Функциски шаблони

Пр. наоѓање најголем од три цели броја, три реални броја, три знаци и три зборови со користење на функции за преклопување

```
#include <iostream>
#include <string>
using namespace std;
int najgolem (int, int, int);
double najgolem(double, double, double);
char najgolem(char, char, char);
string najgolem(string, string, string);
int main()
{
   int a,b,c;
   double d,e,f;
   char g,h,i;
   string j,k,l;

   cout << "vnesi tri celi broja" << endl;
   cin>>a>>b>>c;
   cout<<"najgolem e "<<najgolem(a,b,c)<<endl;

    cout << "vnesi tri realni broja" << endl;
   cin>>d>>e>>f;
   cout<<"najgolem e "<<najgolem(d,e,f)<<endl;

    cout << "vnesi tri znaci" << endl;
   cin>>g>>h>>i;
   cout<<"najgolem e "<<najgolem(g,h,i)<<endl;

    cout << "vnesi tri zbora" << endl;
   cin>>j>>k>>l;
   cout<<"najgolem e "<<najgolem(j,k,l)<<endl;
   return 0;
}

int najgolem (int a1, int a2, int a3)
{
   int n=a1;
   if(a2>n) n=a2;
   if(a3>n) n=a3;
   return n;
}

double najgolem(double a1, double a2, double a3)
{
   float n=a1;
   if(a2>n) n=a2;
   if(a3>n) n=a3;
   return n;
```

```
}
char najgolem(char a1, char a2, char a3)
{
   char n=a1;
   if(a2>n) n=a2;
   if(a3>n) n=a3;
   return n;
}

string najgolem(string a1, string a2, string a3)
{
   string n=a1;
   if(a2>n) n=a2;
   if(a3>n) n=a3;
   return n;
}
```

Во примерот од минатиот час за наоѓање најголем има четири преклопени функции и секоја посебно е дефинирана. Дали може наместо четири да се напише една функција?

Ова е можно само ако преклопените функции извршуваат исти операции над различен тип на податоци, како во претходниот пример. Таква функција се нарекува **ФУНКЦИСКИ ШАБЛОН** и пред насловот го содржи резервираниот збор template и листа на параметри ставени во аглести загради:

template <листа на параметри>

Листа на параметри содржи тип на параметарот пред кој се наведува резервираниот збор typename.

Пр.

```
template <typename T>
template <typename T1,  typename T2>
typename <typename ucenik, typename student>
```

Функциски шаблон за претходниот пример е следниов:

```
#include <iostream>
#include <string>
using namespace std;

template <typename T>
T najgolem (T a1, T a2, T a3)
{
   T n=a1;
   if(a2>n) n=a2;
   if(a3>n) n=a3;
   return n;
}


int main()
{
   int a,b,c;
   double d,e,f;
   char g,h,i;
```

```
    string j,k,l;

    cout << "vnesi tri celi broja" << endl;
    cin>>a>>b>>c;
    cout<<"najgolem e "<<najgolem(a,b,c)<<endl;

     cout << "vnesi tri realni broja" << endl;
    cin>>d>>e>>f;
    cout<<"najgolem e "<<najgolem(d,e,f)<<endl;

     cout << "vnesi tri znaci" << endl;
    cin>>g>>h>>i;
    cout<<"najgolem e "<<najgolem(g,h,i)<<endl;

     cout << "vnesi tri zbora" << endl;
    cin>>j>>k>>l;
    cout<<"najgolem e "<<najgolem(j,k,l)<<endl;
    return 0;
}
```

Типот на параметрите и на повратната вредност е ист – T. Аргументите за повик на функцијата не се читаат туку се иницијализираат. При азвршување на апликацијата, резултатот во двете е ист.

Дефиницијата на функциски шаблон е слична со дефиницијата на било која функција со параметри, но е проширена со параметри на типовите податоци наречени **параметри на шаблонот** или **формални параметри на тип.**

При повикот на функцискиот шаблон се врши замена на параметрите на шаблонот со соодветните аргументи на повикот и се врши замена на типот на параметрите со соодветниот тип на аргументите. Притоа се генерира посебна функција со изворен код и со соодветниот тип од повикот кој се нарекува **специјализација на функциски шаблон.**

Значи во зависност од типот на аргументите на повикот се генерира фамилија на преклопени функции, бидејќи се со исто име а различен број на параметри.

Пр. Програма која користи функциски шаблон за подредување на низа по растечки редослед.

```
#include <iostream>
#include <string>
using namespace std;

template <typename tipel, typename celbroj>
void zamena(tipel x[], celbroj a, celbroj b)
{
   tipel pom;
   pom=x[a];
   x[a]=x[b];
   x[b]=pom;
}

template <typename tipel, typename celbroj>
void sortiranje(tipel x[], celbroj n)
{
```

```
    celbroj i,j;
    for(i=0;i<n-1;i++)
    {
        for(j=i+1;j<n;j++)
        {
            if(x[i]>x[j]) zamena(x,i,j);
        }
    }
}

template <typename tipel, typename celbroj>
void pecati(tipel x[], celbroj n)
{
    celbroj i;
    for(i=0;i<n;i++)
    cout<<x[i]<<"    ";
    cout<<endl;
}

int main()
{
    int i, a[100],n;
    float b[100];
    char c[100];
    string d[100];
    cout<<"niza so elementi celi broevi"<<endl;
    cout<<"vnesi broj na elementi"<<endl;
    cin>>n;
    cout<<"vnesi gi elementite "<<endl;
    for(i=0;i<n;i++)
        cin>>a[i];
        sortiranje(a,n);
    cout<<"podredena nizata e "<<endl;
    pecati(a,n);

     cout<<"niza so elementi realni broevi"<<endl;
    cout<<"vnesi broj na elementi"<<endl;
    cin>>n;
    cout<<"vnesi gi elementite "<<endl;
    for(i=0;i<n;i++)
        cin>>b[i];
        sortiranje(b,n);
    cout<<"podredena nizata e "<<endl;
    pecati(b,n);

     cout<<"niza so elementi znaci"<<endl;
    cout<<"vnesi broj na elementi"<<endl;
    cin>>n;
    cout<<"vnesi gi elementite "<<endl;
    for(i=0;i<n;i++)
        cin>>c[i];
        sortiranje(c,n);
```

```
    cout<<"podredena nizata e "<<endl;
    pecati(c,n);

    cout<<"niza so elementi string"<<endl;
    cout<<"vnesi broj na elementi"<<endl;
    cin>>n;
    cout<<"vnesi gi elementite "<<endl;
    for(i=0;i<n;i++)
       cin>>d[i];
       sortiranje(d,n);
    cout<<"podredena nizata e "<<endl;
    pecati(d,n);
}
```

Пр. Програма која користи функциски шаблон за одредување најмал елемент во низа.

```
#include <iostream>
#include <string>
using namespace std;

template <typename tipel, typename celbroj>
int najmal(tipel x[], celbroj a)
{
    tipel najm=x[0];
    int i;
    for(i=1;i<a;i++)
    {
    if(x[i]<najm) najm=x[i];
    }
    return najm;
}

int main()
{
    int a[100],i,n;

    cout<<"niza so elementi celi broevi"<<endl;
    cout<<"vnesi broj na elementi"<<endl;
    cin>>n;
    cout<<"vnesi gi elementite "<<endl;
for(i=0;i<n;i++)
    cin>>a[i];
    cout<<"najmal element e "<<najmal(a,n);

}
```