAX - 1] )
   {
      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* point to the previous location */
      ptr++;
      i++;
   }
   return 0;
}
```

Rezultat će biti:

```
Address of var[0] = bfdbcb20
Value of var[0] = 10
Address of var[1] = bfdbcb24
Value of var[1] = 100
Address of var[2] = bfdbcb28
Value of var[2] = 200
```

Pokazivači i nizovi

Pokazivači, nizovi, ime niza, niz i memorija, posredan pristup,

10 min

- *Upotreba pokazivača kod nizova*
- *Pokazivači, nizovi i adrese u memoriji*
- *Posredan pristup članovima niza*
- *Korelacija nizova i pokazivača*

02

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

Nizovi i pokazivači su, iako nezavisno definisani tipovi podataka u C-u, funkcionalno vezani i programerski se nadopunjuju. Po analogiji sa elementarnim podacima, preko pokazivača se može posredno pristupati i članovima niza. U postupku deklarisanja vodi se računa da je tip podataka u nizu - tip niza istovremeno i tip na koji pokazuje pokazivač tj. opšta sintaksa je:

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

Identifikator ime_niza je i sam pokazivač na tip tako da se inicijalizovanje pokazivača 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};
```

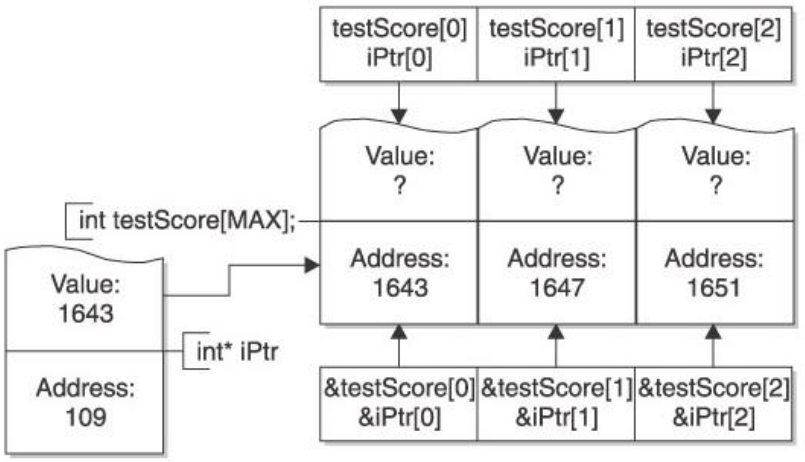
i neka je deklarisana i inicijalizovana promenljiva iptr kao pokazivač na niz testScore, dodeljivanjem adrese niza testScore pokazivačkoj promenljivoj iPtr:

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

Još jednom da napomenemo da ispred imena niza ne stoji adresni operator (&) jer je ime niza ustvari adresa prvog elementa niza. Moguće je napisati sledeći deo koda korišćenjem pokazivača na niz:

```
for (int i = 0; i < MAX; i++)
{
  printf("The address of index %d",i)
  printf(" of the array is %x\n", & iPtr[i]);
  printf("The value at index %d", i)
  printf(" of the array is %d\n", iPtr[i]);
}
```

Stoga će, kao što je prikazano na slici iPtr[2] i testScore[2] imati istu vrednost:



Slika-1 Promenljivi (iPtr) i konstantni (testScore) pokazivači korišćeni za pristupanje članovima niza

POSREDAN PRISTUP ČLANOVIMA NIZA

Pristup članovima niza ne mora biti izvršen samo indeksacijom, već može biti ostvaren posrednim pristupom gde se ime niza posmatra kao pokazivač

Primer koji sledi ilustruje prethodno pokazana sintaksna pravila kojima se posredno pristupa članovima niza

```
#include <stdio.h>

int main ()
{
   /* an array with 5 elements */
   double balance[5] = {1000.0, 2.0, 3.4, 17.0,
50.0};
   double *p;
   int i;

   p = balance;

   /* output each array element's value */
   printf( "Array values using pointer\n");
   for ( i = 0; i < 5; i++ )
   {
       printf("*(p + %d) : %f\n",  i, *(p + i) );
   }

   printf( "Array values using balance as
address\n");
   for ( i = 0; i < 5; i++ )
   {
       printf("*(balance + %d) : %f\n",  i,
*(balance + i) );
   }

   return 0;
}
```

Rezultat programa je:

```
Array values using pointer
*(p + 0) : 1000.000000
*(p + 1) : 2.000000
*(p + 2) : 3.400000
*(p + 3) : 17.000000
*(p + 4) : 50.000000
Array values using balance as address
*(balance + 0) : 1000.000000
*(balance + 1) : 2.000000
*(balance + 2) : 3.400000
*(balance + 3) : 17.000000
*(balance + 4) : 50.000000
```

KORELACIJA NIZOVA I POKAZIVAČA

Pokazivači i nizovi su usko povezani. Pokazivač koji pokazuje na prvi element niza može da pristupi tom nizu ili korišćenjem pokazivačke aritmetike ili korišćenjem indeksacije

Pokazivači i nizovi su usko povezani. Naime, pokazivači i nizovi su u strogoj međusobnoj korelaciji u velikom broju slučajeva. Na primer, pokazivač koji pokazuje na prvi element niza može da pristupi tom nizu ili korišćenjem pokazivačke aritmetike ili korišćenjem indeksacije koja je karakteristična za nizove

```
#include <stdio.h>
#define MAX 3
int main ()
{
   int  var[MAX] = {10, 100, 200};
   int  *ptr,i;

   // let us have array address in pointer.
   ptr = var;
   for (i = 0; i < MAX; i++)
   {
      printf("Address of var[%d] = %x\n",i,ptr);
      printf("Value of var[%d] = %d\n",i,*ptr);

      // point to the next location
      ptr++;
   }
   return 0;
}
```

Rezultat programa će biti:

```
Address of var[0] = 0xbfa088b0
Value of var[0] = 10
Address of var[1] = 0xbfa088b4
Value of var[1] = 100
Address of var[2] = 0xbfa088b8
Value of var[2] = 200
```

Pokazivačka aritmetika

pokazivači, inkrementiranje, dekrementiranje, konstantan pointer

10 min

- *Osnovno o pokazivačkoj aritmetici*
- *Operator uvećanja kod pokazivača (inkrementiranje)*
- *Primer inkrementiranja pokazivača*
- *Operator umanjenja kod pokazivača (dekrementiranje)*
- *Razlike između nizova i pokazivača*

03

OSNOVNO O POKAZIVAČKOJ ARITMETICI

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

Kao što smo već u nekoj od prethodnih lekcija opisali, C pointer je adresa koja sadrži neku brojnu vrednost. Stoga se mogu primeniti aritmetičke operacije nad pokazivačima, kao što je to bilo opisano u prethodnoj lekciji o nizovima, na isti način kao i sa ostalim brojnim vrednostima. Postoje četiri aritmetička operatora koja se mogu primeniti na pokazivačima: ++, --, + i -.

Da bi smo bolje razumeli pokazivačku aritmetiku, pretpostavimo da je ptr pokazivač na celobrojnu vrednost koja se nalazi na adresi označenoj sa 1000. Pretpostavimo da se radi o 32-bitnoj celobrojnoj vrednosti, pa izvršimo sledeću aritmetičku operaciju nad pokazivačem:

```
ptr++;
```

Sada, nakon prethodne operacije, pokazivač ptr će pokazivati na memorijsku lokaciju označenu sa 1004 iz razloga što kada se ptr inkrementira (uveća za jedan) on će pokazivati na sledeću celobrojnu lokaciju koja je za 4 bajta pomerena u odnosu na trenutnu memorijsku lokaciju. Ova operacija inkrementiranja će pomeriti pokazivač na sledeću memorijsku lokaciju, bez narušavanja vrednosti koja se nalazi na trenutnoj lokaciji. Ukoliko ptr pokazuje na karakter (tip char) čija je adresa 1000, onda će nakon prethodne operacije inkrementiranja pokazivač biti pomeren na lokaciju 1001 jer je veličina char tipa podatka 1 bajt pa se za toliko i vrši pomeranje.

OPERATOR UVEĆANJA KOD POKAZIVAČA (INKREMENTIRANJE)

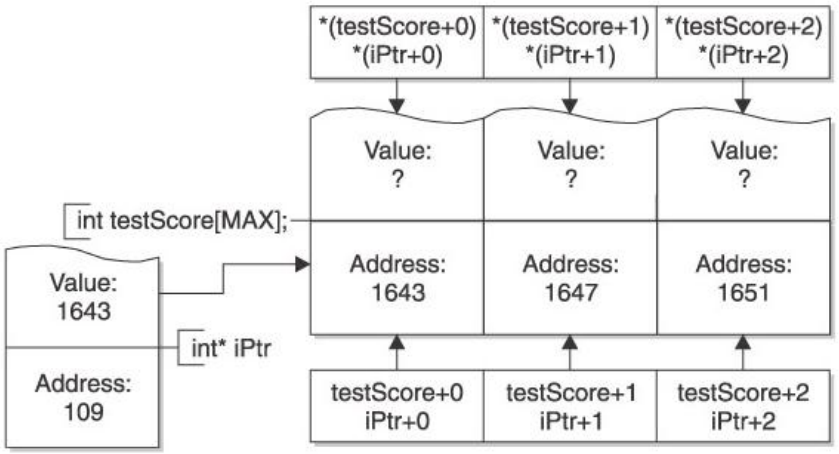
Uvećanjem pokazivača za jedan se ustvari dodaje pokazivaču vrednost veličine tipa podatka (u bajtovima) na koji pokazuje

Često je poželjno koristiti u programima pokazivač na niz jer je moguće uvećati (inkrementirati) pokazivač za razliku od imena niza koji ne može biti inkrementiran jer je ime niza ustvari konstantan pokazivač (o konstantnim pokazivačima će biti više reči na kraju lekcije). 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("The address of index %d of the array
is %d\n",
                          i,iPtr);
      printf("The value at index %d of the array is
%d \n",
                          i,*iPtr);
   }
   return 0;
}
```

Kao što se može videti na Slici-1, 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 Efekat inkrementiranja pokazivača za 1

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

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

PRIMER INKREMENTIRANJA POKAZIVAČA

Inkrementiranje pokazivača se najčešće koristi kod pokazivača na nizove

U narednom programu je dat primer inkrementiranja pokazivača u cilju pristupa uzastopnim elementima celobrojnog niza:

```
#include <stdio.h>
const int MAX = 3;

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

   /* let us have array address in pointer */
   ptr = var;
   for ( i = 0; i < MAX; i++)
   {
      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* move to the next location */
      ptr++;
   }
   return 0;
}
```

Nakon kompajliranja i izvršenja prethodnog koda dobija se sledeći rezultat:

```
Address of var[0] = bf882b30
Value of var[0] = 10
Address of var[1] = bf882b34
Value of var[1] = 100
Address of var[2] = bf882b38
```

OPERATOR UMANJENJA KOD POKAZIVAČA (DEKREMENTIRANJE)

Umanjenjem pokazivača za jedan se ustvari vrednost pokazivača umanjuje za onoliko bajtova kolika je vrednost tipa podatka na koji pokazuje

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, kao što je prikazano u nastavku:

```
#include <stdio.h>
const int MAX = 3;

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

   /* let us have array address in pointer */
   ptr = &var[MAX-1];
   for ( i = MAX; i > 0; i--)
   {
      printf("Address of var[%d] = %x\n", i, ptr );
      printf("Value of var[%d] = %d\n", i, *ptr );

      /* move to the previous location */
      ptr--;
   }
   return 0;
}
```

Izlaz je:

```
Address of var[3] = bfedbcd8
Value of var[3] = 200
Address of var[2] = bfedbcd4
Value of var[2] = 100
Address of var[1] = bfedbcd0
Value of var[1] = 10
```

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;    // This is a correct syntax
      var++;       // This is incorrect.
   }
   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 kontantan 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.

Pokazivači i višedimenzionalni nizovi

Nizovi pokazivača, pokazivač na pokazivač, pokazivač na 2D niz

15 min

- *Nizovi pokazivača*
- *Pokazivači na pokazivače*
- *Pokazivači na višedimenzionalne nizove*
- *Upotreba pokazivača na 2D nizove*
- *Poređenje pokazivača i višedimenzionalnih nizova*

04

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:

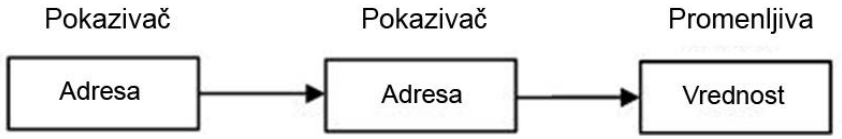
```
#include <stdio.h>
#define MAX 3
int main ()
{
   int  var[] = {10, 100, 200};
   int i, *ptr[MAX];

   for ( i = 0; i < MAX; i++)
   {
      ptr[i] = &var[i]; /* assign the address of integer. */
   }
   for ( i = 0; i < MAX; i++)
   {
      printf("Value of 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 je oblik višestruke indirekcije, odnosno može se posmatrati kao lanac pokazivača. Kao što je poznato, pokazivač sadrži adresu promenljive. Stoga, kada definišemo pokazivač na pokazivač, to ustvari znači da prvi pokazivač sadrži adresu drugog pokazivača, a drugi pokazivač pokazuje na lokaciju na kojoj se nalazi stvarna promenljiva, što je prikazano na sledećoj slici:



Slika-1 Lanac pokazivača kao pokazivač na pokazivač

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 asterisk primeni dva puta, kao što je to opisano u narednom primeru:

```
#include <stdio.h>

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

   var = 3000;

   /* take the address of var */
   ptr = &var;

   /* take the address of ptr using address of
operator & */
   pptr = &ptr;

   /* take the value using pptr */
   printf("Value of var = %d\n", var );
   printf("Value available at *ptr = %d\n", *ptr );
   printf("Value available at **pptr = %d\n",
**pptr);

   return 0;
}
```

Rezultat prethodnog programa je:

```
Value of var = 3000
Value available at *ptr = 3000
Value available at **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} };
// Indexing: myMatrix[i][j] is same as
*(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 ustvari ime dvodimenzionalnog niza, dok su a[0], a[1] i a[2] ustvari 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

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

Početnici u C-u su ponekad u zabuni kada je u pitanju poređenje dvodimenzionalnih nizova i nizova pokazivača. Razmotrimo sada njihove razlike. 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 dvodimenzioni 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 (navođenjem inicijalizatora) ili dinamički (tokom izvršavanja programa). 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 20-to elementni niz - neki mogu da pokazuju na 2-elementni niz, neki na 50-elementi niz, 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 (u koji se, prilikom inicijalizacije upisuju imena meseci):

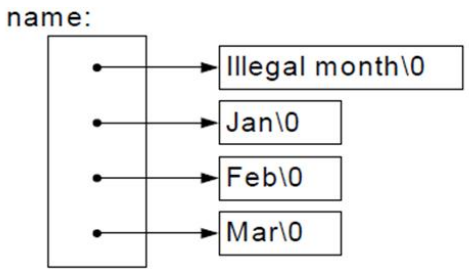
```
char aname[][15] = { "Illegal month", "Jan", "Feb", "Mar" };
```

U memoriji računara ovaj niz može biti predstavljen kao (Slika-2):

Slika-2 Grafička ilustracija niza aname

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

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



Slika-3 Grafička ilustracija pokazivača name na niz stringova

Pokazivači i funkcije

Pokazivači, argument funkcije, vrednost funkcije, pokazivač na f-ju

- *Pokazivači kao argumenti funkcije*
- *Upotreba pokazivača kao argumenata funkcije*
- *Upotreba pokazivača kao argumenata funkcije*
- *Pokazivači na funkcije*
- *Niz pokazivača na funkcije*
- *Vrednost funkcije je pokazivač*

05

POKAZIVAČI KAO ARGUMENTI FUNKCIJE

*Pokazivači kao i ostali tipovi podataka mogu biti argumenti funkcija, s tim da se za razliku od drugih programskih jezika, kroz sintaksu mora provlačiti upotreba operatora * za posredni pristup*

Sintaksa na primeru prototipa funkcije je:

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

U nastavku je dat primer gde 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 );

   /* print the actual value */
   printf("Number of seconds: %ld\n", sec );

   return 0;
}

void getSeconds(unsigned long *par)
{
   /* get the current number of seconds */
   *par = time( NULL );
}
```

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

```
#include <stdio.h>

/* function declaration */
double getAverage(int *arr, int size);

int main ()
{
   /* an int array with 5 elements */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* pass pointer to the array as an argument */
   avg = getAverage( balance, 5 ) ;

   /* output the returned value  */
   printf("Average value is: %f\n", avg );

   return 0;
}

double getAverage(int *arr, int size)
{
  int    i, sum = 0;
  double avg;
}
```

UPOTREBA POKAZIVAČA KAO ARGUMENATA FUNKCIJE

Pokazivači kao argumenti funkcije se koriste u situacijama gde je neophodno da se promene izvršene nad argumentima vide i u bloku odakle je funkcija pozvana

U nastavku je dat još jedan primer korišćenja nizova koji se u funkciju prosleđuju preko pokazivača i vrši se sortiranje korišćenjem metode selekcije selection_sortf( ). Funkcija radi na isti način kao primer koji je obrađen u okviru lekcije o sortiranju, samo se koriste pokazivači da bi se pristupilo članovima niza.

```
// The swapf( ) function exchanges the values of two float variables.
// Arguments:  Two pointers to float.

void swapf( float *p1, float *p2 )
{
   float tmp = *p1;  *p1 = *p2;  *p2 = tmp;   // Swap *p1 and *p2.
}
// The function selection_sortf( ) uses the selection-sort
// algorithm to sort an array of float elements.
// Arguments: An array of float, and its length.

void selection_sortf( float a[ ], int n )  // Sort an array a of n float elements.
{
  register float *last = a + n-1,      // A pointer to the last element.
                 *p,                   // A pointer to a selected element.
                 *minPtr;              // A pointer to the current minimum.

  if ( n <= 1 ) return;                // Nothing to sort.

  for (  ; a < last; ++a )             // Walk the pointer a through the array.
  {
    minPtr = a;                        // Find the smallest element
```

UPOTREBA POKAZIVAČA KAO ARGUMENATA FUNKCIJE

Pokazivači kao argumenti funkcije se koriste u situacijama gde je neophodno da se promene izvršene nad argumentima vide i u bloku odakle je funkcija pozvana

```
for (  ; a < last; ++a )            // Walk the pointer a through the array.
 {
   minPtr = a;                       // Find the smallest element
   for ( p = a+1;  p <= last;  ++p ) // between a and the end of the array.
     if ( *p < *minPtr )
       minPtr = p;
   swapf( a, minPtr );               // Swap the smallest element
 }                                   // with the element at a.
```

Verzija funkcije sa pokazivačima je generalno mnogo efikasnija od funkcije koja koristi indekse, pošto pristupanje elementu niza korišćenjem indeksa i, kao u izrazima a[i] ili *(a+i), zahteva da se na vrednost adrese prvog člana niza a doda vrednosti i*sizeof(*element_type*) da bi se dobila adresa konkretnog člana niza. Verzija sa pokazivačima zahteva manje aritmetike, jer se pokazivač inkementira umesto indeksa, i direktno pokazuje na zahtevani element niza.

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

Upotreba pokazivača na funkcije pruža 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 što će se 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 stoga možemo da pozivamo funkciju korišćenjem pokazivača:

```
double result;
funcPtr = pow; // Let funcPtr point to the function pow( ).
                          // The expression *funcPtr now
yields the
                          // function pow( ).

result = (*funcPtr)( 1.5, 2.0 );  // Call the function
referenced by
                                  // funcPtr.
result = funcPtr( 1.5, 2.0 );     // The same function
call.
```

Kao što ilustruje poslednja linija prethodnog koda, kada se funkcija pozove korišćenjem pokazivača, ne mora da se navode 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 odgovarajuće numeričke dugmiće. 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

U nastavku je dat primer koji od korisnika zahteva da unese dva broja a zatim izvršava prosta izračunavanja nad njima. Matematičke funkcije su pozvane korišćenjem pokazivača koji su elementi niza funcTable.

```
#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; }

// Array of 5 pointers to functions that take two double parameters
// and return a double:
double (*funcTable[5])(double, double)
          = { Add, Sub, Mul, Div, pow }; // Initializer list.

// An array of pointers to strings for output:
char *msgTable[5] = { "Sum", "Difference", "Product", "Quotient",
"Power" };

int main( )
{
  int i;                  // An index variable.
  double x = 0, y = 0;

  printf( "Enter two operands for some arithmetic:\n" );
  if ( scanf( "%lf %lf", &x, &y ) != 2 )
    printf( "Invalid input.\n" );
  for ( i = 0; i < 5; ++i )
    printf( "%10s: %6.2f\n", msgTable[i], funcTable[i](x, y) );

  return 0;
}
```

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 indeksiaranja 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:

```
typedef double func_t( double, double );     // The functions'
type is now
                                             // named func_t.
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

Poslednji ali ne manje važan način zajedničke upotrebe pokazivača i funkcija su funkcije koje kao rezultat vraćaju pokazivač. Sintaksa za prototip funkcije je:

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

Na sličan način kao što se pokazivači prosleđuju funkciji, moguće je vratiti pokazivač iz funkcije kao vrednost funkcije. Da bi se to ostvarilo, neophodno je deklarisati funkciju koja vraća pokazivač na sledeći način:

```
int * myFunction()
{
...
}
```

Ono što ovde treba napomenuti da 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.

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

/* function to generate and retrun random numbers. */
int * getRandom( )
{
   static int  r[10];
   int i;

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

   return r;
}

/* main function to call above defined function */
int main ()
{
   /* a pointer to an int */
   int *p; int i;

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

   return 0;
}
```

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.

Pokazivači i stringovi

Pokazivači, string, nizovi pokazivača na string

- *Upotreba pokazivača na stringove*
- *Niz pokazivača na listu stringova*
- *Nizovi pokazivača na stringove*

06

UPOTREBA POKAZIVAČA NA STRINGOVE

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

Sa C stringovima smo se već upoznali. Stringovi predstavljaju nizove karakatera, 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];

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

 puts(pB); /*prikaz stringa B na ekranu*/
 return (0);
}
```

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

NIZ POKAZIVAČA NA LISTU STRINGOVA

Nizovi pokazivača na stringove se mogu koristi na sličan način kao kod višedimenzionalnih nizova obilnih brojnih vrednosti

Osim niza pokazivača na ceo broj, moguće je korišćenje niza pokazivača na karaktere, kao što je prikazano u sledećem primeru, gde svaki član niza pokazuje na string, što znači da se niz pokazivača koristi da bi se čuvala lista C-stringova.

```
#include <stdio.h>
const int MAX = 4;

int main ()
{
   char *names[] = {
                   "Marko Markovic",
                   "Mirko Markovic ",
                   "Milan Markovic ",
                   "Mitar Markovic ",
   };
   int i = 0;

   for ( i = 0; i < MAX; i++)
   {
      printf("Value of names[%d] = %s\n", i, names[i] );
   }
   return 0;
}
```

Rezultat je:

```
Value of names[0] = Marko Markovic
Value of names[1] = Mirko Markovic
Value of names[2] = Milan Markovic
Value of names[3] = Mitar Markovic
```

NIZOVI POKAZIVAČA NA STRINGOVE

Nizovi pokazivača se koriste u slučajevima gde treba napraviti uštede u memoriji koja je nepotrebno zauzeta statičkim rezervisanje prostora višedimenzionalnog niza

Na primer ukoliko želimo da procesiramo string, njega možemo smestiti u dvodimenzionalni niz, čiji je broj kolona dovoljno veliki da se u svaku pojedinačnu vrstu može smestiti najduži string koji se može javiti:

```
#define ARRAY_LEN 100
#define STRLEN_MAX 256
char myStrings[ARRAY_LEN][STRLEN_MAX] =
{ // Several corollaries of Murphy's Law:
  "If anything can go wrong, it will.",
  "Nothing is foolproof, because fools are so ingenious.",
  "Every solution breeds new problems."
};
```

Medjutim, ovakav način definicije ustvari nepotrebno troši veliki deo memorije, jer se samo mali deo od ukupno 25,600 bajtova koliko je inicijalno zauzeto pri deklaracijii niza koristi. Prosto rešenje u ovakvom i slučnim slučajevima je da se koristi niz pokazivača na objekte(u ovom slučaju pokazivač na stringove) i da se alocira (zauzme) memorija samo za one objekte (tekstove) koji ustvari postoje. Nekorišćeni elementi niza će u tom slučaju biti nul pokazivači.

```
#define ARRAY_LEN 100
char *myStrPtr[ARRAY_LEN] =    // Array of pointers to char
{ // Several corollaries of Murphy's Law:
  "If anything can go wrong, it will.",
  "Nothing is foolproof, because fools are so ingenious.",
  "Every solution breeds new problems."
};
```

Pokazivači i funkcije za rad sa stringovima

Pokazivači, stringovi, funkcije

- *Razlike između pokazivača na string i niza karaktera*
- *Pokazivači i funkcije za rad sa stringovima*
- *Definisanje funkcije strcpy() korišćenjem pokazivača*

07

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;
```

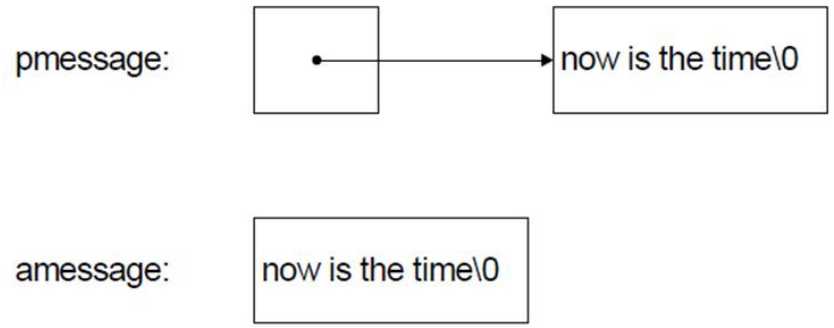
onda iskaz

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

dodeljuju pokazivaču pmessage adresu prvog karaktera u nizu. Prethodni iskaz nije kopiranje niza jer su uposleni samo pokazivači. 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"; /* an array */
char *pmessage = "now is the time"; /* a pointer */
```

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-1 Razlika između niza amessage i pokazivača pmessage

POKAZIVAČI I FUNKCIJE ZA RAD SA STRINGOVIMA

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

U primeru koji sledi koristi se standardna funkcija strcat( ) da bi se dodao string str2 na kraj stringa str1. Niz karaktera, odnosno string str1 mora biti dovoljno veliki da bi mogao da prihvati sve karaktere stringa koji se dodaje na njega.

```
#include <string.h>

char str1[30] = "Let's go";
char str2[ ] = " to London!";
/* ... */
strcat( str1, str2 );
puts( str1 );
```

Kao izlaz, prikazuje se novi sadržaj niza karaktera str1:

```
Let's go to London!
```

Imena str1 i str2 su ustvari pokazivači na prvi element niza karaktera, odnosno na prvi karakter u stringu. Takav pokazivač se naziva "pokazivač na string" ili kraće "string pokazivač". Funkcije za manipulaciju stringovima kao što su strcat( ) i puts( ) kao argumente prihvataju početne adrese stringova. Takve funkcije ustvari procesiraju prosleđeni string karakter po karakter sve dok se ne stigne do kraja stringa odnsno do nul karaktera, '\0'.

Funkcija koja je data u nastavku je samo jedan od mogućih implementacija standardne funkcije strcat( ). Ona koristi pokazivače da bi se putovalo kroz stringove na koje referenciraju (pokazuju) njeni argumenti.

```
// The function strcat( ) appends a copy of the second
string
// to the end of the first string.
// Arguments:    Pointers to the two strings.
// Return value: A pointer to the first string, now
concatenated with the second.

char *strcat( char * s1, const char * s2 )
{
  char *rtnPtr = s1;
  while ( *s1 != '\0' )                // Find the end of
string s1.
    ++s1;
  while (( *s1++ = *s2++ ) != '\0' )   // The first
character from s2 replaces
    ;                                  // the terminator of
s1.
  return rtnPtr;
}
```

Niz karaktera čija je adresa prvog elementa označena sa s1 mora imati dovoljno veliku dužinu, tj. da mu dužina bude jednaka bar zbiru dužina oba stringa, plus jedan za null karakter. Da bi se ovo proverilo, neophodno je pre poziva funkcije strcat( ), pozvati srandardnu funkciju strlen( ), koja kao rezultat vraća dužinu stringa na koju pokazuje argument funkcije:

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

DEFINISANJE FUNKCIJE STRCPY() KORIŠĆENJEM POKAZIVAČA

C jezik ne trerira string kao jedinstvenu celinu već kao niz karaktera, pa je stoga neophodno koristiti cikluse da bi se vršila izmena odgovarajućih stringova sa kojima se radi

Prikazaćemo još neke mogućnosti pokazivača i nizova karaktera posmatrajući dve korisne funkcije iz standardne biblioteke. Prva funkcija je strcpy(s,t) koja kopira niz t u niz s. Bilo bi zgodno da možemo samo napisati s=t, ali ovim bi smo samo iskopirali pokazivač, ne i znakove. Da bi kopirali znakove neophodno je koristiti petlju. Verzija sa nizovima ima sledeći izgled.

```
/* strcpy: copy t to s; array subscript version */
void strcpy(char *s, char *t)
{
          int i;
          i = 0;
          while ((s[i] = t[i]) != '\0')   i++;
}
```

Verzija sa pokazivačima ima sledeći oblik

```
/* strcpy: copy t to s; pointer version */
void strcpy(char *s, char *t)
{
          int i;
          i = 0;
          while ((*s = *t) != '\0')  {
                    s++;
                    t++;
          }
}
```

Kako se argumenti prenose po vrednosti, funkcija strcpy može koristiti parametre s i t kako god to ona želi. U prethodnom primeru, argumenti su prikladno incijalizirani pokazivači, koji putuju kroz niz, znak po znak, sve dok se poslednji znak '\0' ne preslika iz t u s.

Pokazivači i konstante

Pokazivač, konstanta, pokazivač na konstantu, konstantni pointer

- *Konstantan pokazivač i pokazivač na konstantu*
- *Pokazivači i konstante – zaključak*

08

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 bi ste deklarisali konstantan (const) 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 konstanta stoga je moguće menjati vrednosti koja se nalazi na toj adresi korišćenjem const pokazivača:

```
*pnPtr = 6; // allowed, pnPtr points to a non-const 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 ustvari 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 is non-const
```

ali nije moguće napisati:

```
*pnPtr = 6; // pnPtr treats its value as const
```

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

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

const int *pnPtr = &nValue;
pnPtr = &nValue2; // okay
```

POKAZIVAČI I KONSTANTE – ZAKLJUČAK

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:

- Ne konstantni 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 ne-konstantnu vrednost može da menja vrednost na adresi na koju pokazuje.
- Pokazivač na konstantu vrednost tretira promenljivu kao konstantu (iako je promenljiva možda deklarisana kao ne-konstantna) 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.
const pokazivači se najčešće koriste kod prosleđivanja vrednosti funkcijama.

Složene deklaracije

Proste deklaracije, složene deklaracija, interpretacija izraza

- *Proste i složene deklaracije*
- *Interpretacija složenih deklaracija*
- *Primeri interpretacije složenih deklaracija*

09

PROSTE I SLOŽENE DEKLARACIJE

Prilikom interpretacije složenih deklaracija uvek je neophodno krenuti od naziva identifikatora, i izvršiti postupak „korak po korak“ dok se ne interpretiraju svi simboli u deklaraciji

Simboli (), [], i * u deklaraciji specificiraju da identifikator ima funkciju, niz ili pokazivački tip. Složene deklaracije mogu da sadrže višestruko pojavljivanje bilo kog od prethodno pomenutih simbola. Ovi osnovni simboli u deklaraciji imaju sledeće značenje:

- ( ) - funkcija čiji je rezultat tipa (int, char, itd)...
- [ ] - niz čiji su elementi tipa...
- * - pokazivač na ...

U deklaraciji, ovi simboli imaju isti prioritet i asocijativnost kao što bi ovi isti operatori imali i u nekom proizvoljnom izrazu. Osim toga, kao i u izrazu, mogu se koristiti dodatne zagrade da se modifikuje redosled izvršavanja odnosno redosled interpretiranja. Na primer:

```
int *abc[10];    // An array of 10 elements whose type is pointer to int.
int (*abc)[10];  // A pointer to a array of 10 elements whose type is int.
```

U deklaraciji koja uključuje tip funkcije, zagrade koje definišu funkciju mogu da sadrže listu argumenata. U sledećem primeru je deklarisan pokazivač na funkciju:

```
int (*fPtr)(double x);    // fPtr is a pointer to a function that has
                                  // one double parameter and returns int.
```

Prilikom interpretacije složenih deklaracija, uvek je neophodno krenuti od identifikatora (tj. naziva promenljivih). Krenuvši s tog mesta, treba ponavljati sledeće korake dok se ne interpretiraju svi simboli u deklaraciji:

1. Ukoliko se leva obična zagrada (() ili uglasta ([) pojavljuju odmah do odgovarajuće desne zagrade onda odmah interpretirati par zagrada ( ) ili [ ].
2. U suprotnom, ako se znak asterisk (*) pojavi na levoj strani onda prvo interpretirati asterisk.

INTERPRETACIJA SLOŽENIH DEKLARACIJA

Iako se složene deklaracije retko primenljuju u praksi ipak je bitno je njihovo razumevanje i poznavanje načina njihove interpretacije

U nastavku je dat primer i interpretacija složene deklaracije

```
extern char *(* fTab[ ])(void);
```

Korak	Interpretirani simbol	Značenje (pročitati ovu kolonu kao rečenicu od vrha do dna)
1. početi sa identifikatorom	fTab	fTab je...
2. par uglastih zagrada	fTab[ ]	niz čiji su elementi tipa...
3. asterisk s leve strane	(* fTab[ ])	pokazivač na...
4. zagrade za funkciju (i lista parametera) na desnoj strani	(* fTab[ ])(void)	funkciju koja nema argumente, i čija je povratna vrednost tipa...
5. asterisk s leve strane	*(* fTab[ ])(void)	pokazivač na ...
6. nema više asterska, običnih niti uglasti zagrada: očitati tip podatka	char *(* fTab[ ])(void)	char

Slika-1 Postupak interpretacije kod složene deklaracije

fTab je niz nekompletnog tipa, jer deklaracija ne specificira dužinu niza. Pre korišćenja ovog niza neophodno je definisati ga u programu sa unapred zadatom dužinom.
Zagrade () moraju da stoje oko *fTab[ ], jer bez njih fTab bi bio deklarisan kao niz čiji su elementi funkcije. U nastavku je dat još jedan primer i interpretacija na osnovu prethodnog pravila:

```
float (* func( ))[3][10];
```

Prethodni izraz se čita kao:
1. Identifikator func je...
2. funkciju koja nema argumente, čija je povratna vrednost tipa...
3. pokazivač na...
4. niz od tri elementa tipa...
5. niz od 10 elementa tipa...
6. float.

Drugim rečima, funkcija func vraća pokazivač na 2D niz koji ima 3 vrste i 10 kolona. Ovde je opet neophodno navesti zagrade () oko * func( ), jer bez njih funkcija bi bila deklarisana da vraća niz što je nemoguće. U nastavku su dati još neki primeri složenih deklaracija tipova:

```
float *[ ]
                // niz pokazivača na float. Broj elemenata niza je
neodređen
float (*)[10]
                // pokazivač na niz od 10 elemenata, koji su tipa float.
double *(double *)
                // Funkcija čiji je jedan argument pokazivač na double,
                // i čiji je rezultat takođe pokazivač na double
double (*)( )
                //Pokazivač na funkciju čija je povratna vrednost tipa
double.
                // Broj I tip argumenata funkcije nije specificiran.
int *(*(*)[10])(void)
                //Pokazivač na niz od 10 elemenata čiji je tip “pokazivač
na funkciju”,
                //gde funkcija  nema specificiranu listu argumenata
                // i čija je povratna vrednost: “pokazivač na int”.
```

PRIMERI INTERPRETACIJE SLOŽENIH DEKLARACIJA

Programski jezik C često doživljava kritike zbog sintakse njegovih deklracija, i to je jedan od razloga što ima reputaciju veoma teškog jezika za učenje

Programski jezik C doživljava kritike zbog sintakse njegovih deklaracija, posebno onih koje uvode pokazivače na funkcije. Sintaksa mora ujediniti deklarisanje i način upotrebe, što dobro prolazi kod jednostavnih slučajeva. Međutim, kod kompleksnijih problema, sve to deluje konfuzno, što zbog činjenice da se deklaracije ne mogu očitati sleva nadesno, što zbog prečestog korištenja zagrada. Razlika između:

```
int *f(); /* f : funkcija čija je ulazna vrednost pokazivač, a izlazna tipa int */
```

i

```
int (*pf)(); /* pf : pokazivač na funkciju koja vraća celobrojnu vrednost */
```

najbolje ilustrira problem: * je prefiksni operator, pa je manjeg prioriteta od malih zagrada (), tako da su one obavezne radi pravilnog povezivanja. Iako se složene deklaracije retko pojavljuju u praksi, bitno je njihovo razumevanje, a naravno i primena. U nastavku su dati samo neki od primera složenih deklaracija.

```
char **argv
          //argv : pokazivač na pokazivač na char
int (*daytab)[13]
          //daytab : pokazivač na niz[13] od int
int *daytab[13]
          //daytab : niz[13] pokazivača na int
void *comp()
          //comp : funkcija čija je ulazna vrednost pokazivač, a izlazna void
void (*comp)()
          //comp : pokazivač na funkciju koja vraća void
char(*(*x())[])()
          //x : funkcija koja vraća pokazivač na niz[] pokazivača na funkciju koja vraća char
char(*(*x[]))()[]
          // x : polje[3] pokazivač na funkciju koja vraća pokazivač na polje[5] od char
```

Vežbe

Pokazivači, nizovi, funkcije, stringovi

- *Primer. Osnovi upotrebe pokazivača*
- *Primer. Pokazivačka aritmetika*
- *Pokazivač kao argument funkcije – Primer1*
- *Pokazivač kao argument funkcije – Primer2*
- *Primer. Veza između pokazivača i nizova*
- *Primer. Nizovi i pokazivači*

10

PRIMER. OSNOVI UPOTREBE POKAZIVAČA

Za rad sa pokazivačima se koriste dve vrste operatora: adresni (&) i indirektni operator(*)

Primer 1. Pokazivači - osnovni pojam

```
#include <stdio.h>
void main()
{
            int x = 3;
            int *px;
            printf("Adresa promenljive x je : %p\n", &x);
            printf("Vrednost promenljive x je : %d\n", x);
            px = &x;
            printf("Vrednost promenljive px je (tj. px) : %p\n", px);
            printf("Vrednost promenljive na koju ukazuje px (tj. *px) je : %d\n", *px);
            *px = 6;
            printf("Vrednost promenljive na koju ukazuje px (tj. *px) je : %d\n", *px);
            printf("Vrednost promenljive x je : %d\n", x);
}
```

Primer 2. Upotreba pokazivača za posredan pristup promenljivoj

```
#include <stdio.h>
void main(void)
{
            int a;
            int *adresa;
            a=0; /* direktna dodela vrednosti promenljivoj a */
            printf("Promenljiva a je direktno dobila vrednost %d \n", a);
            adresa=&a; /* promenljiva adresa dobija vrednost adrese promenljive a */
            printf("Adresa promenljive a je adresa=%d \n", adresa);
            *adresa=1; /* posredna dodela vrednosti promenljivoj a */
            printf("Promenljiva a je posredno dobila vrednost , a=%d \n", a);
            printf("ili posredno ocitavanje *adresa= %d", *adresa);
}
```

PRIMER. POKAZIVAČKA ARITMETIKA

U nastavku su dati različiti primeri inkrementiranja, dekrementiranja, sabiranja i oduzimanja sa pokazivačima

```
#include <stdio.h>
void main()
{
            char s[] = "abcde";
            int t[] = {1, 2, 3, 4, 5};
            char* ps = &s[0];
            int* pt = &t[0];

            printf("ps = %p\n", ps);
            printf("ps+1 = %p\n", ps+1);
            printf("ps+2 = %p\n", ps+2);
            printf("ps-1 = %p\n", ps-1);
            printf("ps-2 = %p\n", ps-2);
            printf("pt = %p\n", pt);
            printf("pt+1 = %p\n", pt+1);
            printf("pt+2 = %p\n", pt+2);
            printf("pt-1 = %p\n", pt-1);
            printf("pt-2 = %p\n", pt-2);

            for (ps = s; *ps; ps++)
            putchar(*ps);
            putchar('\n');
            ps = &s[3];
            printf("s = %p\n", s);
            printf("ps = %p\n", ps);
            printf("ps - s = %d\n", ps - s);
            pt = &t[3];
            printf("t = %p\n", t);
            printf("pt = %p\n", pt);
            printf("pt - t = %d\n", pt - t);
}
```

POKAZIVAČ KAO ARGUMENT FUNKCIJE – PRIMER1

Demonstracija prenosa argumenata preko pokazivača

```
#include <stdio.h>
void swap_wrong(int x, int y)
{
            int tmp;
            printf("x : %p\n", &x); printf("y : %p\n", &y);
            tmp = x;
            x = y;
            y = tmp;
}

void swap(int* px, int* py)
{
            int tmp;
            printf("px = %p\n", px);
            printf("py = %p\n", py);
            tmp = *px;
            *px = *py;
            *py = tmp;
}

void main()
{
            int x = 3, y = 5;
            printf("Adresa promenljive x je %p\n", &x);
            printf("Vrednost promenljive x je %d\n", x);
            printf("Adresa promenljive y je %p\n", &y);
            printf("Vrednost promenljive y je %d\n", y);
            swap_wrong(x, y);
            printf("Posle swap_wrong:\n");
            printf("Vrednost promenljive x je %d\n", x);
}
```

POKAZIVAČ KAO ARGUMENT FUNKCIJE – PRIMER2

Zadatak. Napisati funkciju koja istovremeno vraća dve vrednosti - količnik i ostatak dva data broja

```
#include <stdio.h>
void div_and_mod(int x, int y, int* div, int* mod)
{
	printf("Kolicnik postavljam na adresu : %p\n", div);
	printf("Ostatak postavljam na adresu : %p\n", mod);
	*div = x / y;
	*mod = x % y;
}

void main()
{
	int div, mod;
	printf("Adresa promenljive div je %p\n", &div);
	printf("Adresa promenljive mod je %p\n", &mod);
	div_and_mod(5, 2, &div, &mod);
	printf("Vrednost promenljive div je %d\n", div);
	printf("Vrednost promenljive mod je %d\n", mod);
}
```

PRIMER. VEZA IZMEĐU POKAZIVAČA I NIZOVA

Naredni primer demonstrira korišćenje pokazivača za manipulaciju niza na koji pokazuje

```
#include <stdio.h>
void print_array(int* pa, int n);
void main()
{
          int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
          int num_of_elements = sizeof(a)/sizeof(int);
          int* pa;
          printf("Niz a : %p\n", a);
          printf("Adresa prvog elementa niza a (&a[0]) : %p\n", &a[0]);
          pa = a;
          printf("Pokazivac pa ukazuje na adresu : %p\n", pa);
          printf("a + 3 = %p\n", a + 3); printf("&a[3] = %p\n", &a[3]);
          printf("pa[5] = %d\n", pa[5]); printf("*(pa + 5) = %d\n", *(pa+5));

          printf("sizeof(a) = %ld\n", sizeof(a));
          printf("sizeof(pa) = %ld\n", sizeof(pa));
          print_array(a, num_of_elements); print_array(pa, num_of_elements);
}

void print_array(int* pa, int n)
{
          int i;
          for (i = 0; i<n; i++)
          printf("%d ", pa[i]);
          putchar('\n');
}
```

PRIMER. NIZOVI I POKAZIVAČI

U narednom primeru je pokazano korišćenje nitova 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];
          *(p_fniz+2)=*(p_fniz-2)+ *(p_fniz+1); /*
niz[4]=fniz[0]+fniz[3] */
          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" .

PRIMER. POKAZIVAČI NA FUNKCIJE

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

```
#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;
}

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));
}
```

PRIMER. NIZOVI POKAZIVAČA NA STRINGOVE

Primer ilustruje štampanje uporedne table sa nazivima cifara na srpskom i engleskom jeziku korišćenjem običnih 2D nizova karaktera i nizova pokazivača na karaktere

```
#include <stdio.h>

char sbroj[10][10]={"nula","jedan","dva","tri","cetiri","pet",
          "sest","sedam","osam","devet"};

char *p_ebroj[10]={"zero","one","two","three","four","five",
          "six","seven","eight","nine"};

void main(void)
{
          int i;
          for(i=0; i<=9; ++i)
          printf("%d %10s %10s\n", i, sbroj[i], p_ebroj[i]);
}
```

U ovom primeru cilj je odštampati uporednu tabelu sa nazivima cifara na srpskom i na engleskom jeziku. Za nazive na srpskom jeziku deklarisana je tabela sbroj, fiksnih dimenzija 10x10 (deset cifara i maksimalna dužina reči naziva je deset slova). Istovremeno je izvršena inicijalizacija ove matrice. Kao što se vidi, nije potrebno dodeliti svakom članu pojedinačno vrednost, već je na intuitivan način data lista od deset reči pri čemu se svaka nalazi unutar navodnika i međusobno su odvojene zarezima.

Sledi deklaracija niza pointera char *p_ebroj[10] koji je inicijalizovan - članovima niza dodeljeni su nazivi cifara na engleskom. Ovaj niz sadrži pokazivače na stringove koji su dati pri inicijalizaciji, koja se ni po čemu u sintaksi ne razlikuje od matrice sa srpskim nazivima. Iako su oba načina deklarisanja char indeksnih promenljivih potpuno ispravna, rutina u C programiranju podrazumeva upotrebu drugog načina tj. deklaracije niza pokazivača.

Sa aspekta utroška memorije niz pokazivača je u prednosti jer se zauzima prostor koji tačno popunjavaju reči različite dužine. Fiksnim dimenzionisanjem se obezbeđuje pravougaona matrica u kojoj ostaju upražnjene ali potrošene memorijske lokacije. Kada priroda problema nije takva da je potreban direktan pristup pojedinačnim slovima tabele (podrazumeva se upotreba dva indeksa: za reč i za slovo), treba koristiti niz pokazivača. Ista diskusija važi za poređenje deklaracije stringa kao niza karaktera fiksne dimenzije i deklaracije pokazivača na string. Iako je dimenzionisana kao dvodimenziona, za matricu sbroj pri štampi koristi se samo jedan indeks, što je u skladu sa već ranije pokazanom osobinom da su imena nizova/matrica istovremeno pokazivači na njihov tip.

PRIMER. POKAZIVAČ NA STRING KAO ARGUMENT FUNKCIJE

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

```
#include <stdio.h>

int duzina(char *p_string);          /* prototip funkcije */

void main(int argc, char *argv[])    /* argumenti glavnog
programa
                                        argc   - broj
argumenata
                                        argv[] -
argumenti       */
{
      char *p_tekst={"Ovo je znakovni niz\n"};
      int i;

/* stampanje argumenata unetih u komandnoj liniji */
      for(i=1; i<argc; i++)
            printf("argument%d %s\n", i, argv[i]);

/* poziv funkcije za odredjivanje duzine stringa -
parametar pointer */
      i=duzina(p_tekst);
      printf("duzina stringa je %d", i);
}

int duzina(char *p_string)           /* funkcija */
{
      int l=0;                       /* inicijalizacija
brojaca */
      char *p;                       /* pomocni pointer */
      p=p_string;                    /* pointer na prvo
slovo */

      while(*p != '\0')              /* do kraja stringa */
      {
            ++l;                     /* uvecanje brojaca */
            ++p;                     /* pointer na sledece
slovo */
      }
      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 deklarise 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'.

Argumenti glavne funkcije se prenose programu prilikom pokretanja programa, u komandnoj liniji. Prvi parametar generiše se interno u programu i daje podatak o tome koliko parametara je prosleđeno programu. U nizu pokazivača, koji je drugi parametar, su smešteni parametri sa komandne linije. Oni su svi tipa char i obično se koriste u programu kao indikatori za pokretanje odnosno isključivanje pojedinih podopcija programa. U ovom primeru vrši se samo njihovo štampanje. Od ostalih detalja treba izdvojiti način deklarisanja stringa preko pokazivača p_tekst.

PRIMER. POKAZIVAČ NA FUNKCIJU

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

```
#include <stdio.h>

/* prototipovi funkcija */
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 *);    /* niz pointera na funkciju */

/* dodela vrednosti adresa funkcija pokazivacima */
      pointer[0]=zbir;
      pointer[1]=razlika;

/* pozivanje funkcija posredno preko pokazivaca */
      r_zbir=(*pointer[0])(&a, &b);
      r_razlika=(*pointer[1])(&a, &b);

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

/* funkcija za izracunavanje zbira dva integera */
int zbir(int *a, int *b)
{
      return *a+*b;
}

/* funkcija za izracunavanje razlike dva integera */
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 integera. Treba primetiti način na koji se inicijalizuju pokazivači na funkcije; sledi da su nazivi funkcija istovremeno i pokazivači.

NIZ POKAZIVAČA – PROST PRIMER ZA SORTIRANJE LINIJA TEKSTA

Primer čita tekst sa standardnog ulaza, sortira linije u alfabetskom poretku i štampa ih na standardni izlaz. Funkcija koju koristimo radi sa nizom pokazivača na stringove kao argumentom

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
char *getline(void);              // Reads a line of text
int str_compare(const void *, const void *);
#define NLINES_MAX 1000           // Maximum number of text lines.
char *linePtr[NLINES_MAX];        // Array of pointers to char.
int main( )
{
  // Read lines:
  int n = 0;                      // Number of lines read.
  for (  ; n < NLINES_MAX && (linePtr[n] = getline( )) != NULL; ++n );

  if ( !feof(stdin) )             // Handle errors.
  {
    if ( n == NLINES_MAX )
      fputs( "sorttext: too many lines.\n", stderr );
    else
      fputs( "sorttext: error reading from stdin.\n", stderr );
  }
  else                            // Sort and print.
  {
 char **p;
    qsort( linePtr, n, sizeof(char*), str_compare );     // Sort.
    for ( **p = linePtr; p < linePtr+n; ++p )       // Print.
      puts(*p);
  }
  return 0;
```

NIZ POKAZIVAČA – PROST PRIMER ZA SORTIRANJE LINIJA TEKSTA

Primer čita tekst sa standardnog ulaza, sortira linije u alfabetskom poretku i štampa ih na standardni izlaz. Funkcija koju koristimo radi sa nizom pokazivača na stringove kao argumentom

```
#define LEN_MAX 512                             // Maximum length of a line.

char *getline( )
{
  char buffer[LEN_MAX], *linePtr = NULL;
  if ( fgets( buffer, LEN_MAX, stdin ) != NULL )
  {
    size_t len = strlen( buffer );

    if ( buffer[len-1] == '\n' )                // Trim the newline character.
      buffer[len-1] = '\0';
    else
      ++len;

    if ( (linePtr = malloc( len )) != NULL )  // Get enough memory for the line.
      strcpy( linePtr, buffer );          // Copy the line to the allocated block.
  }
  return linePtr;
}

int str_compare( const void *p1, const void *p2 )
{
  return strcmp( *(char **)p1, *(char **)p2 );
}
```

PRIMER. IMPLEMENTACIJA SELECTION SORT-A KORIŠĆENJEM POKAZIVAČA

Sledeći primer demonstrira referenciranje elemenata niza korišćenjem imena niza I korišćenjem pokazivača u clju implementacije selection sort algoritma za sortiranje nizova

```
// This program puts values into an array, sorts the values into
// ascending order and prints the resulting array.
#include <stdio.h>

#define SIZE 10

void selectionSort( int * const, const int ); // prototype
void swap( int * const, int * const ); // prototype

int main()
{
   int i,j;
   int a[ SIZE ] = { 2, 6, 4, 8, 10, 12, 89, 68, 45, 37 };

   printf("Data items in original order\n");

   for (i = 0; i < SIZE; i++ )
    printf("%4d", a[i]);

   selectionSort( a, SIZE ); // sort the array

   printf("\nData items in ascending order\n");

   for (j = 0; j < SIZE; j++ )
      printf("%4d", a[j]);

   printf("\n");
   return 0; // indicates successful termination
} // end main
```

PRIMER. IMPLEMENTACIJA SELECTION SORT-A KORIŠĆENJEM POKAZIVAČA

Sledeći primer demonstrira referenciranje elemenata niza korišćenjem imena niza I korišćenjem pokazivača u clju implementacije selection sort algoritma za sortiranje nizova

```
void selectionSort( int * const array, const int size )
{
   int i,index,smallest; // index of smallest element
   // loop over size - 1 elements
   for (i = 0; i < size - 1; i++ )
   {
      smallest = i; // first index of remaining array
      // loop to find index of smallest element
      for (index = i + 1; index < size; index++ )

         if ( array[ index ] < array[ smallest ] )
            smallest = index;
      swap( &array[ i ], &array[ smallest ] );
   } // end if
} // end function selectionSort
void swap( int * const element1Ptr, int * const element2Ptr )
{
   int hold = *element1Ptr;
   *element1Ptr = *element2Ptr;
   *element2Ptr = hold;
}
```

PRIMER. KOPIRANJE STRINGOVA KORIŠĆENJEM NIZA I POKAZIVAČKE NOTACIJE

Sledeći primer ilustruje vezu i razmenjivost pokazivača i nizova kod funkcija koje vrše kopiranje sadržaja stringa. Funkcije koje koristimo su označen kao copy1() i copy2()

Obe funkcije kopiraju string u niz karaktera. Nakon poređenja prototipova funckcija copy1 i copy2, stvara se utisak da su one identične (zbog razmenjivosti pokazivača i nizova). Obe funkcije obavljaju isti zadatak ali se mogu primetiri razlike u implementaciji.

```
#include <stdio.h>

void copy1( char *, const char * ); // prototype
void copy2( char *, const char * ); // prototype

int main()
{
   char string1[ 10 ];
   char *string2 = "Hello";
   char string3[ 10 ];
   char string4[] = "Good Bye";

   copy1( string1, string2 ); // copy string2 into string1
   printf("string1 = %s\n",string1);

   copy2( string3, string4 ); // copy string4 into string3
   printf("string3 = %s\n",string3);
   return 0; // indicates successful termination
} // end main
```

PRIMER. KOPIRANJE STRINGOVA KORIŠĆENJEM NIZA I POKAZIVAČKE NOTACIJE

Sledeći primer ilustruje vezu i razmenjivost pokazivača i nizova kod funkcija koje vrše kopiranje sadržaja stringa. Funkcije koje koristimo su označen kao copy1() i copy2()

Obe funkcije kopiraju string u niz karaktera. Nakon poređenja prototipova funckcija copy1 i copy2, stvara se utisak da su one identične (zbog razmenjivosti pokazivača i nizova). Obe funkcije obavljaju isti zadatak ali se mogu primetiri razlike u implementaciji.

```
// copy s2 to s1 using array notation
void copy1( char * s1, const char * s2 )
{
 int i;
   // copying occurs in the for header
   for (i = 0; ( s1[ i ] = s2[ i ] ) != '\0'; i++ )
      ; // do nothing in body
} // end function copy1

// copy s2 to s1 using pointer notation
void copy2( char *s1, const char *s2 )
{
   // copying occurs in the for header
   for ( ; ( *s1 = *s2 ) != '\0'; s1++, s2++ )
      ; // do nothing in body
}
```

ZADACI ZA SAMOSTALAN RAD

U nastaku su dati zadaci koji služe za dodatno vežbanje

Napisati funkciju f sa povratnim tipom void koja ima samo jedan argument tipa int i vraća ga udvostručenog.

Napisati funkciju f sa povratnim tipom void koja ima tačno jedan argument, a koja prosleđenu celobrojnu vrednost vraća sa promenjenim znakom.

Napisati funkciju sa tipom povratne vrednosti void koja služi za izračunavanje zbira i razlike svoja prva dva argumenta.

Napisati program koji traži od korisnika da unese deset brojeva koji će biti zapamćeni u nizu. Nakon unosa program treba da sortira unete brojeve po veličini, od najvećeg ka najmanjem, i da ih prikaže u konzoli. Kreirati metodu koja sortira uneti niz. Koristiti bubble sort algoritam za sortiranje niza.

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.

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

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.

Napisati funkciju koja korišćenjem pokazivača kao argumenta funkcije ispituje da li je string palindrom.

Zaključak

12

REZIME

Na osnovu utvrđenog 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.

Kao i pri radu sa bilo kojom drugom promenljivom ili konstantom, neophodno je prvo deklarisati pokazivač pre njegove prve upotrebe. Jedina razlika između deklarisanja obične promenljive i pokazivača je u tome što deklaracija pokazivača sadrži i znak asterisk (*) koji stoji između tipa podatka i naziva promenljive. Ono što treba naglasiti je da se tip podatka pri deklaraciji pokazivača ne odnosi na tip podatka njegove vrednosti kao što je slučaj kod promenljivih i konstante koje sadrže neku vrednost a ne adresu (što je slučaj kod pokazivača). Stvarni tip podatka svih pokazivača bilo da pokazuju na tip int, float , char ili neki drugi, je heksadecimalni broj koji predstavlja memorijsku adresu. Stoga, tip podatka pokazivača se ustvari odnosi na tip podatka druge promenljive (ili konstante) čija memorijska adresa je vrednost pokazivača. Drugim rečima, vrednost pokazivačke promenljive tipa int mora biti adresa celobrojne promemenljive ili konstante, tj vrednost pokazivačke promenljive na tip float mora biti adresa realne promenljive ili konstante, itd.

Uvek treba eksplicitno dodeliti neku vrednost pokazivaču pre njegovog korišćenja, jer se u suprotnom mogu javiti neke ozbiljne greške pri izvršavanju programa. Ukoliko ste sigurni da je vreme da u nekom delu koda dodelite pokazivaču vrednost memorijske adrese neke druge promenljive ili konstante, onda koristite adresni operator koji stoji uz ime željene promenljive ili konstante. Medjutim, ukoliko je u kodu još rano da se zna koja adresa treba biti dodeljena pokazivaču, onda je moguće pokazivaču dodeliti vrednost NULL, što je konstanta definisana u nekoliko standardnih biblioteka C jezika. 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.

O POKAZIVAČIMA, NIZOVIMA I FUNKCIJAMA

Možemo da izvedemo sledeći zaključak:

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 funkciju 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.