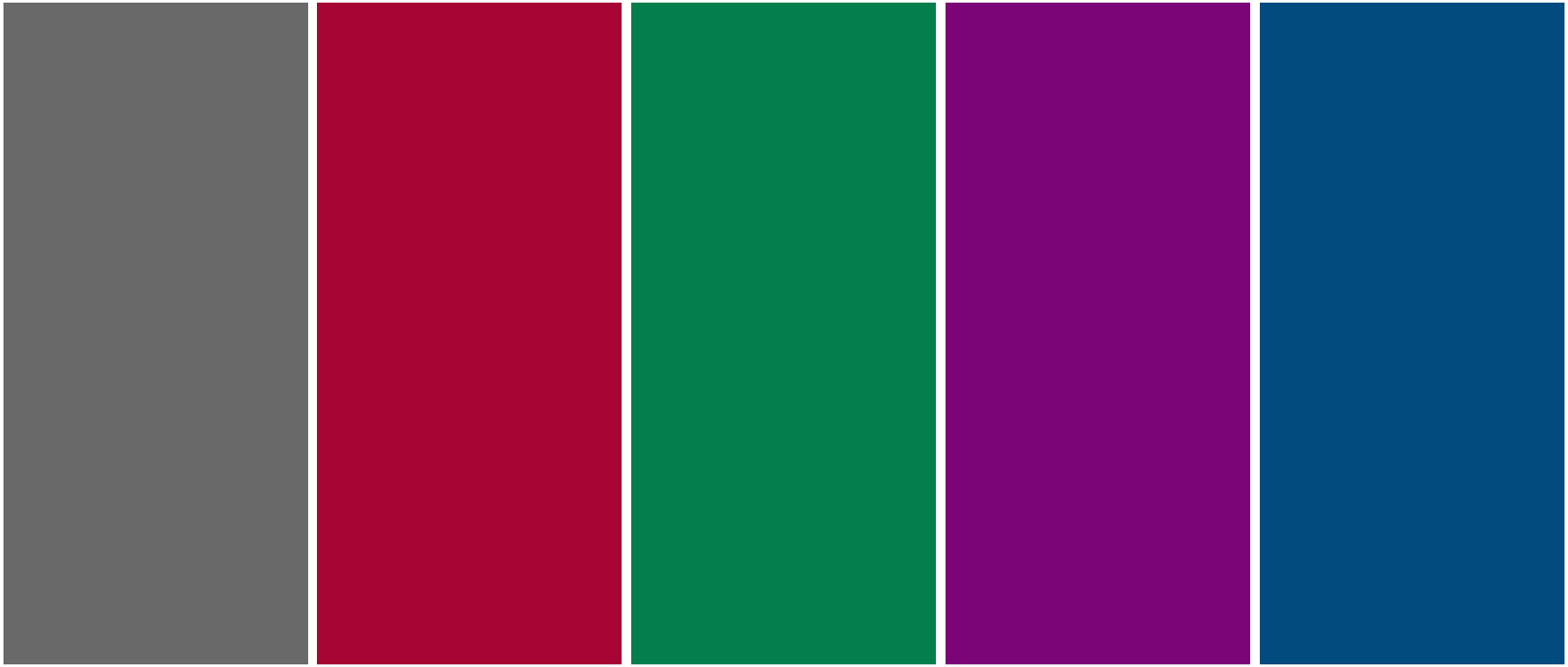
Lekcija 03

Nizovi i stringovi

Miljan Milošević

NIZOVI I STRINGOVI

	01	02	03	04
Uvod	Osnovi o nizovima	Nizovi i funkcije	Nizovi i pokazivači	Uvod u sortiranje nizova



NIZOVI I STRINGOVI

05	06	07	08	09
Višedimenzionalni nizovi	**C-stringovi**	**Funkcije za rad sa C-stringovima**	**Argumenti komandne linije**	**Vežbe**

- *Formatirani ulaz stringova*
- *Konverzija stringova i brojeva*

UVOD

Ova lekcija treba da ostvari sledeće ciljeve:

U okviru ove lekcije studenti se upoznaju sa osnovnim pojmovima u vezi nizova i stringova u programskom jeziku C:

- ❑ Deklaracija, inicijalizacija i korišćenje jednodimenzionalnih i višedimenzionalnih nizova.
- ❑ Nizovi i funkcije
- ❑ Pokazivači i nizovi
- ❑ Nizovi karaktera u C-u (C-stringovi)
- ❑ Pokazivači na nizove karaktera
- ❑ Funkcije za rad sa C-stringovima

Promenljive sa kojima smo radili do sada mogle su istovremeno da čuvaju samo jednu vrednost. Nizovi, međutim, omogućavaju da koristimo jednu promenljivu koja će čuvati ogroman broj podataka. Ovi podaci su smešteni na uzastopnim memorijskim lokacijama, određenim indeksima, gde prvi član niza ima indeks 0 dok se indeks zatim povećava za jedan za svaki uzastopni član niza.

Korišćenje jedne promenljive za niz, koja čuva 100 vrednosti, ima mnogo veće pogodnosti nego korišćenje 100 različitih promenljivih gde svaka može da čuva samo jednu vrednost. Osim toga, mnogo je lakše pratiti promene jedne u odnosu na 100 različitih promenljivih. Najveća pogodnost korišćenja nizova je ta što se mogu koristiti petlje (ciklusi) da bi se pristupilo svakom uzastopnom elementu niza.

Kada je u pitanju rad sa karakterima, često se dešava da se korisnički ulaz u program sastoji iz više od jednog karaktera. Pojedinačni karakteri se takođe mogu organizovati u niz. Nizovi karaktera obično sadrže null karakter kao poslednji član niza. Takvi nizovi karaktera koji na kraju sadrže null karakter se nazivaju „C-stringovi“. Na kraju lekcije će biti opisane funkcije koje se koriste za C-stringove.

Osnovi o nizovima

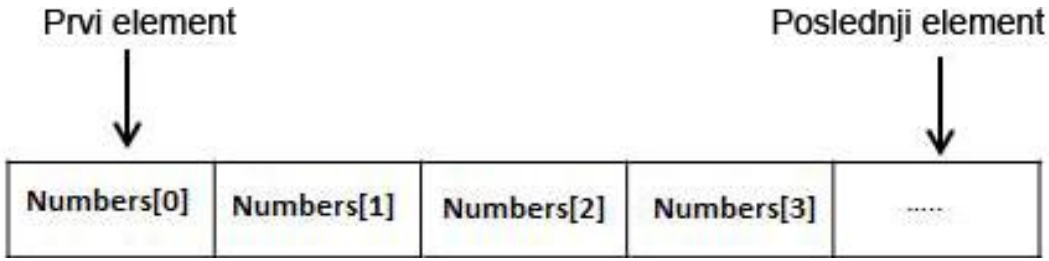
Niz, inicijalizacija, deklaracija, upotreba

01

OPIS NIZA

Niz je struktura podatka koja služi da se u njoj smesti kolekcija elemenata istog tipa i fiksne veličine

C programski jezik kao i ostali jezici u sebi ima ugrađenu strukturu podatka koja se zove niz i koja služi da se u njoj smesti sekvencijalna kolekcija elemenata istog tipa i fiksne veličine. Niz služi za smeštanje kolekcije podataka, ali je često mnogo jednostavnije razmišljati o nizu kao kolekciji promenljih istog tipa. Članovi niza zauzimaju susedne memorijske lokacije (Slika-1). Adresa sa najmanjom vrednošću (najniža adresa) odgovara prvom članu niza dok najviše adrese odgovaraju poslednjem članu niza.



Slika-1 Predstavljanje niza u memoriji računara

Deklaracija niza u C-u ima sledeći oblik (prvo se navodi tip promenljive, zatim ime promenljive, a u uglastim zagradama [ ] se navodi broj elemenata niza – tj, veličina niza):

```
type arrayName [ arraySize ];
```

Ovo je takozvani jednodimenzionalni niz. Veličina niza (arraySize) mora biti pozitivna celobrojna vrednost (veća od nule) dok tip (type) može biti bilo koji validan C/C++ tip podatka (primitivan ili složen). Na primer, da bi se deklarisao niz balance koji se sastoji iz 10 elemenata tipa double, koristi se sledeći izraz:

```
double balance[10];
```

INICIJALIZACIJA NIZA

Inicijalizacija niza je dodeljivanje vrednosti elementima niza u istom iskazu u kome se vrši deklaracija niza

Inicijalizacija niza se može izvršiti ili član po član, ili je moguće koristiti sledeći izraz:

```
double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

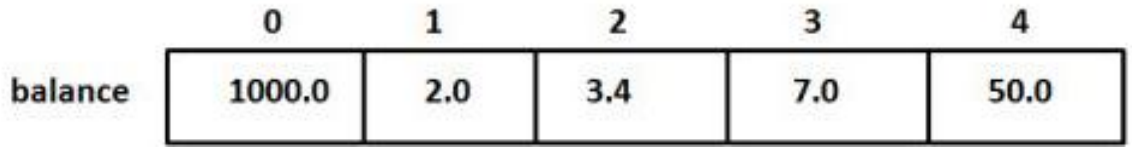
pri čemu broj vrednosti unutar vitičastih zagrada { } ne sme biti veći od maksimalnog broja elemenata niza koji smo naveli u okviru uglastih zagrada [ ].
Takođe je moguće u toku deklaracije izostaviti maksimalnu dimenziju, a samo u okviru vitičastih zagrada inicijalizovati članove niza, što će automatski setovati maksimalnu veličinu niza na broj unetih elemenata. Stoga, ukoliko napišemo:

```
double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};
```

kreiraće se potpuno identičan niz kao u prethodnom primeru, gde je niz imao 5 elemenata. U nastavku je dat primer dodele vrednosti odgovarajućem članu niza, kojem naravno pristupamo preko indeksa:

```
balance[4] = 50.0;
```

Prethodni izraz dodeljuje petom članu niza vrednost 50.0. Kao i u Java-i, indeks niza počinje sa nulom (a[0]) dok poslednji član niza ima indeks N-1, gde je sa N označena dimenzija niza. Na narednoj slici je grafički prikazan niz balance iz prethodnog primera:



Slika-2 Predstavljanje niza balance u memoriji računara

OPERATOR SIZEOF() I NIZOVI

Operator sizeof koristimo da bi smo odredili broj elemenata niza u slučaju kada dimenzija niza nije navedena u toku inicijalizacije

Operator sizeof() može biti korišćen kod nizova, i kao rezultat vraća ukupnu veličinu memorije koja je rezervisana za niz:

```
int anArray[] = { 0, 1, 2, 3, 4 }; // deklaracija niza od 5 elemenata
printf("%d",sizeof(anArray)); // stampa 20 (5 elements * 4 bytes each)
```

U programskom jeziku C ne postoji direktan način da se ispita kolika je veličina niza ali je moguće to utvrditi korišćenjem operatora sizeof na sledeći način:

```
int nElements = sizeof(anArray) / sizeof(anArray[0]);
```

S ozbirom da svi elementi niza imaju istu veličinu (pošto se radi o istom tipu podatka), deljenjem ukupne količine memorije niza sa veličinom memorije koja odgovara jednom elementu niza mi ustvari dobijamo broj elemenata. Pri tome treba koristiti element sa indeksom 0 s obzirom da niz koji je deklarisan mora imati bar jedan element.

NIZOVI I NABROJIVI TIP

Česta je praksa da se indeksi niza označe vrednostima nabrojivog tipa kada želimo da opišemo šta ta konkretna promenljiva znači

Jedan od većih problema sa indeksacijom niza korišćenjem celobrojnih vrednosti je taj što indeksi ne obezbeđuju informaciju o tome šta ta konkretna promenljiva znači. Pretpostavimo da imamo niz od 5 studenata:

```
const int nNumberOfStudents = 5;
int anTestScores[nNumberOfStudents];
anTestScores[2] = 76;
```

Postavlja se pitanje šta predstavlja član niza sa indeksom 2? Naravno, to nije do kraja jasno. Stoga, veoma je česta praksa kada je to moguće znati unapred, da se indeksi niza označe vrednostima nabrojivog tipa, kao u narednom primeru:

```
enum StudentNames
{
    KENNY, // 0
    KYLE, // 1
    STAN, // 2
    BUTTERS, // 3
    CARTMAN, // 4
    MAX_STUDENTS // 5
};

int anTestScores[MAX_STUDENTS]; // allocate 5
integers
anTestScores[STAN] = 76;
```

Na ovaj način, korišćenjem prethodnog dela koda, postaje jasnije šta ustvari predstavljaju odgovarajući članovi niza.

Može se primetiti da je dodatna nabrojiva vrednost pod imenom MAX_STUDENTS dodata u nabrojivi tip StudentNames da bi imali podatak o maksimalnoj veličini niza. Ovo je korisno za bolje dokumentovanje problema, a i zato jer će niz biti automatski proširen ukoliko je u nabrojivi tip dodata još neka nabrojiva konstanta:

```
enum StudentNames
{
    KENNY, // 0
    KYLE, // 1
    STAN, // 2
    BUTTERS, // 3
    CARTMAN, // 4
    WENDY, // 5
    MAX_STUDENTS // 6
};

int anTestScores[MAX_STUDENTS]; // allocate 6
integers
anTestScores[STAN] = 76;
```

Treba imati na umu da ovaj trik funkcioniše samo u slučaju da se ručno ne promene vrednosti nabrojivog tipa, pa treba biti oprezan sa korišćenjem nizova i nabrojivih konstanti!

PRISTUPANJE ELEMENTU NIZA

Elementu niza se pristupa preko indeksa koji stoji u uglastim zagradama uz ime niza

```
#include <stdio.h>

int main ()
{
   int n[ 10 ]; /* n is an array of 10 integers */
   int i,j;

   for ( i = 0; i < 10; i++ )
   {
      n[ i ] = i + 100; /* set element at location i
to i + 100 */
   }

   /* output each array element's value */
   for (j = 0; j < 10; j++ )
   {
      printf("Element[%d] = %d\n", j, n[j] );
   }

   return 0;
}
```

Kao i u Java-i, elementu niza se pristupa preko indeksa koji stoji u uglastim zagradama uz ime niza:

```
double salary = balance[9];
```

Prethodni izraz će vrednost desetog elementa niza balance dodeliti promenljivoj salary. Na levoj strani je dat prost primer deklarisanja, inicijalizacije i pristupa elementima niza.

Nakon izvršavanja prethodnog programa, na ekranu ce biti ispisan sledeći rezultat:

```
Element[0] = 100
Element[1] = 101
Element[2] = 102
Element[3] = 103
Element[4] = 104
Element[5] = 105
Element[6] = 106
Element[7] = 107
Element[8] = 108
Element[9] = 109
```

NAJČEŠĆA GREŠKA PRI RADU SA NIZOM: IZLAZAK IZ OPSEGA

Izlazak iz opsega je jedna od težih grešaka za otkrivanje a može da izazove ozbiljne probleme

Jedna od velikih začkoljica pri radu sa petljama i nizovima je da se proveri da li se petlja izvršava tačno odgovarajući broj puta. Pri tome se može izaći iz opsega niza za tačno jedan član, jer se želi pristupiti elementu preko indeksa koji je veći nego što je ukupna veličina niza. Uzmimo u obzir sledeći program:

```
const int nArraySize = 5;
int anArray[nArraySize ] = { 6, 8, 2, 4, 9 };
int nMaxValue = 0;
for (int nIndex = 0; nIndex <= nArraySize; nIndex++)
    if (anArray[nIndex] > nMaxValue)
        nMaxValue = anArray[nIndex];
```

Problem sa prethodnim programom je taj što je uslovni izraz u okviru for konstrukcije netačan. Deklarisani niz ima 5 elemenata, tako da indeksi idu u opsegu od 0 do 4. Međutim, petlja ide od 0 do 5. Kao posledica, u poslednjem ciklusu for petlje izvršiće se sledeća sekvenca koda:

```
if (anArray[5] > nMaxValue)
    nMaxValue = anArray[5];
```

ali član anArray[5] niza nije definisan! Ovo može da izazove raličite problem, a najverovatnije je da će anArray[5] imati neku nedodeljenu vrednost ili beskonačno veliki broj. Kao posledica greške u pisanju koda promenljivoj nMaxValue će biti dodeljena pogrešna vrednost.

Ono što je još veći problem kod izlaska iz opsega je dodela vrednosti članu anArray[5], što može dovesti do izmene neke druge promenljive (ili dela promenljive). Ovakve greške u programu su jedne od težih za otkrivanje a mogu da izazovu ozbiljne probleme!

Nizovi i funkcije

Nizovi, funkcije, argumenti, prosledjivanje po adresi

02

PROSLEĐIVANJE NIZA FUNKCIJI

Postoje tri načina da se niz kroz listu argumenata prosledi funkciji: kao pokazivač, kao dimenzionisani niz i kao nedimenzionisani niz

Da bi se niz prosledio funkciji kroz listu argumenata, postoje tri načina na koja se to može uraditi. Sva tri načina su veoma slična međusobno, i proizvode isti rezultat, zato što svi slučajevi šalju poruku kompajleru da se radi sa pokazivačem na celobrojnu vrednost. Na sličan način je moguće proslediti i višedimenzionalne nizove što će biti pokazano u nastavku.

- Prosleđivanje niza korišćenjem pokazivača kao formalnog parametra. Već smo uveli pokazivače, ali više detalja o pokazivačima biće u narednom predavanju.

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

- Formalni parametar kao dimenzionisani niz, kao što je dato u nastavku:

```
void myFunction(int param[10])
{
.
.
.
}
```

- Formalni parametar kao nedimenzionisani niz:

```
void myFunction(int param[])
{
.
.
.
}
```

PRIMER KORIŠĆENJA NIZOVA U FUNKCIJI

Kada se ime niza prosledi funkciji kroz listu argumenata prosleđivanje se vrši po adresi

Sada pogledajmo sledeći primer,tj. funkciju getAvarage() koja kao argumente uzima nedimenzionisani niz arr i veličinu niza size, a kao rezultat vraća srednju vrednost članova niza:

```
double getAverage(int arr[], int size)
{
  int    i;
  double avg;
  double sum;

  for (i = 0; i < size; ++i)
  {
    sum += arr[i];
  }

  avg = sum / size;

  return avg;
}
```

Da bi smo pozvali funkciju iz glavnog programa koristimo kod koji se nalazi ispod.
Nakon izvršavanja prethodnih linija koda, dobija se sledeći rezultat:

```
Average value is: 214.400000
```

```
#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 ", avg );

   return 0;
}
```

Kao što možete videti iz prethodnog primera, nije neophodno navesti dimenziju niza u listi formalnih parametara jer C ne proverava granice formalnih parametara (isto je u Java-i).

Nizovi i pokazivači

Nizovi, pokazivači, ime niza kao konstantan pokazivač

03

POKAZIVAČI NA NIZOVE

Ime niza ustvari predstavlja pokazivač na prvi element niza

Pokazivače smo već uveli u 1. lekciji, a i u prethodnoj lekciji je bilo reči o vezi između nizova i pokazivača. Sada ćemo otići korak dalje i malo opširnije opisati njihovu vezu. Naime, ime niza ustvari predstavlja pokazivač na prvi element niza. Stoga, u sledećoj deklaraciji:

```
double balance[50];
```

balance je pokazivač na &balance[0], što je ustvari adresa prvog elementa niza balance (prvi element je sa indeksom 0). Tako, u narednom delu koda, pokazivaču p ćemo dodeliti adresu prvog elementa niza balance:

```
double *p;
double balance[10];

p = balance;
```

U C-u (a i C++-u) je dozvoljeno koristiti imena niza kao konstantne pokazivače, i obrnuto. Stoga, korišćenje izraza *(balance + 4) je legalan način da se pristupi podatku 5. člana niza odnosno članu balance[4].
Treba znati da ako jednom pokazivaču p dodelite adresu niza onda ćete moći elementima niza da pristupite pomoću *p, *(p+1), *(p+2) i tako dalje.

UVOD U POKAZIVAČKU ARITMETIKU

Operacije uvećanja i umanjenja pokazivača na niz utiču na to da se pokazivač pomeri na naredni odnosno prethodni element niza

U programskom jeziku C moguće je koristiti binarne operatore + i - u cilju obavljanja aritmetičkih operacija nad pokazivačima. Na primer, može se modifikovati pokazivač na način da pokazuje na objekat koji je nekoliko memorijskih lokacija udaljen od objekta na koji pokazivač originalno pokazuje. Ovakve aritmetičke operacija sa pokazivačima su često pogodne kada se radi sa nizovima. Pretpostavimo da imamo pokazivač na realnu promenljivu (tipa double) i niz realnih brojeva (tipa double):

```
double *dPtr, dArr[5] = { 0.0, 1.1, 2.2, 3.3, 4.4 },   // Initialize an array and
*dPtr = dArr;                                          // a pointer to its first element.
```

U prethodnoj liniji smo postavili da pokazivač pokazuje na niz, što znači da će pokazivač u tom slučaju pokazivati na prvi član niza. Dodavanje jedne celobrojne vrednosti pokazivaču ili oduzimanje jedne celobrojne vrednosti od pokazivača dovodi do toga da pokazivač sada pokazuje na adresu koja je za sizeof(tip podatka) pomerena u odnosu na originalnu poziciju. Odnosno, ako pomeramo za više celobrojnih vrednosti, kompajler će automatski pomnožiti taj ceo broj sa veličinom objekta na koji se pokazivač originalno odnosti, kao što pokazuje sledeći primer:

```
int i = 0;                  // An index variable.
dPtr = dPtr + 1;            // Advance dPtr to the second element. Addends
dPtr = 2 + dPtr;            // can be in either order. dPtr now points to dArr[3].
printf( "%.1f\n", *dPtr );        // Print the element referenced by dPtr.
printf( "%.1f\n", *(dPtr -1) );  // Print the element before that, without
                                              // modifying the pointer dPtr.
```

Izraz dPtr = dPtr + 1; dodaje veličinu memorije jednog elemeta niza pokazivaču, pa će stoga dPtr pokazivati na sledeći element niza, dArr[1]. S obzirom da je da je dPtr deklarisan kao pokazivač na double, njegova vrednost će biti uvećana za sizeof(double). Izraz dPtr = dPtr + 1; ima isti efekat kao neki od sledećih operatora dodele ili inkrementiranja:

```
dPtr += 1;
++dPtr;
dPtr++;
```

Uvod u sortiranje nizova

Sortiranje, selection sort, qsort funkcija

04

UVOD U SORTIRANJE, PRIMENA FUNKCIJE SWAP()

Sortiranje nizova je premeštanje članova niza u cilju dobijanja uređenog poretka (rastućeg, opadajućeg, neopadajućeg ili nerastućeg)

Postoji veliki broj slučajeva u programerskim problemima gde je korisno izvršiti sortiranje nizova. Veliki broj algoritama (kao što je na primer algoritam pretraživanja niza, sa ciljem da se ispita da li se element nalazi u nizu ili ne) postaje prostiji za izvršavanje kada se radi sa sortiranim nizovima. Osim toga, sortiranje je veoma korisno za bolju preglednost i čitljivost, kao npr. u slučaju kada hoćemo da oštampamo imena studenata u alfabetskom poretku.

Sortiranje se generalno izvršava uzastopnim poređenjem parova elemenata niza, i uzastopnom zamenom mesta sve dok se ne ispuni željeni uslov. Način na koji se vrši poređenje odgovarajućeg para elemenata zavisi od vrste algoritma za sortiranje, kao i od redosleda u kome želimo da uredimo elemente niza (rastući ili opadajući redosled). Da bi smo zamenili mesta dvema članova niza, npr: v[i] i v[j] možemo da napišemo metodu swap():

```
/* swap: interchange v[i] and v[j] */
void swap(int v[], int i, int j)
{
          int temp;
          temp = v[i];
          v[i] = v[j];
          v[j] = temp;
}
```

SORTIRANJE KORIŠĆENJEM METODE SELEKCIJE: SELECTION SORT

Selection sort je verovatno najprostiji ali i najsporiji način za sortiranje niza

Postoji veliki broj načina da se izvrši sortiranje niza. U nastavku će biti opisan metod Selection Sort, koji je verovatno najprostiji način za razumevanje, ali s druge strane verovano jedan od najsporijih metoda za sortiranje niza.

Selection sort se odvija u sledećim koracima:
1) Počevši od člana sa indeksom 0, pretražiti ceo niz sa ciljem da se pronađe indeks čija je vrednost najmanja
2) Zameniti mesta najmanjeg člana sa vrednošću koja se nalazi u članu niza sa indeksom 0.
3) Ponoviti korake 1 i 2 počevši od člana sa sledećim indeksom.

Drugačije rečeno, cilj je da se pronađe najmanji element niza i da se njegova vrednost smesti na početak niza (u član sa indeskom 0). Zatim treba pretražiti ostatak niza (od 2. do N-tog člana), pronaći najmanji i smestiti ga na drugu poziciju (u član sa indeksom 1). Ovaj proces treba ponoviti dok se ne stigne do kraja niza.

U nastavku je dat primer rada algoritma za sortiranje niza koji je dimenzije 5. Neka je niz deklarisan na sledeći način:

{ 30, 50, 20, 10, 40 }

Pri korak je pronalaženje najmanjeg člana u podnizu od člana sa indeksom 0 do poslednjeg člana, što je vrednost 10 (element sa indeksom 3):
{ 30, 50, 20, **10**, 40 }
Zatim se vrši zamena vrednosti na pozicijama 0 i 3, sto znači da najmanji element prelazi na polje niza sa indeksom 0:
{ **10**, 50, 20, **30**, 40 }
Sada krećemo da pronađemo najmanji u podnizu od indeksa 1 do poslednjeg indeksa, što je vrednost 20 (polje sa indeksom 2):
{ 10, 50, **20**, 30, 40 }
pa vršimo zamenu mesta članovima sa indeksima 1 i 2:
{ 10, **20**, **50**, 30, 40 }
Sledeći korak je nalaženje najmanjeg člana u podnizu od indeksa 2 do poslednjeg indeksa, a to je broj 30:
{ 10, 20, 50, **30**, 40 }
pa vršimo zamenu mesta članovima sa indeksima 2 i 3:
{ 10, 20, **30**, **50**, 40 }
Poslednji korak je da se nađe najmanji u podnizu od indeksa 3 do 4, što je 40:
{ 10, 20, 30, 50, **40** }
pa zamenimo mesta vrednostima sa indeksom 3 i 4:
{ 10, 20, 30, **40**, **50** }
i dobijamo sortirani niz!
{ 10, 20, 30, 40, 50 }

SELECTION SORT - IMPLEMENTACIJA

Korišćenjem ugnježdene petlje vrši se naizmenično poređenje i eventualna zamena mesta članovima niza u cilju dobijanja uređenog poretka

U nastavku je dat deo koda u C-u za prethodno opisani algoritam sortiranja:

```
int nStartIndex, nCurrentIndex, nSize = 5;
int anArray[] = { 30, 50, 20, 10, 40 };

// Step through each element of the array
for (nStartIndex = 0; nStartIndex < nSize; nStartIndex++)
{
 // nSmallestIndex is the index of the smallest element
 // we've encountered so far.
 int nSmallestIndex = nStartIndex;

 // Search through every element starting at nStartIndex+1
 for (nCurrentIndex = nStartIndex + 1; nCurrentIndex < nSize; nCurrentIndex++)
 {
  // If the current element is smaller than our previously found smallest
  if (anArray[nCurrentIndex] < anArray[nSmallestIndex])
   // Store the index in nSmallestIndex
   nSmallestIndex = nCurrentIndex;
 }
 // Swap our start element with our smallest element
 swap(anArray, nStartIndex, nSmallestIndex);
}
```

Najkompleksniji deo prethodnog algoritma je ugnježdena for petlja. Spoljašnja petlja (nStartIndex) prolazi kroz svaki element, jedan po jedan. Unutrašnja petlja (nCurrentIndex) pronalazi najmanji element u podnizu polazeći od nStartIndex i postavlja indeks najmanjeg u *nSmallestIndex*. Najmanji element koji se nalazi na poziciji nSmallestIndex se zatim zamenjuje sa elementom koji se nalazi na početnom indeksu. Zatim se indeks u spoljašnjoj petlji uvećava za 1(nStartIndex) i ponavlja se postupak za sledeći manji podniz.

FUNKCIJA ZA SORTIRANJE NIZA QSORT()

U okviru standardne C biblioteke stdlib.h se nalazi funkcija qsort() koju možete koristiti za sortiranje nizova. Ova funkcija radi po principu brzog sortiranja odnosno „quick sort“

Standardna funkcija C biblioteke void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*)) vrši sortiranje niza i može se naći u standardnoj stdlib.h biblioteci. Deklaracija funkcije qsort() ima sledeći oblik:

```
void qsort(void *base, size_t nitems, size_t size, int (*compar)(const void *, const void*))
```

pri čemu su:
- base – pokazivač na prvi element niza koji će biti sortiran.
- nitems – broj elemenata niza
- size – veličina u bajtovima svakog pojedinačnog elementa niza.
- compar – funkcija koja vrši poređenje dva elementa. Ova funkcija je ustvari i kriterijum sortiranja niza.

U nastavku je dat primer korišćenja funkcije qsort():

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

int values[] = { 88, 56, 100, 2, 25 };

int cmpfunc (const void * a, const void * b)
{
   return ( *(int*)a - *(int*)b );
}

int main()
{
   int n;

   printf("Before sorting the list is: \n");
   for( n = 0 ; n < 5; n++ ) {
      printf("%d ", values[n]);
   }
```

FUNKCIJA ZA SORTIRANJE NIZA QSORT()

U okviru standardne C biblioteke stdlib.h se nalazi funkcija qsort() koju možete koristiti za sortiranje nizova. Ova funkcija radi po principu brzog sortiranja odnosno „quick sort“

```
   qsort(values, 5, sizeof(int), cmpfunc);

   printf("\nAfter sorting the list is: \n");
   for( n = 0 ; n < 5; n++ ) {
      printf("%d ", values[n]);
   }

  return(0);
}
```

Višedimenzionalni nizovi

Višedimenzionalni nizovi, 2D nizovi, funkcije i 2D nizovi

05

VIŠEDIMENZIONALNI I 2D NIZOVI

2D nizovi su najprostiji oblik višedimenzionalnih nizova i mogu se drugačije opisati kao nizovi nizova

U C-u (a i C++-u) višedimenzionalni nizovi se deklarišu na sledeći način:

```
type name[size1][size2]...[sizeN];
```

Tako, na primer, ako želimo da definišemo niz dimenzija 5x10x4 to možemo uraditi na sledeći način:

```
int threedim[5][10][4];
```

Najprostiji oblik višedimenzionalnih nizova su dvodimenzionalni nizovi (2D). 2D niz je u osnovi niz odnosno lista 1D nizova (niz 1D nizova). Osnovni način deklarisanja 2D nizova je dat u nastavku:

```
type arrayName [ x ][ y ];
```

gde type predstavlja tip, arrayName ime niza, dok se u uglastim zagradama navode dimenzije X i Y niza (X vrsta i Y kolona). 2D niz je ustvari tabela koja ima X vrsta i Y kolona. Proizvoljni 2D niz a, koji se sastoji iz 3 vrste i 4 kolona može biti grafički predstavljen na sledeći način:

	Kolona 0	Kolona 1	Kolona 2	Kolona 3
Vrsta 0	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Vrsta 1	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Vrsta 2	a[2][0]	a[2][1]	a[2][2]	a[2][3]

Slika-1 Grafički prikaz 2D niza

stoga je svaki element 2D niza određen indeksom vrste i indeksom kolone na sledeći način a[i][j], gde je A ime niza, a I i J odgovarajući indeksi vrste i kolone.

INICIJALIZACIJA I PRISTUPANJE ČLANOVIMA 2D NIZA

Inicijalizacija i pristupanje članovima 2D niza se ostvaruje na sličan način kao i kod 1D niza osim što ovde imamo jednu dimenziju više

❑ Inicijalizacija dvodimenzionalnog niza

Višedimenzionalni niz može biti inicijalizovan uređenom grupom vrednosti oivičenim vitičastim zagrada, i svaka grupa predstavlja jednu vrstu. U nastavku je dat primer niza koji ima 3 vrste i u svakoj vrsti vrednost za jednu od 4 kolona.

```
int a[3][4] = {
 {0, 1, 2, 3} ,   /*  initializers for row 0 */
 {4, 5, 6, 7} ,   /*  initializers for row 1 */
 {8, 9, 10, 11}   /*  initializers for row 2 */
};
```

Ugnježdene vitičaste zagrade koje uokviruju jednu vrstu su sasvim proizvoljne (ne moraju se pisati), tako da se prethodni primer može predstaviti i na sledeći način:

```
int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};
```

❑ Pristupanje članovima 2D niza

Pristupanje članu 2D niza se ostvaruje navođenjem indeksa vrste i kolone u uglastim zagradama nakon imena niza, na sledeći način:

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

pri čemu je iz 2D niza uzeta vrednost koja se nalazi u 3. vrsti i 4. koloni, i ta vrednost je dodeljena promenljivoj *val*. Kao što je opisano u prethodnom delu, moguće je generisati nizove sa više dimenzija, ali je najčešća praksa da se koriste 1D i 2D nizovi.

U nastavku je dat primer sa 2D nizovima gde je korišćena ugnježdena petlja za učitavanje i štampanje članova niza:

```
#include <stdio.h>

int main ()
{
   /* an array with 5 rows and 2 columns*/
   int a[5][2] = { {0,0}, {1,2}, {2,4},
{3,6},{4,8}};
   int i, j;

   /* output each array element's value */
   for ( i = 0; i < 5; i++ )
   {
      for ( j = 0; j < 2; j++ )
      {
         printf("a[%d][%d] = %d\n", i,j,
a[i][j] );
      }
   }
   return 0;
}
```

Nakon izvršavanja prethodnog koda dobija se sledeći rezultat:

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

VIŠEDIMENZIONALNI NIZOVI I FUNKCIJE

Višedimenzionalni nizovi se kroz listu argumenata prosleđuju funkciji na isti način kao i 1D nizovi

```
#include <stdio.h>

void display(int n[3][2]);

int main()
{
 int num[3][2] = {
        {3, 4},
        {9, 5},
        {7, 1}
 };

    display(num);
    return 0;
}

void display(int n[3][2])
{
    printf("Displaying Values: \n");
    for(int i = 0;  i < 3; ++ i)
           {
        for(int j = 0; j < 2; ++j)
                    {
            printf("%5d ",n[i][j]);
        }
                    printf("\n");
    }
}
```

Višedimenzionalni nizovi se mogu kao formalni parametri proslediti funkciji na sličan način kao kod 1D nizova. U prethodnom delu koda je dat primer korišćenja višedimenzionalnih nizova kao argumenata funkcije. Funkcija display() služi za štampanje elemenata niza.

Kao rezultat, dobija se sledeći izlaz:

```
Displaying Values:
3 4 9 5 7 1
```

C-stringovi

String konstante, stringovi,upotreba, memorija, pokazivači

06

STRING KONSTANTE I INICIJALIZACIJA STRINGOVA

String konstante se pišu između navodnika. U programskom jeziku C string je predstavljen kao jednodimenzionalni niz karaktera koji se završava null karakterom '\0'.

❑ String konstante

String literali ili string konstante u programskom jeziku C/C++ (kao i Javi) se pišu između navodnika "".Kao i u Java-i, string konstanta se može sastojati iz karaktera, znakova interpunkcije, escape i univerzalnih karaktera. U C-u je moguće preseći dugačku rečenicu korišćenjem literala koji su razdvojeni prazninama (whitespace). U nastavku je dat primer korišćenja string literala u C- u. Sve tri naredne konstrukcije daju isti rezultat:

```
"hello, dear"
"hello, \
dear"
"hello, " "d" "ear"
```

❑ Inicijalizacija stringova

U programskom jeziku C-string je predstavljen kao jednodimenzionalni niz karaktera koji se završava null karakterom '\0'. U narednoj liniji koda izvršena je deklaracija i inicijalizacija stringa "Hello". Treba imati na umu da je dimenzija niza greeting 6 jer je poslednje mesto rezervisano za null karakter '\0'.

```
char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
```

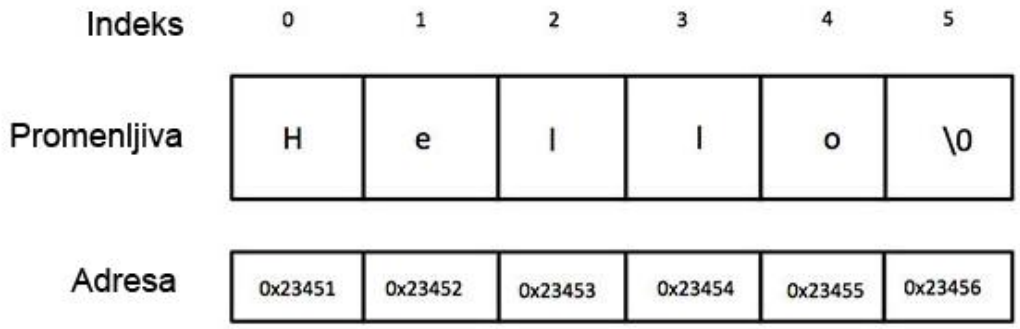
Na osnovu opisanog pravila koje važi za null karakter, moguće je prethodni string definisati korišćenjem sledećeg izraza:

```
char greeting[] = "Hello";
```

PREDSTAVLJANJE STRINGOVA U MEMORIJI I UPOTREBA

Niz karaktera je kao i običan niz u memoriji smešten na susednim lokacijama. Ime niza karaktera (stringa) takođe predstavlja pokazivač na prvi element niza

U nastavku je dat grafički prikaz segmenta memorije u kome je smestena reč "Hello":



Slika-1 Grafičko predstavljanje tekstualne promenljive (stringa) u memoriji računara

Prilikom definisanja string konstanti i literala nije neophodno staviti null karakter na kraj stringa pošto C kompajler to radi umesto vas prilikom inicijalizacije stringa. U nastavku je dat C primer koji štampa string "Hello" na standardni izlaz:

```
#include <stdio.h>
int main ()
{
   char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};
   printf("Greeting message: %s\n", greeting );
   return 0;
}
```

Nakon izvršenja prethodnog koda dobija se sledeći rezultat:

```
Greeting message: Hello
```

STRINGOVI KAO ARGUMENTI FUNKCIJE

C-stringovi (nizovi karaktera) se funkciji prosleđuju na isti način kao i obični nizovi

Stringovi se prosleđuju funkciji na isti način kao i nizovi. U nastavku je dat prost primer u programskom jeziku C koji demonstrira korišćenje stringa u funkciji kao argumenta funkcije.

```
#include <stdio.h>

void display(char s[]);

int main()
{
    char str[100];
    printf("Enter a string: ");
           gets(str);
    display(str);
    return 0;
}

void display(char s[])
{
    printf("You entered: %s\n",s);
}
```

Rezultat prethodnog koda je:

```
Enter a string: Programming is fun.
You entered: Programming is fun.
```

NIZOVI STRINGOVA

Nizovi stringova su 2D nizovi karaktera gde svaki string predstavlja sekvencu karaktera koja se završava null karakterom

Nizovi stringova u C-u su definisani kao 2D nizovi karaktera, pri čemu svaki string predstavlja sekvencu karaktera koja se završava null karakterom. Sledi primer niza stringova koji služi za dane u nedelji.

```
#include <stdio.h>
#include <string.h>
#define DAYS  7
#define MAX  10
void main()
{
          int j;
          char week[DAYS] [MAX] = {"Sunday", "Monday", "Tuesday",
                    "Wednesday", "Thursday", "Friday", "Saturday" };
          for(j=0;j<DAYS;j++)
          {
                    printf("%s\n",week[j]);
          }
}
```

Rezultat prethodnog programa:

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday

U prethodnom primeru, kao što smo već spomenuli, koristimo dvodimenzionalni niz karaktera koji u stvari predstavlja niz strigova. Ako posmatramo deklaraciju niza week, prvi indeks predstavlja ukupan broj stringova dok drugi indeks predstavlja maksimalnu dužinu reči. Vrednost konstante "MAX" je postavljena na 10 s obzirom da se reč "Wednesday" sastoji iz 9 karaktera dok je poslednje polje ostavljeno za null karakter koji označava kraj stringa (tako da imamo ukupno 10 mesta).

POKAZIVAČI I STRINGOVI

Pokaživači na stringove u C-u su deklarisani kao pokazivači na nizove karaktera. Pomoću funkcije scanf nije moguže čitati string korišćenjem pokazivačke promenljive

Pokaživači na stringove u C-u su deklarisani kao pokazivači na nizove karaktera, odnosno na prvi član niza. Kada se nekom pokazivaču na string dodeli neka tekstualna vrednost, automatski se na kraj tog stringa dodaje null karakter. U nastavku je dat primer korišćenja:

```
#include<stdio.h>
int main()
{
            char *ptr_mystring;

            ptr_mystring = "HELLO";
            printf("%s\n", ptr_mystring);

            return 0;
}
```

Pomoću funkcije scanf nije moguće čitati string korišćenjem pokazivačke promenljive. U nastavku je dat primer gde u cilju učitavanja nekog teksta koristimo definisani niz my_array, a zatim pokazivaču ptr_section2 dodeljujemo my_array, pa tek nakon toga korišćenjem pokazivača ptr_section2 štampamo vrednost teksta pomoću funkcije printf:

```
#include<stdio.h>
int main()
{
     char my_array[10];
     char *ptr_section2;

     printf("Type hello and enter\n");
    scanf("%s", my_array);
     ptr_section2 = my_array;
     printf("%s\n", ptr_section2);

     return 0;
}
```

Funkcije za rad sa C-stringovima

Funkcije za stringove, standardna biblioteka, stdio, string, stdlib

- *Formatirani ulaz stringova*
- *Konverzija stringova i brojeva*

07

ULAZNO/IZLAZNE FUNKCIJA ZA RAD SA TEKSTOM: GETS() I PUTS()

Funkcija gets() učitava celu liniju i smešta u bafer, dok funkcija puts() štampa tekst u konzolu i prebacuje kursor u novu liniju

Funkcija char *gets(char *s) učitava liniju sa standardnog ulaza stdin i smešta u buffer na koji pokazuje s (neka je za vas u ovom trenutku to niz, pošto će o pokazivačima biti više reči nešto kasnije) sve dok se ne stigne do kraja reda ili do kraja fajla (EOF).

Funkcija int puts(const char *s) štampa tekst (string) s i prebacuje kursor u novu liniju standardnog izlaza stdout.

```
#include <stdio.h>
int main( )
{
   char str[100];

   printf( "Enter a value :");
   gets( str );

   printf( "\nYou entered: ");
   puts( str );

   return 0;
}
```

Rezultat prethodnog koda (za uneti tekst: this is test):

```
Enter a value : this is test
You entered: this is test
```

FUNKCIJE STRING I STDLIB BIBLIOTEKE ZA RAD SA STRINGOVIMA

Funkcije za rad sa stringovima su smeštene u datoteci string.h, dok su funkcija za pretvaranje stringova u brojeve smeštene u datoteci stdlib.h

Ime funkcije	Svrha
strcpy(s1, s2);	Kopira string s2 u string s1.
strcat(s1, s2);	Dodaje string s2 na kraj stringa s1.
strlen(s1);	vraća dužinu stringa s1.
strcmp(s1, s2);	vraća: 0 ukoliko su s1 i s2 isti; broj manji od 0 ukoliko je s1<s2; broj veći od 0 ukoliko je s1>s2.
strchr(s1, ch);	vraća pokazivač na mesto prvog pojavljivanja karaktera ch u stringu s1.
strstr(s1, s2);	vraća pokazivač na prvo pojavljivanje stringa s2 u stringu s1.

Slika-1 Osnovne funkcije za rad sa tekstom kao nizom karaktera

Ime funkcije	Svrha
double atof(const char *str)	Konvertuje string *str u realni broj
int atoi(const char *str)	Konvertuje string *str u ceo broj

Slika-2 Osnovne funkcije za konverziju stringa u brojeve

U okviru programskog jezika C postoji veliki broj standardnih funkcija za rad sa stringovima kao nizovima karaktera. U nastavku su date neke od najčešće korišćenih funkcija za rad sa nizovima karaktera. Takođe su dati prosti primeri korišćenja standarnih funkcija za rad sa nizovima karaktera:

❑ Funkcija strcpy

U programskom jeziku C nije moguće jednostavno izjednačiti dva stringa (string1 = string2). Kopiranje jednog stringa u drugi se radi na sledeći način:

```
char str_one[] = "abc";
char str_two[] = "def";
strcpy(str_one , str_two); //str_one becomes "def"
```

Napomena: strcpy() ne proverava prostor rezervisan za nizove koji se kopiraju jedan u drugi, tako da može da dođe do memorijskih problema kao u sledećem primeru:

```
char str_one[] = "abc";
char str_two[] = "define";
strcpy(str_one , str_two);
```

Prilikom izvršavanja programa doći će do greške, jer smo kopirali string str_two dužine 7 u string str_one dužine 4.

UPOTREBA FUNKCIJA ZA RAD SA STRINGOVIMA

Kompletna lista funkcija za rad sa stringova može se naći u standardnoj C biblioteci

- Upotreba funkcije strcmp za poređenje dve reči:

```
char name[6];
printf("Enter you name: ");
scanf("%s", name);
if( strcmp( name, "jane" ) == 0 )
           printf("Hello, jane!\n");
```

- Upotreba funkcije strcat za spajanje dva stringa:

```
char age [20];
printf("Enter you age: ");
scanf("%s", age);
strcat( age, " years old." );
printf("You are %s\n", age);
```

- Upotreba funkcije strlen za određivanje dužine stringa:

```
char name[5] = "jane";
int result = strlen(name); //Will return 4.
printf("Lenght is %d\n", result);
```

U kodu koji je prikazan u nastavku je dat još jedan demonstracioni primer korišćenja standardnih funkcija za rad sa nizovima karaktera.

Kompletna lista funkcija za rad sa stringova može se naći u standardnoj C biblioteci.

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

int main ()
{
   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;

   /* copy str1 into str3 */
   strcpy(str3, str1);
   printf("strcpy( str3, str1) :  %s\n", str3 );

   /* concatenates str1 and str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );

   /* total lenghth of str1 after concatenation */
   len = strlen(str1);
   printf("strlen(str1) :  %d\n", len );

   return 0;
}
```

Kao izlaz prethodnog programa se dobija sledeći rezultat:

```
strcpy( str3, str1) :  Hello
strcat( str1, str2):   HelloWorld
strlen(str1) :  10
```

FUNKCIJE ZA PRETVARANJE MALIH SLOVA U VELIKA I OBRATNO

Funkcija toupper() se koristi za pretvaranja malih slova u velika, dok se funkcija tolower() koristi za pretvaranje velikih slova u mala.

❑ Funkcija toupper()

Deklaracija funkcije toupper() ima sledeći oblik:

```
int toupper(int c);
```

gde je c malo slovo koje treba biti konvertovano u veliko. Funkcija kao rezultat vraća ekvivalentno veliko slovo koje odgovara slovu c. Naravno, ukoliko to slovo postoji. Rezultat je celobrojna vrednost koja može biti implicitno konvertovana u karakter (tip char).

U nastavku je dat primer korišćenja funkcije toupper:

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

int main()
{
   int i = 0;
   char c;
   char str[] = "Tutorials Point";

   while(str[i])
   {
      putchar (toupper(str[i]));
      i++;
   }

  return(0);
}
```

❑ Funkcija tolower()

Deklaracija funkcije tolower() ima sledeći oblik:

```
int tolower(int c);
```

gde je c veliko slovo koje treba biti konvertovano u malo. Funkcija kao rezultat vraća ekvivalentno malo slovo koje odgovara slovu c. Naravno, ukoliko to slovo postoji. Rezultat je celobrojna vrednost koja može biti implicitno konvertovana u karakter (tip char).

U nastavku je dat primer korišćenja funkcije tolower:

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

int main()
{
            int counter=0;
            char mychar;
            char str[]="TeSt THis seNTeNce.\n";

            while (str[counter])
            {
                        mychar=str[counter];
                        putchar (tolower(mychar));
                        counter++;
            }
            return 0;
}
```

FUNKCIJE KOJE PROVERAVAJU VREDNOST KARAKTERA

Deklaracija nekoliko korisnih funkcija za rad sa karakterima koje proveravaju vrednost karaktera se nalaze u header fajlu ctype.h standardne C biblioteke

U okviru standardne C biblioteke postoji odgovarajući header fajl ctype.h u kome se nalazi deklaracija nekoliko korisnih funkcija za rad sa karakterima. Sve funkcije kao rezultat vraćaju vrednost različitu od nule ako prosleđeni karakter zadovoljava odgovarajući uslov (TRUE), odnosno vraćaju nulu ako je uslov netačan. U nastavku je data tabela i značenje pojedinih funkcija za rad sa karakterima.

Funkcija	Opis
isalpha	Ispituje da li je argument slovo alfabeta.
isalnum	Ispituje da li je argument slovo alfabeta ili broj.
isdigit	Ispituje da li je argument broj.
islower	Ispituje da li je argument malo slovo
ispunct	Ipistuje da li je argument znak interpunkcije
isupper	Ispituje da li je argument veliko slovo
isspace	Ispituje da li je argument praznina (whitespace)

Slika-3 Funkcije koje proveravaju vrednost karaktera

U nastavku je dat deo koda koji ilustruje korišćenje ovih standardnih funkcija u cilju da se ispita da li se uneta šifra (password) sastoji iz početnog velikog slova, za kojim sledi broj, i na kraju je malo slovo, kao npr. “Z3s.”

```
bool isValidPassWord(char* pw)
{
   if (!isupper(pw[0])    return false;
   if (!isdigit(pw[1])       return false;
   if (!islower(pw[2])     return false;
   return true;
}
```

Formatirani ulaz stringova

Sprintf, sscanf

07

FUNKCIJA SPRINTF()

Funkcija sprintf smešta formatirani izlaz u string. Njena deklaracija se nalazi u okviru stdio.h header fajla

Funkcija standardne C biblioteke int sprintf(char *str, const char *format, ...) smešta formatirani izlaz u string na koji pokazuje pokazivač str. Deklaracija funkcije sprintf() ima sledeći oblik.

```
int sprintf(char *str, const char *format, ...);
```

gde su:

- str – pokazivač na niz znakova u koji se smešta rezultujući niz
- format – Ovo je string odnosno tekst koji će biti upisan u bafer (buffer). Formatiranje se vrši na isti način kao i za funkciju printf.

U nastavku je dat primer korišćenja funkcije sprintf():

```
#include <stdio.h>
#include <math.h>
#define PI 3.141593

int main()
{
   char str[80];

   sprintf(str, "Value of Pi = %f", PI);
   puts(str);

   return(0);
}
```

Nakon kompajliranja dobija se sledeći rezultat:

```
Value of Pi = 3.141593
```

FUNKCIJA SSCANF()

Funkcija sscanf učitava podatke iz stringa umesto sa standardnog ulaza i prema odgovarajućem formatu prebacuje sadržaj stringa u navedene argumente

Standardna funkcija C biblioteke int sscanf(const char *str, const char *format, ...) učitava podatke iz stringa umesto sa standardnog ulaza, pri čemu je str pokazivač na niz karaktera odnosno na string. Deklaracija funkcije sscanf() ima sledeći oblik:

```
int sscanf(char *string, char *format, arg1, arg2, ...);
```

Ova funkcija skenira string na osnovnu formata koji je specificiran u format a zatim smešta rezultujuće vrednosti u promenljive arg1, arg2, itd. Ovi argumenti moraju biti pokazivači. Formatirani string se obično sastoji iz specifikatora konverzije (kao kod obične scanf funkcije), koji se koriste u cilju kontrole konvertovanja ulaznih podataka. Formatirani string može da sadrži:

- Praznine i tabulatore koji neće biti ignorisani.
- Proizvoljne karaktere (sve osim %), za koje se očekuje da će se poklopiti sa sledećim karakterom različitim od praznine koji dolazi sa ulaznog toka.
- Specifikatore konverzije, koji se sastoje od znakova %, opcionog broja koji specificira širinu polja, kao i karaktera konverzije.

U sledećem primeru je opisano korišćenje funkcije sscanf().

```
#include <stdio.h>
#include <stdlib.h>
int main()
{
   int day, year;
   char weekday[20], month[20], dtm[100];
   strcpy( dtm, "Saturday March 25 1989" );
   sscanf( dtm, "%s %s %d  %d", weekday, month,
&day, &year );
   printf("%s %d, %d = %s\n", month, day, year,
weekday );
   return(0);
}
```

Izlaz prethodnog programa je:

```
March 25, 1989 = Saturday
```

U nastavku su dati još neki primeri korišćenja funkcija sscanf i printf:

```
char *msg = “Hello there”;
char *nums = “1 3 5 7 9”;
char s[10], t[10];
int a, b, c, n;
n = sscanf(msg, “%s %s”, s, t);
n = printf(“%10s %-10s”, t, s);
n = sscanf(nums, “%d %d %d”, &a, &b, &c);
printf(“%d flower%s”, n, n > 1 ? “s” : “ “);
```

Konverzija stringova i brojeva

atoi, atof, itoa

07

FUNKCIJA ATOF()

Funkcija atof ili "ASCII to float" konvertuje tekstualnu reprezentaciju realnog broja u odgovarajući numerički tip podatka za realan broj (float)

Standardna funkcija C biblioteke double atof(const char *str) konvertuje string u realan broj. String predstavlja tekstualnu reprezentaciju realnog broja i smešten je u niz karaktera na koji pokazuje argument str, dok se kao rezultat funkcije dobija realan broj tipa double. Deklaracija funkcije atof() ima sledeći oblik:

```
double atof(const char *str);
```

Funkcija kao rezultat vraća odgovarajući realan broj, a u slučaju da se ne može izvršiti validna konverzija funkcija kao rezultat vraća nulu (0.0). U nastavku je dat primer korišćenja funkcije atof():

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main()
{
   float val;
   char str[20];

   strcpy(str, "98993489");
   val = atof(str);
   printf("String value = %s, Float value = %f\n", str, val);

   strcpy(str, "tutorialspoint.com");
   val = atof(str);
   printf("String value = %s, Float value = %f\n", str, val);

   return(0);
}
```

Rezultat programa je:

String value = 98993489, Float value = 98993488.000000
String value = tutorialspoint.com, Float value = 0.000000

FUNKCIJA ATOI()

Funkcija atoi ili "ASCII to int" konvertuje tekstualnu reprezentaciju celog broja u odgovarajući numerički tip podatka za ceo broj (int)

Standardna funkcija C biblioteke int atoi(const char *str) konvertuje string u ceo broj. String predstavlja tekstualnu reprezentaciju celog broja i smešten je u niz karaktera na koji pokazuje argument str, dok se kao rezultat funkcije dobija ceo broj tipa int. Deklaracija funkcije atoi() ima sledeći oblik:

```
int atoi(const char *str);
```

Funkcija kao rezultat vraća odgovarajući ceo broj, a u slučaju da se ne može izvršiti validna konverzija funkcija kao rezultat vraća nulu. U nastavku je dat primer korišćenja funkcije **atoi()**:

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

int main()
{
   int val;
   char str[20];

   strcpy(str, "98993489");
   val = atoi(str);
   printf("String value = %s, Int value = %d\n", str, val);

   strcpy(str, "tutorialspoint.com");
   val = atoi(str);
   printf("String value = %s, Int value = %d\n", str, val);

   return(0);
}
```

FUNKCIJA ITOA()

Funkcija itoa() ili "Integer to ASCII." kao nestandardna funkcija C biblioteke konvertuje celobrojnu vrednost u C-string, tj u tekstualnu reprezentaciju celog broja

Funkcija itoa() je deo stdlib biblioteke. Ovo je nestandardna C funkcija i njen zadataka je da konvertuje celobrojnu vrednost u C-string. Deklaracija funkcije ima sledeći oblik:

```
char *  itoa ( int value, char * str, int base );
```

pri čemu su:

- value – vrednost koju konvertujemo u string
- str – niz u memoriji koji služi za smeštanje C-stringa
- base – numerička osnova koja može biti izmedju 2 i 26, pri čemu 10 znači decimalnu osnovu, 8 oktalnu a 2 binarnu.

Povratna vrednost funkcije je pokazivač na niz karaktera odnosno C string koji je isti kao i parametar str. Ova funkcija nije definisana u ANSI-C niti je deo C++-a ali je podržana od strane nekih kompajlera. Alternativa ovoj funkciji može biti napisana korišćenjem funkcije sprintf na neki od sledećih načina:

```
sprintf(str,"%d",value)  // konvertuje u decimalni
sprintf(str,"%x",value)  // konvertuje u heksadecimalni
sprintf(str,"%o",value)  // konvertuje u oktalnu osnovu
```

U nastavku je dat primer korišćenja itoa() funkcije:

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

int main ()
{
  int i;
  char buffer [33];
  printf ("Enter a number: ");
  scanf ("%d",&i);
  itoa (i,buffer,10);
  printf ("decimal: %s\n",buffer);
  itoa (i,buffer,16);
  printf ("hexadecimal: %s\n",buffer);
  itoa (i,buffer,2);
  printf ("binary: %s\n",buffer);
  return 0;
}
```

Izlaz programa je:

```
Enter a number: 1750
decimal: 1750
hexadecimal: 6d6
binary: 11011010110
```

Argumenti komandne linije

Komandna linija, konzola, argc, argv

08

OSNOVNA RAZMATRANJA

Argumenti komandne linije se kroz argumente funkcije main() prosleđuju glavnom programu

Programski jezik C (i C++) dozvoljava da se neki podaci kao argumenti komande linije proslede programu. Ovi argumenti se nazivaju argumenti komandne linije i mnogo puta su poželjni da postoje u programu kao alternativa ručnom unosu ovih istih promenljivih u kodu (hard coding).

Argumenti komandne linije se hvataju kroz argumente funkcije main() gde se argc odnosi na broj argumenata, dok argv[] predstavlja niz pokazivača, pri čemu svaki od pokazivača u nizu pokazuje na poseban string. Prostije rečeno, char *argv[] će da predstavlja niz stringova, a svaki string se odnosti na zaseban argument komandne linije. U listingu na desnoj strani je dat prost primer koji prvo ispituje da li uopšte postoji argument komandne linije, pa u zavisnosti od toga sprovodi odgovarajuću akciju.

Ovde treba napomenuti da član argv[0] sadrži samo ime programa (/a.out u ovom primeru) dok argv[1] ustvari predstalja pokazivač na prvi konktretan argument, dok *argv[n] predstavlja poslednji argument. Ukoliko nije prosleđen nijedan argument, promenljiva argc će imati vrednost 1, a u slučaju da se unese jedan argument argc će imati vrednost 2. U narednoj lekciji će biti više reči o pokazivačima i stringovima u slučaju da neke stvari nisu dovoljno jasne.

```
#include <stdio.h>
int main( int argc, char *argv[] )
{
   if( argc == 2 )
   {
      printf("The argument supplied is %s\n",
argv[1]);
   }
   else if( argc > 2 )
   {
      printf("Too many arguments supplied.\n");
   }
   else
   {
      printf("One argument expected.\n");
   }
}
```

Nakon izvršavanja koda, dobićemo sledeći rezultat.

```
$./a.out testing
The argument supplied is testing
```

U slučaju da umesto jednog unesemo dva argumenta u komandnoj liniji biće proizveden sledeći rezultat.

```
$./a.out testing1 testing2
Too many arguments supplied.
```

U slučaju da ne prosledimo nijedan argument rezultat je:

```
One argument expected
```

Vežba

Nizovi, funkcije, sortiranje, C-string, funkcije za C stringove

09

OPIS NIZA, INICIJALIZACIJA NIZA

Inicijalizacija je dodeljivanje vrednosti elementima niza u istom iskazu u kome se vrši i deklaracija niza

Primer. Naredni primer ilustruje inicijalizaciju nizova, i korišćenje operatora sizeof() u cilju određivanja broja elemenata niza.

```
#include <stdio.h>
void main()
{
          /* Niz inicijalizujemo tako sto mu navodimo vrednosti u viticasnim zagradama.
          Dimenzija niza se odredjuje na osnovu broja inicijalizatora */
          int a[] = {1, 2, 3, 4, 5, 6};
          /* Isto vazi i za niske karaktera */
          char s[] = {'a', 'b', 'c'};
          /* Ekvivalentno prethodnom bi bilo char s[] = {97, 98, 99}; */
          /* Broj elemenata niza */
          int a_br_elem = sizeof(a)/sizeof(int);
          int s_br_elem = sizeof(s)/sizeof(char);
          /* Ispisujemo nizove */
          int i;
          for (i = 0; i < a_br_elem; i++)
          printf("a[%d]=%d\n",i, a[i]);
          for (i = 0; i < s_br_elem; i++)
          printf("s[%d]=%c\n",i, s[i]);
}
```

FORMIRANJE NIZA SA NENEGATIVNIM ČLANOVIMA

Petlje (ciklusi) su sastavni deo koda kada se jave zadaci i problemi sa nizovima

Zadatak. Napisati program koji prihvata sa standardnog ulaza pozitivan ceo broj n (n <=50), a zatim prihvata po jedan element n-dimenzionalnog niza celih brojeva. Formirati drugi niz koji sadrži samo nenegativne elemente unetog niza, a potom članove drugog niza ispisati na standardni izlaz.

```
#include <stdio.h>
void main()
{
          int n,indeks,a[50],j,rezultat[50];
          do
          {
                    printf("Unesite broj elemenata niza\n");
                    scanf("%d", &n);
          } while (n <1 || n > 50);

          for( indeks=0; indeks<n; indeks++)
                    scanf("%d", &a[indeks]);

          for( indeks=0, j=0; indeks<n; indeks++)
          {
                    if(a[indeks]<0) continue;
                    rezultat[j++]=a[indeks];
          }

          printf("\nNovi niz je: ");
          for( indeks=0; indeks<j; indeks++)
          printf("%d\t", rezultat[indeks]);
          printf("\n");
}
```

BROJANJE POJAVLJIVANA SVIH CIFARA I PRAZNINA

U nastavku je data jedna moguća varijanta programa koji za odgovarajuću ulaznu sekvecnu karaktera broji pojavljivanje svakog od njih

Postoji 12 različitih tipova ulaznih podataka: 10 cifara, praznina i ostali karakteri, pa je stoga pogodno koristiti nizove u cilju čuvanja broja pojavljivanja odgovarajućih cifara, umesto da se koriste individualne promenljive za svaku cifru. U nastavku je data jedna verzija programa za brojanje pojavljivanja cifara:

```
#include <stdio.h>

/* count digits, white space, others */
void main()
{
        int c, i, nwhite, nother;
        int ndigit[10];
        nwhite = nother = 0;

        for (i = 0; i < 10; ++i)
                ndigit[i] = 0;

        while ((c = getchar()) != EOF)
                if (c >= '0' && c <= '9')
                        ++ndigit[c-'0'];
                else if (c == ' ' || c == '\n' || c == '\t')
                        ++nwhite;
                else
                        ++nother;

        printf("digits =");
        for (i = 0; i < 10; ++i)
                printf(" %d", ndigit[i]);

        printf(", white space = %d, other = %d\n", nwhite, nother);
}
```

SORTIRANJE NIZOVA U C-U

Selection sort je najlakša metoda za razumevanje ali je istovremeno i najsporiji način da se izvrši uređenje niza

Zadatak. Program za sortiranje elemenata niza u opadajućem poretku. Napisati glavni program i proceduru za sortiranje:

Funkcija za sortiranje

```
void selection_sort(int n, int a[ ])
{
     int i,j,k;
     for (i=0;i<n-1;i++)
          for (j=i+1;j<n;j++)
               if (a[i]>a[j])
           {
                  k=a[i];
                  a[i]=a[j];
                  a[j]=k;
           }
}
```

Glavni program:

```
void main()
{
            int i, n;
            int a[100];
            void selection_sort(int n, int a[ ]);
            printf ("Unesi broj clanova niza:\n");
            scanf ("%d", &n);
            printf ("Unesi clanove niza: \n");
            for (i=0; i<n; i++)
            {
                        scanf("%d", &a[i]);
            }
            selection_sort(n, a);
            printf ("Sortirani niz je: \n");
            for (i=0; i<n; i++)
            {
                        printf ("%d \n", a[i]);
            }
}
```

NIZOVI I FUNKCIJE

Nizovi se prosleđuju funkciji na tri načina: kao pokazivač, kao niz sa dimenzijom i kao nedimenzionisani niz

Zadatak. Napisati funkciju za ispis niza brojeva - demonstrira prenos niza brojeva u funkciju.

```
#include <stdio.h>
void print_array( int a[], int n)
{
          int i;
          for (i = 0; i < n; i++)
          {
                    a[i]++;
                    printf("%d ",a[i]);
          }
          putchar('\n');
          /* Obratite paznju na ovaj sadrzaj koji se stampa!!! */
          printf("sizeof(a) - u okviru fje : %ld\n", sizeof(a));
}

void main()
{
          int a[] = {1, 2, 3, 4, 5, 6, 7, 8, 9};
          int i;
          printf("sizeof(a) - u okviru main : %ld\n", sizeof(a));
          int n= sizeof(a)/sizeof(int);
          print_array(a, n);
          for (i = 0; i < n; i++)
                    printf("%d ",a[i]);
          putchar('\n');
}
```

MAKSIMUM NIZA

Zadatak . Napisati program koji učitava sa standardnog ulaza broj n (n <=100), a potom n elemenata niza celih brojeva.

Program treba da ispiše maksimum unetog niza na standardni izlaz.

```
#include <stdio.h>
#define DIM 100
int max( int prvi, int drugi );
void main()
{
          int a[DIM], indeks, n, Max;
          printf("\nUnesite dimenziju niza: ");
          scanf("%d", &n);
          printf("\nUnesite clanove niza: ");
          for(indeks=0; indeks<n; indeks++)
                    scanf("%d", &a[indeks]);
          Max=a[0];
          for(indeks=1; indeks<n; indeks++)
                    Max=max(Max, a[indeks]);
          printf("\nMaksimum niza je %d \n", Max);
}

/* max: vraca maksimum dva argumenta */
int max( int prvi, int drugi )
{
          return ( (prvi>=drugi)?prvi:drugi );
}
```

SKALARNO MNOŽENJE NIZOVA

Zadatak . Napisati program koji će ispisati rezultat skalarnog množenja uređene šestorke a i uređene šestorke b.

Uređenu šestorku a čini prvih šest prirodnih brojeva, a uređenu šestorku b čine brojevi 8,7,6,5,4,3.

```
#include <stdio.h>

long mnozi(int x[],int y[],int n);
void main()
{
          int a[]={1,2,3,4,5,6}, b[]={8,7,6,5,4,3};
          printf("Skalarno a*b= %ld\n",mnozi(a,b,6));
}

long mnozi(int x[],int y[],int n)
{
          int br;
          long suma=0;
          for(br=0;br<n;br++)
                              suma=suma+x[br]*y[br];
          return suma;
}
```

PROSLEĐIVANJE NIZA I JEDNOG ELEMENTA NIZA FUNKCIJI

Kada se ime niza prosledi funkciji kroz listu argumenata prosleđivanje se vrši po adresi. Kada se samo jedan član niza prosledi funkcije onda se prosleđivanje vrši po vrednosti

```
// Passing arrays and individual array elements to functions.
#include <stdio.h>

void modifyArray( int [], int ); // appears strange
void modifyElement( int );

int main()
{
   const int arraySize = 5; // size of array a
   int a[ arraySize ] = { 0, 1, 2, 3, 4 }; // initialize array a

   printf("Effects of passing entire array by reference:");
   printf("\n\nThe values of the original array are:\n");

   // output original array elements
   for ( int i = 0; i < arraySize; i++ )
      printf("%3d",a[ i ]);

    printf("\n");

   // pass array a to modifyArray by reference
   modifyArray( a, arraySize );
   printf("The values of the modified array are:\n");

   // output modified array elements
   for ( int j = 0; j < arraySize; j++ )
      printf("%3d",a[ j ]);

   printf("\n\n\nEffects of passing array element by value:");
   printf("\n\na[3] before modifyElement: %3d\n", a[ 3 ]);
```

U prethodnom listingu je dat program kojim opisujemo razliku između prosleđivanja celog niza odnosno samo jednog elementa niza u funkciju. U programu imamo dve funkcije: modifyArray() kojoj prosleđujemo niz čiji sadržaj menjamo u okviru funkcije, i modifyElement() kojoj prosleđujemo jedan element niza i njega modifikujemo unutar funkcije. Kao što se može videti iz listina dobijenog rezulta programa, niz je izmenjen nakon poziva funkcije modifyArray, ali nije izmenjen posle poziva modifyElement.

Rezultat:

```
Effects of passing entire array by
reference:
 The values of the original array are:
   0   1   2   3   4
 The values of the modified array are:
   0   2   4   6   8
 Effects of passing array element by
value:
 a[3] before modifyElement: 6
 Value of element in modifyElement: 12
 a[3] after modifyElement: 6
```

PROSLEĐIVANJE NIZA I JEDNOG ELEMENTA NIZA FUNKCIJI

Kada se ime niza prosledi funkciji kroz listu argumenata prosleđivanje se vrši po adresi. Kada se samo jedan član niza prosledi funkcije onda se prosleđivanje vrši po vrednosti

```
modifyElement( a[ 3 ] ); // pass array element a[ 3 ] by value
   printf("a[3] after modifyElement: %3d\n", a[ 3 ]);

   return 0; // indicates successful termination
} // end main

// in function modifyArray, "b" points to the original array "a" in memory
void modifyArray( int b[], int sizeOfArray )
{
   // multiply each array element by 2
   for ( int k = 0; k < sizeOfArray; k++ )
      b[ k ] *= 2;
} // end function modifyArray

// in function modifyElement, "e" is a local copy of
// array element a[ 3 ] passed from main
void modifyElement( int e )
{
   // multiply parameter by 2
   printf("Value of element in modifyElement: %3d\n", e *= 2 );
} // end function modifyElement
```

VIŠEDIMENZIONALNI NIZOVI – MNOŽENJE MATRICA

Množenje matrica je jedan složeni matematički problem koji se može implementirati i rešiti u nekom programskom jeziku

Zadatak. Napisati program koji vrši mnozenje dve matrice. Uslov da se dve matrice množe je taj da broj kolona prve matrice bude jednak broju vrsta druge matrice.

```
#include <stdio.h>

int main()
{
  int m, n, p, q, c, d, k, sum = 0;
  int first[10][10], second[10][10], multiply[10][10];

  printf("Enter the number of rows and columns of first matrix\n");
  scanf("%d%d", &m, &n);
  printf("Enter the elements of first matrix\n");

  for (  c = 0 ; c < m ; c++ )
    for ( d = 0 ; d < n ; d++ )
      scanf("%d", &first[c][d]);

  printf("Enter the number of rows and columns of second matrix\n");
  scanf("%d%d", &p, &q);

  if ( n != p )
    printf("Matrices with entered orders can't be multiplied with each other.\n");
  else
  {
    printf("Enter the elements of second matrix\n");

    for ( c = 0 ; c < p ; c++ )
      for ( d = 0 ; d < q ; d++ )
        scanf("%d", &second[c][d]);
```

VIŠEDIMENZIONALNI NIZOVI – MNOŽENJE MATRICA

Množenje matrica je jedan složeni matematički problem koji se može implementirati i rešiti u nekom programskom jeziku

```
for ( c = 0 ; c < m ; c++ )
    {
      for ( d = 0 ; d < q ; d++ )
      {
        for ( k = 0 ; k < p ; k++ )
        {
          sum = sum + first[c][k]*second[k][d];
        }

        multiply[c][d] = sum;
        sum = 0;
      }
    }

    printf("Product of entered matrices:-\n");

    for ( c = 0 ; c < m ; c++ )
    {
      for ( d = 0 ; d < q ; d++ )
        printf("%d\t", multiply[c][d]);

      printf("\n");
    }
  }
```

PRIMER UPOTREBE C-STRINGOVA

Stringovi u C-u ustvari prestavljaju nizove karaktera gde je poslednji član niza null karakter

Zadatak. Napisati program koji učitava reč sa standardnog ulaza koja ima ne više od 100 karaktera i ispisuje je na standardni izlaz. Reč je bilo koji niz karaktera koja ne sadrži beline (blanko, tab, prelaz u novi red,...). Učitanu reč čuvati u stringu (nizu karaktera).

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

void get_word(char s[])
{
          int c, i = 0;
          while (!isspace(c=getchar())) s[i++] = c;
          s[i] = '\0'; /*OBAVEZAN kraj stringa u C-u */
}

void main()
{
          char s[100];
          get_word(s);
          printf("%s\n", s);
}
```

UPOTREBA FUNKCIJA ZA PRETVARANJA MALIH SLOVA U VELIKA I OBRATNO

Za ispitivanje da li je slovo malo ili veliko koristimo funkcije isupper() i islower(), respektivno, dok za pretvaranje malih slova u velika i obratno koristimo funkcije toupper() odnosno tolower()

Zadatak. Napisati program koji tekst sa standardnog ulaza prepisuje na standardni izlaz pretvarajući početna slova rečenice u velika. Pretpostaviti da se u tekstu znaci .?! pojavljuju samo kao znaci završetka rečenica. Dakle, u tekstu nema rednih brojeva koji bi se završavali tačkom. Tekst se završava markerom kraja datoteke(Ctrl+Z).

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

void main()
{
            enum {F,T};
            int znak,prvi=T;
            while ( (znak=getchar() ) != EOF)
            {
                        if (isupper(znak) )
                        {
                                    if (!prvi) znak=tolower(znak);
                                    else prvi=F;
                        }
                        else if (islower(znak) )
                        {
                                    if(prvi)
                                    {
                                                znak=toupper(znak);
                                                prvi=F;
                                    }
                        }
                        else
                                    if (znak =='.' || znak=='!' || znak =='?') prvi=T;
                        putchar(znak);
            }
}
```

FUNKCIJE I C-STRINGOVI

Zadatak. Napisati funkcije za rad sa nizovima karaktera

Sadrzaj datoteke mstring.h:

```
int string_length(char s[]) ;
void string_copy(char dest[], char src[]) ;
void string_concatenate(char s[], char t[])
int string_compare(char s[], char t[])
int string_char(char s[], char c)
int string_last_char(char s[], char c)
int string_string(char str[], char sub[])
```

Sadrzaj datoteke stringovi.c

```
#include <stdio.h>
void main()
{
            char s[100];
            char t[] = "Zdravo";
            char u[] = " svima";
            string_copy(s, t);          //strcpy(s,t);
            printf("%s\n", s);
            string_concatenate(s, u);  //strcat(s,u);
            printf("%s\n", s);
            printf("%d\n",string_char("racunari", 'n'));
            printf("%d\n",string_last_char("racunari",
'a'));
            printf("%d\n",string_string("racunari",
"rac"));
            printf("%d\n",string_string("racunari",
"ari"));
            printf("%d\n",string_string("racunari",
"cun"));
            printf("%d\n",string_string("racunari",
"cna"));
}
```

Sadrzaj datoteke mstring.c:

```
#include "mstring.h"
int string_length(char s[])
{
            int i;
            for (i = 0; s[i]; i++) ;
            return i;
}

void string_copy(char dest[], char src[])
{
            int i;
            for (i = 0; (dest[i]=src[i]) != '\0';
i++) ;
}

void string_concatenate(char s[], char t[])
{
            int i, j;
            for (i = 0; s[i]; i++);
            for (j = 0; s[i] = t[j]; j++, i++) ;
}

int string_compare(char s[], char t[])
{
            int i;
            for (i = 0; s[i]==t[i]; i++)
            if (s[i] == '\0') return 0;
            return s[i] - t[i];
}
```

FUNKCIJE ZA RAD SA C-STRINGOVIMA

Funkcije za rad sa stringovima u C-u se nalaze u okviru heder fajla string.h standardne C biblioteke

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

const int MAXIMUM_LENGTH = 80;

int main()
{
 char first_string[MAXIMUM_LENGTH];
 char second_string[MAXIMUM_LENGTH];

 printf("Enter first string: ");
 gets(first_string);
 printf("Enter second string: ");
 gets(second_string);

 printf("Before copying the strings were ");
 if (strcmp(first_string,second_string))
  printf("not ");
 printf("the same.\n");

 strcpy(first_string,second_string);

 printf("After copying the strings were ");
 if (strcmp(first_string,second_string))
  printf("not ");
 printf("the same.\n");

 strcat(first_string,second_string);

 printf("After concatenating, the first string is: ");
 printf("%s\n",first_string);

 return 0;
}
```

U prethodnom listingu je dat primer korišćenja funkcija za rad sa stringovima: "gets(...)", "strcmp(...)", "strcpy(...)" i "strcat(...)":

Mogući izgled konzole nakon izvršavanja prethodnog koda (ukoliko je korisnik uneo stringove: Hello class i Hello Rob):

```
Enter first string: Hello
class.
Enter second string: Hello Rob.
Before copying the strings were
not the same.
After copying the strings were
the same.
After concatenating, the first
string is: Hello Rob.Hello Rob.
```

KONVERZIJA IZMEĐU C-STRINGOVA I BROJEVA

Za pretvaranje stringova u brojeve najčešće koristimo funkcije atoi() i atof() čije se deklaracije nalaze u okviru header fajla stdlib.h standardne C biblioteke

Funkcija atoi() ima jedan argument, C-string, i kao rezultat vraća ceo broj koji reprezentuje string. Primer upotrebe je sledeći:

```
int num = atoi("7654");
```

U nastavku je dat program u kome se upotrebljava funkcija atoi:

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

int main(void)
{
   char input[80];
   int num;
   printf("Enter an integer: ");
   scanf("%s",&input);
   for (int x = 0; x < strlen(input); x++)
   {
      if (x == 0)
      {
         if (!isdigit(input[x]) && input[x] != '-')
            return false;
      }
      else
      {
         if (!isdigit(input[x]))
            return false;
      }
   }
   num = atoi(input);
   printf("%3d\n", num);
   return 0;
}
```

PRONALAŽENJE NAJDUŽEG ULAZNOG TEKSTA

Cilj zadatka je da se napiše program koji će da čita liniju po liniju sa standardnog ulaza i da pronađe i oštampa najveći tekst među njima

Da bi smo ilustrovali korišćenje niza karaktera i funkcija koje manipulišu sa njima, napisaćemo program koji čita tekst liniju po liniju i štampa najveći među njima.

Kada pronađemo liniju koja je veća od prethodne onda je neophodno tu liniju sačuvati negde, i zato koristimo funkciju copy koja kopira sadržaj na neko bezbedno mesto. Funkcija getline() će da služi za čitanje teksta i smeštanje u neki niz karaktera.

Funkcije getline i copy su deklarisane na početku programa dok njihove definicije pišemo ispod funkcije main. Funkcije main i getline komuniciraju preko para argumenata i povratne vrednosti. Funkcija getline je deklarisana na sledeći način

```
int getline(char s[], int lim);
```

pri čemu je prvi argument niz karaktera a drugi argument lim je ceo broj koji predstavlja dužinu niza. U nastakvu je dat program koji pronalazi najveći uneti tekst:

```
#include <stdio.h>
#define MAXLINE 1000 /* maximum input line length */

int getline(char line[], int maxline);
void copy(char to[], char from[]);

/* print the longest input line */
void main()
{
            int len; /* current line length */
            int max; /* maximum length seen so far
*/
            char line[MAXLINE]; /* current input
line */
            char longest[MAXLINE]; /* longest line
saved here */
            max = 0;
            while ((len = getline(line, MAXLINE)) >
0)
                        if (len > max)
                        {
                                    max = len;
                                    copy(longest,
line);
                        }
            if (max > 0) /* there was a line */
            printf("%s", longest);
            return 0;
}

/* getline: read a line into s, return length */
int getline(char s[],int lim)
{
            int c, i;
            for (i=0; i < lim-1 &&
(c=getchar())!=EOF && c!='\n'; ++i)
```

PRONALAŽENJE NAJDUŽEG ULAZNOG TEKSTA

Cilj zadatka je da se napiše program koji će da čita liniju po liniju sa standardnog ulaza i da pronađe i oštampa najveći tekst među njima

Da bi smo ilustrovali korišćenje niza karaktera i funkcija koje manipulišu sa njima, napisaćemo program koji čita tekst liniju po liniju i štampa najveći među njima.

Kada pronađemo liniju koja je veća od prethodne onda je neophodno tu liniju sačuvati negde, i zato koristimo funkciju copy koja kopira sadržaj na neko bezbedno mesto. Funkcija getline() će da služi za čitanje teksta i smeštanje u neki niz karaktera.

Funkcije getline i copy su deklarisane na početku programa dok njihove definicije pišemo ispod funkcije main. Funkcije main i getline komuniciraju preko para argumenata i povratne vrednosti. Funkcija getline je deklarisana na sledeći način

```
int getline(char s[], int lim);
```

pri čemu je prvi argument niz karaktera a drugi argument lim je ceo broj koji predstavlja dužinu niza. U nastakvu je dat program koji pronalazi najveći uneti tekst:

```
 s[i] = c;
 if (c == '\n')
 {
  s[i] = c;
  ++i;
 }
 s[i] = '\0';
 return i;
}

/* copy: copy 'from' into 'to'; assume to is
big enough */
void copy(char to[], char from[])
{
 int i;
 i = 0;
 while ((to[i] = from[i]) != '\0')
  ++i;
}
```

PRONALAŽENJE NAJDUŽEG TEKSTA – KORIŠĆENJE GLOBALNIH PROMENLJIVIH

Cilj je da se na osnovu prethodnog primera napiše novi kod gde će odgovarajuće promenljive biti izmeštene i tretirane kao globalne u cilju smanjenja prostora u stek memoriji

U nastakvu je dat primer za identičan problem kao u prethodnom primeru s tim što su sledeće promenljive izmeštene izvan main funkcije i definisane kao globalne:

```
int  max; /* maximum length seen so far */
char line[MAXLINE]; /* current input line */
char longest[MAXLINE]; /* longest line saved here */
```

U nastavku je dat ceo kod:

```
#include <stdio.h>
#define MAXLINE 1000 /* maximum input line size */

int  max; /* maximum length seen so far */
char line[MAXLINE]; /* current input line */
char longest[MAXLINE]; /* longest line saved here */

int getline(void);
void copy(void);

/* print longest input line; specialized version */
void main()
{
            int len;
            max = 0;
            while ((len = getline()) > 0)
            if (len > max) {
            max = len;
            copy();
            }
            if (max > 0) /* there was a line */
            printf("%s", longest);
            return 0;
}
```

PRONALAŽENJE NAJDUŽEG TEKSTA – KORIŠĆENJE GLOBALNIH PROMENLJIVIH

Cilj je da se na osnovu prethodnog primera napiše novi kod gde će odgovarajuće promenljive biti izmeštene i tretirane kao globalne u cilju smanjenja prostora u stek memoriji

U nastakvu je dat primer za identičan problem kao u prethodnom primeru s tim što su sledeće promenljive izmeštene izvan main funkcije i definisane kao globalne:

```
int  max; /* maximum length seen so far */
char line[MAXLINE]; /* current input line */
char longest[MAXLINE]; /* longest line saved here */
```

U nastavku je dat ceo kod:

```
/* getline: specialized version */
int getline(void)
{
 int c, i;
 for (i = 0; i < MAXLINE - 1 && (c=getchar()) != EOF && c != '\n'; ++i)
 line[i] = c;
 if (c == '\n') {
 line[i] = c;
 ++i;
 }
 line[i] = '\0';
 return i;
}
/* copy: specialized version */
void copy(void)
{
 int i;
 i = 0;
 while ((longest[i] = line[i]) != '\0')
 ++i;
}
```

ZADACI ZA SAMOSTALAN RAD

U nastavku su dati dodatni zadaci za vežbu

1. Napraviti program koji generiše niz neparnih brojeva a potom taj niz množi sa 2 i prebacuje u novi niz.
2. Napravi niz brojeva od 100 elemenata. Svaki član se računa kao (2*i)* (2*i). Ispisati sve elemente ovog niza
3. Napraviti program koji generiše niz neparnih brojeva a potom taj niz množi sa 2 i prebacuje u novi niz. Pronaći najmanji element niza.
4. Napraviti program koji menja svaku reč Jabuka u tekstu u Kruška i svaku reč Put u tekstu u Autoput. Ispisati prva 3 i zadnja 3 karaktera izmenjenog teksta.
5. Napraviti program koji ispisuje 10 random karaktera na ekranu sa prvim velikim slovom. Napomena: Koristiti funkciju rand() biblioteke stdlib.h i pretvaranje brojeva u karaktere.
6. NCP koji čita tekst sa standardnog ulaza i svaku liniju teksta ispisuje na standardni izlaz šifrirane po shemi koja utiče samo na slova:

SHEMA
A B ... Y Z a b ... y z
c d ... a b D E ... B C
Broj linija teksta nije unapred poznat, linije su limitirane dužine. Može se pretpostaviti da mašinski set znakova odgovara ASCII kodu.
PRIMER

ULAZ	IZLAZ
1. Baba 23/05	1. dDED 23/05
32. Zaza 34/04	32. bDCD 34/04

7. Napisati program koji traži od korisnika da preko tastature popuni niz od 10 brojeva brojevima od 99 do 999 (program trazi unos prvog broja koji ce biti unet u niz, nakon uspesnog unosa trazi se unos drugog broja itd. Uneti brojevi moraju biti u intervalu od 99 do 999, a ukoliko je broj izvan tog intervala traži se od korisnika da ponovi unos). Nakon unosa poslednjeg (desetog) elementa od korisnika se trazi da unese jos jedan broj u istom intervalu. Program koristi taj broj za pretragu, tj. treba da pretraži prethodno uneti niz i utvrdi da li se i koliko puta uneti kontrolni broj nalazi u nizu, i o tome obavesti korisnika.

Zaključak

11

REZIME

Na osnovu svega obrađenog možemo da izvedemo sledeći zaključak:

Za razliku od tipa podatka niza koji može biti int, float ili char, tačno određeni niz ne može istovremeno sadržati cele brojeve, realne brojeve i karaktere. Svi članovi niza moraju biti istog tipa.

Niz mora biti deklarisan pre korišćenja. Sintaksa za deklarisanje niza je gotovo identična sintaksi za deklarisanje ostalih primitivnih promenljivih kao što su int, char, double, itd. Jedina razlika izmedju deklarisanja obične primitivne promenljive i niza je ta što se prilikom deklarisanja niza nakon imena niza navodi broj uokviren uglastim zagradama ([ ]). Taj broj predstavlja deklarator veličine niza.

Niz se može kreirati i prilikom inicijalizacije. Inicijalizacija je postupak gde se u istom iskazu vrši deklaracija niza i dodeljivanje početne vrednosti njegovim elementima, za razliku od običnog dodeljivanja gde se promena vrednosti elemenata niza vrši u linijama koje slede nakon linije deklaracije.

Nizovi se mogu proslediti funkciji, i u tom slučaju se niz prosleđuje po adresi.

Standardna C biblioketa ctype.h sadrži veliki broj funkcija koje su korisne za rad sa karakterima. Funkcije *toupper* i *tolower* imaju samo jedan parametar, karakter, a u slučaju da karakter predstavlja slovo abecede (od A do Z ili od a do z) onda funkcija *toupper* konvertuje mala slova u velika, dok *toupper* radi obratno.

Moguće je koristiti funkciju *strlen* da bi ste odredili dužinu C-string, funkciju *strcpy* da se C-stringu dodeli neka vrednost, strcat da se jedan string doda na kraj drugog. Takođe je moguće koristiti funkiju *strcmp* u cilju poređenja dva stringa.

Standardna biblioteka *stdlib.h* sadrži nekoliko koristnih funkcija za konvertovanje C-string reprezentacije brojeva u numeričke tipove podataka i obratno. Ove funkcije su: *atoi,* ili “ASCII to integer”, *atol,* ili “ASCII to Long”, *atof,* ili “ASCII to float”, i funkcija *itoa,* ili “Integer to ASCII”.