UNIVERZITET METROPOLITAN BEOGRAD

# CS130 - C/C++ PROGRAMSKI JEZIK

## Nizovi i stringovi

Lekcija 03

PRIRUČNIK ZA STUDENTE

# CS130 - C/C++ PROGRAMSKI JEZIK

## Lekcija 03

## *NIZOVI I STRINGOVI*


- Nizovi i stringovi
- Poglavlje 1: Osnovi o nizovima
- Poglavlje 2: Nizovi i funkcije
- Poglavlje 3: Nizovi i pokazivači
- Poglavlje 4: Višedimenzionalni nizovi
- Poglavlje 5: C-stringovi
- Poglavlje 6: Funkcije za rad sa C-stringovima
- Poglavlje 7: Argumenti komandne linije
- Poglavlje 8: Vežbe
- Poglavlje 9: Zadaci za samostalan rad
- Poglavlje 10: Domaći zadatak
- Zaključak

# UVOD

## *Ova lekcija treba da ostvari sledeće ciljeve:*

U okviru ove lekcije ćemo opisati teme koje su karakteristične za korišćenje nizova i takozvanih C-stringova. Nizovi nam omogućavaju da koristimo jednu promenljivu koja će čuvati ogroman broj podataka. Po konceptu su slični listama u Python-u i drugim programskim jezicima.

Kada je u pitanju rad sa karakterima i rečima, u C jeziku se koristi specifični niz karaktera koji na kraju sadrže null karakter i naziva se ***C-string*** . Opisaćemo način rada sa C-stringovima, a na kraju lekcije će biti opisane funkcije koje se koriste za C-stringove.

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

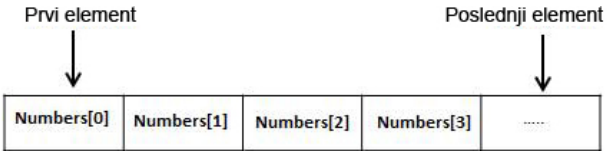
# Poglavlje 1

# Osnovi o nizovima

## OPIS NIZA

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

Niz služi za smeštanje kolekcije promenljivih istog tipa. Članovi niza zauzimaju susedne memorijske lokacije (Slika 1.1).




Slika 1.1 Predstavljanje niza u memoriji računara [5]

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*) je pozitivna celobrojna vrednost (veća od nule) dok tip (type) može biti bilo koji C/C++ tip podatka. 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:

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

i kreiraće se potpuno identičan niz. U nastavku je dat primer dodele vrednosti odgovarajućem članu niza:

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

Prethodni izraz dodeljuje petom članu niza vrednost *50.0*. Kao i u Python-u, indeks niza počinje sa nulom (*a[0]*) dok poslednji član niza ima indeks *N-1*, gde je sa *N* označena dimenzija niza.

# 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 };
printf("%d",sizeof(anArray));
```

Prethodni kod će dati rezultat 20 (5 * 4 bajta).

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 nElems = sizeof(anArray) / sizeof(anArray[0]);
```

S obzirom 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 u stvari 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*

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
};

// alociraj niz 5 celobrojnih elemenata
int anTestScores[MAX_STUDENTS];
anTestScores[STAN] = 76;
```

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.

# PRISTUP ELEMENTU NIZA

## *Elementi niza mogu biti indeksirani konstantnim i promenljivim vrednostima, ali deklaracija niza mora biti izvršena konstantom*

Kao i u Javi, 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*.

Članovima niza se može pristupiti preko indeksa koji može biti neka celobrojna promenljiva vrednost:

```
int anArray[5];
int nIndex = 3;
anArray[nIndex] = 7;
```

Međutim, kada se vrši deklaracija niza, veličina niza mora biti konstanta. U sledećem primeru su pokazani pravilni i nepravilni načini deklaracije niza.

```
int anArray[5];
#define ARRAY_SIZE 5
int anArray[ARRAY_SIZE];

const int nArraySize = 5;
int anArray[nArraySize];

enum ArrayElements
{
    MAX_ARRAY_SIZE = 5;
};
int anArray[MAX_ARRAY_SIZE];
```

```
int nSize = 5;
int anArray[nSize];   // Nije ok!
```

Da zaključimo, elementi niza mogu biti indeksirani konstantnim i promenljivim vrednostima, ali deklaracija niza mora biti izvršena konstantom. Ovo znači da dužina niza mora biti unapred poznata, pre kompajliranja.

U nizovima se mogu smestiti proizvoljni tipovi podataka, uključujući sve primitivne tipove ali i strukture (***struct***) o kojima će biti više reči u nekoj od narednih lekcija.

# Poglavlje 2

# Nizovi i funkcije

## PROSLEĐIVANJE NIZA FUNKCIJI

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

Niz se može proslediti funkciji kroz listu argumenata na dva uobičajena načina.

- Formalni parametar kao dimenzionisani niz:

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

- Formalni parametar kao nedimenzionisani niz:

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

Bilo da koristimo jedan ili drugi način rezultat će biti isti. Obe funkcije se iz nekog drugog bloka pozivaju na identičan način.

```
int balance[5] = {1000, 2, 3, 17, 50};
myFunction(balance);
```

Takođe treba znati da se izmene koje se u funkciji izvrše nad nizom pamte i kada se završi funkcija. Zašto je to tako videćemo u narednoj sekciji kada budemo opisali veoma blisku vezu između nizova i pokazivača.

S obzirom na to da, što ćemo videti u sledećoj sekciji, ime niza u stvari predstavlja pokazivač na prvi njegov element, to znači da funkciji možemo niz proslediti i preko pokazivača. To je u stvari treći način da se niz prosledi funkciji:

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

Sva tri načina su veoma slična međusobno, i proizvode isti rezultat, zato što svi slučajevi šalju istu 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.

# PRIMER KORIŠĆENJA NIZOVA U FUNKCIJI

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

Sada pogledajmo 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 bismo pozvali funkciju iz glavnog programa koristimo sledeći kod:

```
#include <stdio.h>

/* deklaracija funkcije */
double getAverage(int arr[], int size);

int main ()
{
   /* niz od 5 elemenata tipa int */
   int balance[5] = {1000, 2, 3, 17, 50};
   double avg;

   /* prosledi pokazivač na niz kao argument funkcije */
   avg = getAverage( balance, 5 ) ;

   /* odštampaj vrednost vraćenu iz funkcije */
   printf( "Srednja vrednost je : %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 Javi).

Nakon izvršavanja prethodnih linija koda, dobija se sledeći rezultat:

```
Srednja vrednost je: 214.400000
```

# Poglavlje 3

# Nizovi i pokazivači

## POKAZIVAČI NA NIZOVE

### *Ime niza ustvari predstavlja pokazivač na prvi element niza*

Pokazivače smo već uveli u ***Lekciji L01*** . Sada ćemo otići korak dalje i malo opširnije opisati njihovu vezu sa nizovima. Naime, ime niza u stvari 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 izraza **p*, **(p+1)*, **(p+2)* i tako dalje.

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

## 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.

Pretpostavimo da imamo pokazivač na realnu promenljivu (tipa ***double***) i niz realnih brojeva (tipa ***double***):

```
double *dPtr;
double dArr[5] = { 0.0, 1.1, 2.2, 3.3, 4.4 },
double *dPtr = dArr;
```

U prethodnim linijama 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 odnosi, kao što pokazuje sledeći primer:

```
int i = 0;
dPtr = dPtr + 1;
dPtr = 2 + dPtr;
printf( "%.1f\n", *dPtr );
printf( "%.1f\n", *(dPtr -1) );
```

Izraz:

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

dodaje veličinu memorije jednog elementa 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++;
```

O vezi nizova i pokazivača, kao i o pokazivačkoj aritmetici će biti više reči u narednoj lekciji.

# Poglavlje 4

# Višedimenzionalni nizovi

## 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 u stvari 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 (Slika 4.1):

| | 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 4.1 Grafički prikaz 2D niza [5]

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 *i* 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} ,   /*  inicijalizacija elemenata za vrstu 0 */
 {4, 5, 6, 7} ,   /*  inicijalizacija elemenata za vrstu 1 */
 {8, 9, 10, 11}   /*  inicijalizacija elemenata za vrstu 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 štampanje članova niza:

```
#include <stdio.h>
int main ()
{
   int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};
   int i, j;

   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
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

# VIŠEDIMENZIONALNI NIZOVI I FUNKCIJE

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

Višedimenzionalni nizovi se mogu kao formalni parametri proslediti funkciji na sličan način kao kod 1D nizova. U nastavku je primer korišćenja višedimenzionalnih nizova kao argumenata funkcije.

```
#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("Prikazivanje Vrednosti: \n");
    for(int i = 0;  i < 3; ++ i) {
        for(int j = 0; j < 2; ++j) {
            printf("%5d ",n[i][j]);
        }
        printf("\n");
    }
}
```

Funkcija *display* služi za štampanje elemenata niza.

Kao rezultat, dobija se sledeći izlaz:

```
Prikazivanje Vrednosti:
3 4 9 5 7 1
```

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

# Poglavlje 5

# C-stringovi

## 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++ se pišu između navodnika *""*. 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:

```
"zdravo, dragi"
"zdravo, \
dragi"
"zdravo, " "d" "ragi"
```

### 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* jednaka 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";
```

Prilikom definisanja string konstanti i literala nije neophodno staviti null karakter na kraj stringa pošto C kompajler to radi umesto vas prilikom inicijalizacije stringa.

# 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[] = "Programiranje je zabavno";
    display(str);
    return 0;
}

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

Rezultat prethodnog koda je:

```
string: Programiranje je zabavno
```

# 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  11
void main()
{
   int j;
   char week[DAYS] [MAX] = {"Nedelja", "Ponedeljak", "Utorak",
 "Sreda", "Cetvrtak", "Petak", "Subota" };
   for(j=0;j<DAYS;j++)
   {
```

```
        printf("%s\n",week[j]);
    }
}
```

Rezultat prethodnog programa:

```
Nedelja
Ponedeljak
Utorak
Sreda
Cetvrtak
Petak
Subota
```

U prethodnom primeru, kao što smo već spomenuli, koristimo dvodimenzionalni niz karaktera koji u stvari predstavlja niz stringova.

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

## *Pokazivači na stringove u C-u su deklarisani kao pokazivači na nizove karaktera. Pomoću funkcije scanf nije moguće čitati string korišćenjem pokazivačke promenljive*

Pokazivači 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;
}
```

Pokazivačka promenljiva ne može biti korišćena kao argument funkcije *scanf* u cilju učitavanja stringa. Pogledajmo primer:

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

```
    char my_array[10];
    char *ptr_section2;

    printf("Ukucajte hello pa pritisnite enter\n");
    scanf("%s", my_array);
    ptr_section2 = my_array;
    printf("%s\n", ptr_section2);

    return 0;
}
```

Iz prethodnog primera vidimo da u cilju učitavanja nekog teksta koristimo definisani niz *my_array*, a zatim pokazivaču *ptr_section2* dodeljujemo *my_array*. Tek nakon toga pomoću funkcije *printf* i uz primenu pokazivača *ptr_section2* štampamo vrednost teksta.

# Poglavlje 6

# Funkcije za rad sa C-stringovima

## 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( "Unesite vrednost :");
   gets( str );

   printf( "\nUneli ste: ");
   puts( str );

   return 0;
}
```

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

```
Unesite vrednost : ovo je test
Uneli ste: ovo je test
```

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

# FUNKCIJE 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*

U okviru programskog jezika C postoji veliki broj standardnih funkcija za rad sa stringovima kao nizovima karaktera (Slike 6.1 i 6.2).

| 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 6.1.1 Osnovne funkcije za rad sa tekstom kao nizom karaktera [5]

| 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 6.1.2 Osnovne funkcije za konverziju stringa u brojeve [5]

U nastavku su date neke od najčešće korišćenih funkcija za rad sa nizovima karaktera. Takođe su dati prosti primeri korišćenja standardnih 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 postaje "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("Unesite vaše ime: ");
scanf("%s", name);
if( strcmp( name, "Jana" ) == 0 )
 printf("Cao, Jana!\n");
```

Upotreba funkcije *strcat* za spajanje dva stringa:

```
char age [20];
printf("Unesite vaše godine: ");
scanf("%s", age);
strcat( age, " godina." );
printf("Vi imate %s\n", age);
```

Upotreba funkcije *strlen* za određivanje dužine stringa:

```
char name[5] = "Jana";
int result = strlen(name); //Vraća vrednost 4.
printf("Dužins stringa je %d\n", result);
```

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

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

```
#include <stdio.h>
#include <string.h>
int main ()
{
   char str1[12] = "Hello";
   char str2[12] = "World";
   char str3[12];
   int  len ;

   /* kopira str1 u str3 */
   strcpy(str3, str1);
```

```
   printf("strcpy( str3, str1) :  %s\n", str3 );
   /* spaja str1 i str2 */
   strcat( str1, str2);
   printf("strcat( str1, str2):   %s\n", str1 );
   /* ukupna duzina str1 nakon spajanja */
   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.*

Deklaracija funkcije *toupper* ima sledeći oblik:

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

gde je *c* malo slovo koje treba da bude 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***).

Deklaracija funkcije *tolower* ima sledeći oblik:

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

gde je *c* veliko slovo koje treba da bude 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***).

Primer korišćenja funkcije *toupper*:

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

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

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

# 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 fajl zaglavlja (header fajl) pod nazivom ***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 6.1.3 Funkcije koje proveravaju vrednost karaktera [5]

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;
}
```

# 6.1 Formatirani ulaz stringova

## FUNKCIJA SPRINTF()

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

Funkcija standardne C biblioteke sprintf 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, "Vrednost konstante Pi = %f", PI);
   puts(str);

   return(0);
}
```

Nakon kompajliranja dobija se sledeći rezultat:

```
Vrednost konstante 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 sscanf učitava podatke iz stringa umesto sa standardnog ulaza.

Deklaracija funkcije *sscanf* ima oblik:

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

pri čemu je *string* pokazivač na niz karaktera odnosno na string.

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 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, "Subota Mart 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 = "Zdravo svima";
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 cvet%s", n, n > 1 ? "s" : " ");
printf("a = %d, odgovor = %d\n", a, b+c);
```

# Poglavlje 7

# Argumenti komandne linije

## 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.

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 odnosi na zaseban argument komandne linije. U listingu koji sledi 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]* u stvari predstavlja pokazivač na prvi konkretan 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("Obezbedjeni argument je %s\n", argv[1]);
   else if( argc > 2 )
    printf("Previse argumenata je uneto.\n");
   else
    printf("Ocekuje se bar jedan argument.\n");
}
```

Nakon izvršavanja koda, dobićemo sledeći rezultat.

```
$./a.out testing
Obezbedjeni argument je testing
```

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

```
$./a.out testing1 testing2
Previse argumenata je uneto.
```

U slučaju da ne prosledimo nijedan argument rezultat je:

```
Ocekuje se bar jedan argument.
```

# Poglavlje 8

# Vežbe

## FORMIRANJE NIZA SA NENEGATIVNIM ČLANOVIMA (9 MIN)

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

Napisati program koji prihvata sa standardnog ulaza pozitivan ceo broj *n* (*n <=50*), a zatim prihvata po jedan element *n*-dimenzionog niza celih brojeva. Formirati drugi niz koji sadrži samo ne-negativne 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 (9 MIN)

*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>

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

    printf(", praznine = %d, ostalo = %d\n", nwhite, nother);
}
```

# PROSLEĐIVANJE NIZA I JEDNOG ELEMENTA NIZA FUNKCIJI (9 MIN)

*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*

```
#include <stdio.h>

void modifyArray( int [], int ); // deklaracija deluje cudno?
void modifyElement( int );

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

   printf("Efekat prosledjivanja celog niza po referenci:");
   printf("\n\nVrednosti originalnog niza su:\n");
   for ( int i = 0; i < arraySize; i++ )
    printf("%3d",a[ i ]);
    printf("\n");

   // prosledi niz a funkciji modifyArray po referenci
   modifyArray( a, arraySize );
   printf("vrednosti modifikovanog niza su:\n");
   for ( int j = 0; j < arraySize; j++ )
    printf("%3d",a[ j ]);

   printf("\n\n\nEfekat prosledjivanja jednog elementa niza po vrednosti:");
   printf("\n\na[3] pre poziva modifyElement: %3d\n", a[ 3 ]);
   modifyElement( a[ 3 ] ); // prosledi element niza a[ 3 ] po vrednosti
   printf("a[3] nakon poziva modifyElement: %3d\n", a[ 3 ]);

   return 0;
}

// u funkciji modifyArray, "b" pokazuje na originalni niz "a" iz memorije
void modifyArray( int b[], int sizeOfArray )
{
   // pomnozi svaki element niza sa 2
   for ( int k = 0; k < sizeOfArray; k++ )
    b[ k ] *= 2;
}

// u funkciji modifyElement, "e" je lokalna kopija
// elementa niza a[ 3 ] koji je prosledjen iz main-a
void modifyElement( int e )
{
   // pomnozi parametar sa 2
   printf("Vrednost elementa u modifyElement je: %3d\n", e *= 2 );
}
```

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 rezultata programa, niz je izmenjen nakon poziva funkcije *modifyArray*, ali nije izmenjen posle poziva *modifyElement*. Rezultat je:

```
Efekat posledjivanja celog niza po referenci:
Vrednosti originalnog niza su:
  0  1  2  3  4
Vrednosti modifikovanog niza su:
  0  2  4  6  8
Efekat prosledjivanja jednog elementa niza po vrednosti:
a[3] pre poziva modifyElement: 6
Vrednost elementa u modifyElement: 12
a[3] nakon poziva modifyElement: 6
```

# FUNKCIJE ZA RAD SA C-STRINGOVIMA (9 MIN)

*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("Unesi prvi string: ");
    gets(first_string);
    printf("Unesi drugi string: ");
    gets(second_string);

    printf("Pre kopiranja stringovi su bili ");
    if (strcmp(first_string,second_string))
        printf("razliciti.\n");
    else
        printf("isti.\n");

    strcpy(first_string,second_string);
    printf("Nakon kopiranja stringovi su ");
    if (strcmp(first_string,second_string))
        printf("razliciti.\n");
    else
        printf("isti.\n");

    strcat(first_string,second_string);
    printf("Nakon spajanja (concatenation), prvi string je: ");
    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*** ):

```
Unesi prvi string: Hello class.
Unesi drugi string: Hello Rob.
Pre kopiranja stringovi su bili razliciti.
Posle kopiranja stringovi su isti.
Nakon spajanja (concatenation), prvi string je: Hello Rob.Hello Rob.
```

# PRONALAŽENJE NAJDUŽEG ULAZNOG TEKSTA (9 MIN)

*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 nastavku je dat program koji pronalazi najveći uneti tekst:

```
#include <stdio.h>
#define MAXLINE 1000 /* maksimalna duzina ulazne linije */

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

void main()
{
    int len; /* duzina tekuce linije */
    int max; /* trenutno maksimalna duzina linije */
    char line[MAXLINE]; /* tekuca linija sa ulaza */
    char longest[MAXLINE]; /* najduza linija se ovde cuva */
    max = 0;
    while ((len = getline(line, MAXLINE)) > 0)
    {
        if (len > max)
        {
```

```
            max = len;
            copy(longest, line);
        }
    }

    if (max > 0) /* ukoliko je uneta validna linija */
        printf("%s", longest);

    return 0;
}

int getline(char s[],int lim)
{
    int c, i;
    for (i=0; i < lim-1 && (c=getchar())!=EOF && c!='\n'; ++i)
        s[i] = c;
    if (c == '\n')
    {
        s[i] = c;
        ++i;
    }
    s[i] = '\0';
    return i;
}

void copy(char to[], char from[])
{
    int i;
    i = 0;
    while ((to[i] = from[i]) != '\0')
        ++i;
}
```

# Poglavlje 9

# Zadaci za samostalan rad

## ZADACI ZA SAMOSTALNO VEŽBANJE

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

**Zadatak 1**. Napisati C program koji generiše niz neparnih brojeva a potom taj niz množi sa 2 i prebacuje u novi niz. (5 min)

**Zadatak 2**. Napravi niz brojeva od 100 elemenata. Svaki član se računa kao (2*i)* (2*i). Ispisati sve elemente ovog niza (10 min)

**Zadatak 3**. Napisati C program koji generiše niz neparnih brojeva a potom taj niz množi sa 2 i prebacuje u novi niz. Pronaći najmanji element niza. (10 min)

**Zadatak 4**. Napisati Cprogram 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. (10 min)

**Zadatak 5**. Napisati Cprogram koji ispisuje 10 random karaktera na ekranu sa prvim velikim slovom. Napomena: Koristiti funkciju *rand* biblioteke *stdlib.h* i pretvaranje brojeva u karaktere. (10 min)

## DODATNI ZADACI ZA SAMOSTALNI RAD

*Na osnovu materijala sa predavanja i vežbi, samostalno rešiti sledeće zadatke*

**Zadatak 1**. Napisati C program koji čita tekst sa standardnog ulaza i svaku liniju teksta ispisuje na standardni izlaz šifrirane po šemi koja utiče samo na slova:

(15 min)

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
```

**Zadatak 2**. Napisati program koji traži od korisnika da preko tastature popuni niz od 10 brojeva brojevima od 99 do 999 (program traži unos prvog broja

koji će biti unet u niz, nakon uspešnog unosa traži 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 traži da unese još 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. (15 min)

**Zadatak 3.** Napisati program za sortiranje elemenata niza u opadajućem poretku. Napisati glavni program i proceduru za sortiranje. (15 min)

# Poglavlje 10

# Domaći zadatak

## PRAVILA ZA DOMAĆI ZADATAK

*Detaljno proučiti pravila za izradu domaćih zadataka*

Svaki student dobija od asistenta sopstvenu kombinaciju domaćeg zadatka.

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

Odgovarajući Visual Studio (NetBeans ili CodeBlocks) projekat koji predstavlja rešenje domaćeg zadatka smestiti u folder CS130-DZ03-Ime-Prezime-BrojIndeksa. Zipovani folder CS130-DZ03-Ime-Prezime-BrojIndeksa poslati predmetnom asistentu (lazar.mrkela@metropolitan.ac.rs) u mejlu sa naslovom (subject)CS130-DZ03, inače se neće računati.

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

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

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

Vreme izrade: 1,5h.

# Zaključak

## REZIME

### *Na osnovu svega obrađenog možemo da izvedemo sledeći zaključak:*

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-a, 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”*** .

## REFERENCE

### *Korišćena literatura*

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

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

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

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

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

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

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

[8] Y. Daniel Liang, Introduction to Programming with C++, 3rd edition, Pearson, 2014.