Napredne tehnike programiranja

Vježba 11

Teme

- Predlošci funkcija
- Predlošci klasa

Generičko programiranje

- Generičko programiranje - programiranje koje nam dozvoljava stvaranje funkcija i struktura (klasa) “koje ne ovise o tipu”
- Predlošci –osnova za generičko programiranje
  - Predložak je ustvari formula ili recept za stvaranje strukture ili funkcije
  - Postoje funkcijski predlošci i predlošci klasa

Uvod

- U jeziku C++ u pri definiciji funkcije moramo navesti tipove parametara

```
int max( int a, int b )
```

- Pozovemo li `max(2,3)`, rezultat će biti 3
- Međutim, pozovemo li `max(2.1,5.3)`, rezultat će biti 5 (ili ćemo dobiti grešku).
- Da to izbjegnemo, trebali bi definirati i funkciju:

```
double max( double a, double b );
```

Preopterećene funkcije

- C++ dopušta korištenje preopterećenih

(overloaded) funkcija. To su funkcije koje imaju jednaka imena (i pripadaju istom dosegu namespaceu), ali imaju različitu listu parametara

```
int max( int a, int b );
double max( double a, double b );
string max( string a, string b );
int max( int *p, int size );
```

Preopterećene funkcije

- To smo već vidjeli kod konstruktora (konstruktor bez parametara, s jednim parametrom, dva...)

```
string();

string( string s );

string( char* s );

string( int count, char c );
```

Problem

Preopterećivanje funkcije ima nekoliko nedostataka:

- Ukoliko želimo nešto promijeniti u kodu funkcije, moramo to učiniti na puno mjesta, pa se povećava i mogućnost pogreške
- Ne možemo predvidjeti sve tipove argumenata za koje će korisnik htjeti pozvati funkciju

Primjer

- Napišite funkcije `ispisi` koje ispisuju "Vrijednost je:", te parametre na ekran i prelazi u novi red
- Ako je parametar niz, treba ispisati u vitičastim zagradama sve elemente niza odvojene zarezom

```
void ispisi( int x );
void ispisi( double x );
void ispisi( string s );
void ispisi( int* p, int size ); // int polje
void ispisi( double* p, int size ); // double polje
void ispisi( string* p, int size ); //string polje
```

- Primjer:

```
ispisi( 3 );          // Vrijednost je: 3
ispisi( "RP1" );      // Vrijednost je: RP1
ispisi( 4.9 );        // Vrijednost je: 4.9
int a[] = { 5, 6, 7 };
ispisi( a, 3 );       // Vrijednost je: {5, 6, 7}
```

Generičke funkcije

- Najbolje rješenje kad želimo napisati funkciju koja može primiti argumente različitih tipova su generičke funkcije
- Generička funkcija je predložak funkcije koji prihvaća različite tipove argumenata
- Općeniti oblik:

```
template <argument_predloška> deklaracija funkcije;
```

Generičke funkcije

- Definicija predloka funkcije započinje se ključnom riječi `template`
- Svaki predložak ima listu formalnih parametara koja se navodi između znakova < i > (lista ne može biti prazna)
- Svaki formalni parametar se sastoji od ključne riječi `class` iza koje se navodi identifikator
- Identifikator je ugrađeni ili korisnički tip koji će se navesti prilikom poziva funkcije

Generičke funkcije - primjer

- Umjesto:

```
int kvadrat (int x) {return x*x}
float kvadrat (float x) {return x*x}
double kvadrat (double x) {return x*x}
```

- možemo općenito pisati ovako:

```
T kvadrat(T x){ return x*x;};
```

T – neki općeniti tip

Generičke funkcije - primjer

- Umjesto:

```
T kvadrat(T x){ return x*x;};
```

- moramo staviti oznaku da se radi o predlošku funkcije (dodamo `template` na početak), pa pišemo:

```
template <class T> T Kvadrat(T x){return x*x;}
```

ili

```
template <typename T>T Kvadrat(T x){return x*x;}
```

Generičke funkcije - primjer

- Umjesto:

```
int kvadrat (int x) {return x*x}
float kvadrat (float x) {return x*x}
double kvadrat (double x) {return x*x}
```

- možemo općenito pisati ovako:

```
template <class Tip>
Tip kvadrat(Tip x){
      return x*x;
};
```

Generičke funkcije - primjer

- Kad pozivamo generičku funkciju preporuka je da eksplicitno napišemo za koji tip je pozivamo

```
template <class Tip>
Tip kvadrat(Tip x){
     return x*x;
};

int main(){
int a=5;
cout<<kvadrat(a)<<endl; //radi
cout<<kvadrat<int>(a)<<endl; //preporuča se
```

Generičke funkcije

- Predložak funkcije se ne koristi sve dok kompajler ne naiđe na poziv generičke funkcije
  - Tek tada se stvara i prevodi nova varijanta funkcije ovisno o tipu na kojem je funkcija pozvana
  - Taj proces stvaranja nove konkretne varijante funkcije iz funkcijskog predloška naziva se instancijacija
  - Tipovi se kod funkcijskih predložaka prepoznavaju automatski

Primjer

- Iz funkcije min (koja vraća manji od dva argumenta napiši predložak funkcije

```
int min(int a, int b ) {
            return a < b ? a : b;
        };
```

Generičke funkcije

- Primjer:

```
template <class T>
T min(T a, T b ) {
      return a < b ? a : b;
};
```

Kao parametar funkcije min možemo koristiti bilo koji tip (za kojeg postoji operator <)

- Parametri predloška nalaze se unutar znakova <> i mogu predstavljati:
  - tip (template type parameter)
  - konstantni izraz (template nontype parameter)

Generičke funkcije - primjer

```
template <class T>
T min( T a, T b ) {
return a < b ? a : b;
};
...
min( 1, 2 ); // ok: int min(int,int)
min( 1.0,2.1 ); // ok: min(double,double)
min( string( "rp1" ), string( "rp2" ) );
// ok: min(string,string)
min( 1, 2.3 ); // greška: min(int, double)
```

Generičke funkcije

- Mogu se definirati generičke funkcije s više generičkih tipova:

```
template <class T1, class T2>
T1 min(T1 a, T2 b ) {
     return a < b ? a : b;
};
```

Primjer

- Formalni parametar se pojavljuje samo jednom u predlošku

```
template <class T, class T> // pogreška
void func(T a, T b) {
// ...
}
```

- Ispred svakog parametra je naredba `class`

```
template <class T1, T2> // pogreška

template <class T1,class T2> // ispravno
```

Generičke funkcije - primjer

- Funkcija nalazi najmanji element niza:

```
template <class Type>
Type min_niz( Type *T, int count ) {
      Type min = T[0];
      for( int i = 1; i < count; ++i )
            if( T[i] < min ) min = T[i];
      return min;
}
...
int a[] = { 9, 5, 7 };
int m = min_niz( a, 3 );
```

Primjer

- Napišite generičku implementaciju (u obliku funkcijskog predloška) funkcije `zamijeni` koja zamjenjuje vrijednosti dva dobivena argumenta

Rješenje

```
template <class Type>
void zamijeni (Type &prvi, Type &drugi){
     Type temp=prvi;
     prvi=drugi;
     drugi=temp;
}
```

Primjer

- Napišite generičku funkciju `sortiraj` koja prima dva parametra. Prvi je pokazivač na proizvoljan tip, a drugi je `int`. Prvi parametar pokazuje na nulti element polja, a drugi je broj elemenata polja. Funkcija treba sortirati elemente polja. Iskoristite zamijeni iz prethodnog zadatka
- U main-u pomoću ove funkcije sortirajte niz int-ova i string-ova

Rješenje za int polje

```
void sortiraj(int a[], int n){
	int zamjena;
	do{
		zamjena = 0;
		for(int i = 0; i < n - 1; i++)
		{
			if (a[i] > a[i + 1]){
				int pomocni = a[i];/* zamijeni dva */
				a[i] = a[i + 1];   /* susjedna */
				a[i + 1] = pomocni; /* elementa */
				zamjena = 1;
			}
		}
		n--;  /* skrati polje za jedan element */
	}while (zamjena != 0);
}
```

Rješenje

```
template <class T>
void sortiraj(T a[], int n){
	int zamjena;
	do{
		zamjena = 0;
		for(int i = 0; i < n - 1; i++){
			if (a[i] > a[i + 1]){
				T pomocni = a[i];
				a[i] = a[i + 1];
				a[i + 1] = pomocni;
				zamjena = 1;
			}
		}
		n--;  /* skrati polje za jedan element */
	}while (zamjena != 0);
}
```

Predložak klase

- C++ nam dopušta i parametriziranje klasa
- Klasa sadrži tip koji je parametriziran
- Generičke funkcije i klase se implementiraju u .h datoteci (ne u implementacijskoj), jer se kompajliraju ovisno o klijentskom programu

Predložak klase

- C++ nam dopušta i parametriziranje klasa
- Primjer predloška strukture:

```
template<class T> class stog {
        T podaci[100];
        int vrh;
        stog() { vrh = 100; }
        T top() { ... return podaci[vrh]; }
        void push( T data ); //dekl. funkcije
};
template<class T> //implementacija funkcije
void stog<T>::push( T data )
{ podaci[vrh++]=dana; }
```

Eksplicitno instanciranje

- Kod generičkih klasa kod deklaracije moramo eksplicitno navesti tip
- Primjer deklaracije:

```
stog <int> S;
stog <string> T;
```

- I korištenja strukture

```
S.push( 10 );
T.push( "RP1" );
```

Defaultne vrijednosti

- Za tip i za vrijednost parametara možemo zadati defaultne vrijednosti

```
template <class T=int, int N=10> class polje{...}
```

- moguće deklaracije:

```
polje<> a;                  // polje od 10 int
polje<float> b;             // polje od 10 float
polje<char,100> c;          // polje od 100 char
```

Zadatak

- Napišite generičku strukturu `par` koja predstavlja uređen par dva podatka.
- Struktura ima dva parametra. Prvi je tip prvog parametra, a drugi tip drugog parametra.
- Treba imati konstruktor koji kao parametre prima prvi i drugi član
- Treba imati funkcije prvi() i drugi() koje vraćaju prvi odnosno drugi element para
- Treba imati funkcije postaviprvi() i postavidrugi() koje postavljaju prvi odnosno drugi element para

Rješenje

```
template <class T1, class T2>
class par
{
  private:
    T1 v1;
    T2 v2;
  public:
    par (T1 prvi=0, T2 drugi=0)
    {v1=prvi; v2=drugi;}

    T1 prvi () {return v1;}
    T2 drugi () {return v2;}

    void postaviprvi (T1 val) {v1 = val;}
    void postavidrugi (T2 val) {v2 = val;}
};
```

Zadatak

- Napravite generičku strukturu `stack` koja će se moći ovako upotrebljavati:

```
stack<int> stogIntova;
stack<string> stogStringova;

stogIntova.push( 3 );
stogStringova.push( "abc" );
int a = stogIntova.top();
string b = stogStringova.top();
stogIntova.pop();
cout<<"broj
clanova:"<<stogIntova.size();
```