Računske vježbe 1

Programiranje II

1. Implementirati funkcije koje određuju:
   - maksimalni elemenat niza,
   - veći od dva cijela broja,
   - maksimalni od dva stringa (leksikografski),
   - ,,maksimalni" od dva karaktera,
   - za jedan string, karakter sa najvećom ASCII vrijednošću.

Definisanje funkcija u programskom jeziku C++ ostvaruje se naredbama za definisanje funkcije čiji je opšti oblik:

```
oznaka_tipa naziv_funkcije ( niz_parametara ) tijelo_funkcije
```

U odnosu na programski jezik C u kojem navođenje prototipa funkcije nije bilo obavezujuće (mada se snažno savjetovalo da se prototip ne izostavlja), u programskom jeziku C++ prototip je obavezan. Deklarisanje funkcije postiže se naredbama oblika:

```
oznaka_tipa naziv_funkcije ( niz_parametara ) ;
```

gdje se u nizu parametara njihovi identifikatori mogu izostaviti. Obratiti pažnju da deklaracija funkcije ne sadrži njeno tijelo. Rezultat deklarisanja funkcije jeste njen prototip. Prototip mora biti naveden kada se neka funkcija poziva na mjestu na kome definicija funkcije nije dostupna kompajleru.

U programskom jeziku C++ deklaracija funkcije mora da sadrži tip vrijednosti funkcije, ali i tip svakog parametra. Ovo je neophodno zbog toga što C++ podržava preklapanje/preopterećivanje funkcija (engl. *function overloading*). Kako bi kompajler izvršio razrješenje poziva, mora znati koje su mu sve preklopljene funkcije na raspolaganju. Naravno, definicija i deklaracija se mogu spojiti.

Kao i u programskom jeziku C, program počinje sa pretprocesorom. Uključićemo nekoliko zaglavlja:

```
#include <iostream>
//#include <string.h>
#include <cstring>
```

gdje se **iostream** koristi za ulaz/izlaz i ima prednost u odnosu na **stdio.h** zbog OO orjentacije. Slična je i sudbina zaglavlja **string.h** čija se upotreba obeshrabruje, a koje sada mijenja **cstring** pomoću kojeg dobijamo pristup funkcijama za C stringove. Obratite pažnju da su svi nazivi na engleskom jeziku što je programerska praksa i čega ćemo se pridržavati na ovim vježbama, ali u provjerama znanja nije obavezujuće.

U dijelu pretprocesora imamo još jedan novitet:

```
using namespace std;
```

pomoću kojeg dobijamo pristup prostoru imena (engl. *namespace*) **std** iz već uključenog *iostream*. Da pojednostavimo:

```
std::cout<<maximum(a, n)<<std::endl;
cout<<maximum(a, n)<<endl;
```

ove dvije naredbe su identične, mada nam upotrebu druge omogućava uključivanje prostora imena. U kompleksnim programima ova praksa se izbjegava zbog mogućnosti greške, ali u našem slučaju nam značajno štedi nepotrebno kucanje i povećava preglednost. Obratiti pažnju da smo u slučaju funkcije:

```
char * maximum(char *, char *);
```

vratili pokazivač na niz karaktera jer, kao i u programskom jeziku C, niz ne može biti vrijednost funkcije. Neke od funkcija počinju sa ključnom riječju **inline** čime se zapravo tijelo funkcije kopira na svim mjestima gdje se ona poziva tj. ugrađuje direktno u kod programa. Dobra je praksa da to budu kraće i jednostavnije funkcije jer njihovom pretjeranom upotrebom raste EXE verzija programa. U zadatku je korišćen i ternarni operator:

```
return (izraz) ? a : b; // ako je izraz tacan vrati a, u suprotnom vrati b
```

Rješenje zadatka je:

```
#include <iostream>
//#include <string.h>
#include <cstring>

using namespace std;

int maximum(int *, int);
inline int maximum(int, int);
char * maximum(char *, char *);
inline char maximum(char, char);
char maximum(char*);

int main()
{
    int a[] = {1,5,4,3,2,6}, n = 6;
    cout << maximum(a, n) << endl;
    cout << maximum(2, 5 )<< endl;
    cout << maximum("string", "program")
   << endl;
    cout << maximum('a', 'A') << endl;
    cout << maximum("string");
}

int maximum(int *a, int n)
{
    int m;
    m = a[0];
    for(int i = 1; i < n; i++)
    if(a[i] > m)
        m = a[i];
    return m;
}
```

```
inline int maximum(int a, int b)
{
    return (a >= b) ? a : b;
}

char * maximum(char *a, char *b)
{
    if(strcmp(a,b) >= 0) return a;
    return b;
}

inline char maximum(char a, char b)
{
    return (a >= b) ? a : b;
}

char maximum(char *a)
{
    char m;
    m = a[0];
    for(int i = 1; i < strlen(a); i++)
        if(a[i] > m)
            m = a[i];
    return m;
}
```

2. Date su deklaracije funkcija:

   int f(int, char='0');
   int f(char, char);
   int f(double);

   Šta će se desiti nakon sljedećih poziva funkcije:

   f('0');
   f(2.34);
   f('c', 3);

Za razliku od programskog jezika C gdje smo funkcije realizovali po pravilu jedan problem - jedna funkcija odnosno više sličnih problema - više sličnih funkcija (što nikako nije dobro), C++ nam nudi da preklopimo funkcije i napišemo daleko elegantnija i održivija rješenja. Međutim, preklapanje funkcija može dovesti do ogromnih problema ukoliko se ne koristi na pravi način. Ovi problemi mogu nastati usljed implicitne konverzije argumenata i razrješenja poziva.

U prvom slučaju je očigledno da će se pozvati prva funkcija. Zašto? Druga funkcija otpada zato što ima dva obavezujuća argumenta. Mada je parametar prve funkcije tipa *int* on se standardnom konverzijom može dobiti implicitnim konvertovanjem karaktera u cjelobrojnu vrijednost. Funkcija će se pozvati kao:

```
f('0', '0');
```

Kod drugog poziva će se pozvati treća funkcija zato što je došlo do potpunog poklapanja argumenata tipa *double*. Međutim, u slučaju trećeg poziva imamo malo komplikovaniju situaciju. Prva funkcija ima 0 potpunih i 2 preklapanja konverzijom dok druga funkcija ima 1 potpuno preklapanje i 1 sa konverzijom podataka. Zapamtiti sva pravila razrješenja poziva i znati ih primijeniti nije lako te se uvijek savjetuje da se kada je god to moguće izbjegavaju sporne situacije.

3. Dat je niz cjelobrojnih elemenata. Kreirati funkciju koja podrazumijevano obrće elemente niza "naopačke". Ako je kao argument zadat pozitivan cijeli broj, potrebno je formirati novi niz koji predstavlja elemente starog niza od prvog elementa, sa datim preskokom do kraja. Ako je zadat negativan broj, potrebno je formirati novi niz koji ide od posljednjeg elementa ka početku sa odgovarajućim zadatim preskokom. Podrazumijevana vrijednost preskoka je -1.

U programskom jeziku C upoznali smo se sa naredbama **malloc** i **free** pomoću kojih smo alocirali i dealocirali memoriju. Mada ih i dalje možemo koristiti, programski jezik C++ ima operatore ugrađene u sam jezik koji su više objektno orjentisani. To su **new** i **delete**. Pomoću njih možemo zauzeti i osloboditi memoriju. U slučaju našeg zadatka potrebno je formirati novi niz, odnosno:

```
xNew = new(int[n]);
```

nakon što smo zauzeli odgovarajuću memoriju, potrebno ga je po zahtjevima zadatka popuniti. Funkcija će imati podrazumijevanu vrijednost -1. Napomena, vrijednost podrazumijevanog argumenta se mora unijeti unutar deklaracije funkcije. Zašto? Kao rezultat funkcije vraćamo novu dužinu niza.

C++ kao neki drugi jezici ne posjeduje jedinstven standard za imenovanje funkcija i promjenljivih. Ogromne kompanije kao što su Google i Microsoft definišu interne standarde koji su javno dostupni. Način na koji pišemo kod ne utiče na njegove performanse, ali programiranje je timski posao. Standardi smanjuju greške zbog bolje preglednosti i omogućavaju nam da se lakše upoznamo sa tuđim kodom. U ovom zadatku koristili samo **camelCase** za nazive funkcija i promjenljivih, a pored ovoga postoji i još nekoliko standarda o kojima se možete informisati. Poštovanje ove prakse se neće ocjenjivati, ali će vam biti od koristi!

Rješenje zadatka je:

```
#include <iostream>

using namespace std;

int reorder(int *, int, int *, int = -1);

int main()
{
    int x[10], n, nNew;
    int i, *xNew;

    cout << "Unijeti duzinu niza: ";
    cin >> n;
    cout << "Unijeti elemente niza: ";
    for(i = 0; i < n; i++)
        cin >> x[i];

    xNew = new(int[n]);

    cout << "Unijeti niz je: " << endl;
    for(i = 0; i < n ; i++) cout << x[i] << " ";
    cout << endl;

    nNew = reorder(x, n, xNew);
    for(i = 0; i < n; i++) cout << xNew[i] << " ";
    cout << endl;

    nNew = reorder(x, n, xNew, 3);
    for(i = 0; i < nNew; i++) cout << xNew[i] << " ";
    cout << endl;

    nNew = reorder(x, n, xNew, -2);
    for(i = 0; i < nNew; i++) cout << xNew[i] << " ";
    cout << endl;

    delete [] xNew; // dobra praksa!
}

int reorder(int *x, int n, int *xResult, int step)
{
    int j = 0;
    if(step > 0)
        for(int i = 0; i < n; i += step) xResult[j++] = x[i];
    else
        for(int i = n - 1; i >= 0; i += step) xResult[j++] = x[i];
    return j;
}
```

Pažnja! Većina modernih OS-ova će osloboditi dinamički zauzetu memoriju na kraju programa iako mi to eksplicitno ne naglasimo, međutim programeri se na to ne mogu osloniti te je obavezno osloboditi dinamički zauzetu memoriju kada nam više nije potrebna. Statički zauzeta memorija se čuva u strukturi podataka koja se zove **stack** i o njoj vodi brigu OS. Dinamički zauzeta memorija se čuva u strukturi podataka koja se zove **heap**. Statička memorija se zauzima prije izvršavanja programa dok se dinamička zauzima za vrijeme njegovog trajanja. Postoji sintaksna razlika kada iz memorije brišemo niz, a kada brišemo jedinstvenu vrijednost. Ova sintaksna razlika leži u činjenici da brisanje jedinstvenog objekta se izvršava mnogo brže nego li brisanje čitavog niza, to su dvije skroz različite operacije. Dakle, vodite računa da se upotreba ova dva operatora razlikuje i da će se program srušiti ako ih pobrkamo.