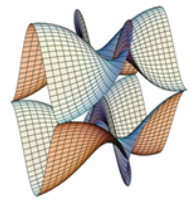

Prirodoslovno-matematički fakultet
Matematički odsjek
Sveučilište u Zagrebu

RAČUNARSKI PRAKTIKUM I

Vježbe 04 - Generičke funkcije i strukture

v2018/2019.

Sastavio: Zvonimir Bujanović

Preopterećivanje funkcija

U C-u nije dozvoljeno imati više funkcija istog imena.

C++ dozvoljava više funkcija istog imena, ako:

- funkcije imaju različit broj parametara, ili,
- funkcije se razlikuju u tipu barem jednog parametra.

Ovo zovemo preopterećivanje funkcija (engl. overloading).

```
int max( int a, int b ) { ... }
double max( double a, double b ) { ... }
int max( int polje[], int duljina ) { ... }
```

Pritom treba biti nedvosmisleno jasno koja se funkcija poziva.

```
a = max( 3, 2 );         // poziva 1. funkciju
b = max( 3.14, 2.71 ); // poziva 2. funkciju
c = max( polje, 5 );    // poziva 3. funkciju
d = max( 1, 2.3 );      // compile error: 1. ili 2. funkcija?
```

Preopterećivanje funkcija

Jedan primjer preopterećenih funkcija je i više konstruktora unutar iste strukture.

```
struct string
{
    ...

    string();
    string( string s );
    string( char *s );
    string( int count, char c );
};
```

Zadatak 1

- Implementirajte funkcije `ispisi` koje ispisuju `"Vrijednost je:"`, a zatim vrijednost primljenog parametra.
- Ako je parametar niz, sve njegove elemente treba ispisati unutar vitičastih zagrada i odvojene zarezom.

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

- Primjeri:

```
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}
```

Ista implementacija funkcija za različite tipove

Često se događa da je jedna te ista implementacija funkcije dobra za različite tipove parametara.

```
int max( int a, int b ) {
    if( a > b )
        return a;
    else
        return b;
}

double max( double a, double b ) {
    if( a > b )
        return a;
    else
        return b;
}
```

1. Kako izbjeći ponavljanje implementacije?
2. Kako automatski dobiti verziju za novi tip, npr. `string`?

Generičke funkcije

C++ uvodi generičke funkcije.

- Te funkcije su parametrizirane jednim ili više tipova podataka (u donjem kodu jednim tipom nazvanim `Type`).
- Funkcija će moći biti pozvana za bilo koji tip podatka `Type` za koji njezina implementacija rezultira smislenim kodom.
- Na primjer, donji kod je smislen za `Type=int`, `double`, `string`...
- Kod je smislen i za C-ovske stringove, `Type=char*`, ali funkcija neće vratiti abecedno veći od stringova `a` i `b`.

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

Generičke funkcije još zovemo i funkcijski predlošci (engl. template).

Generičke funkcije

Primjer korištenja generičkih funkcija.

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

a = max( 1, 2 );      // ok: Type = int
b = max( 1.0, 2.1 ); // ok: Type = double

// ok: Type = string
c = max( string( "rp1" ), string( "rp2" ) );

d = max( 1, 2.3 );    // compile error! Type = ???
```

Prilikom kompajliranja koda, kompajler će automatski generirati sve potrebne verzije funkcije `max`, za sve `Type` koje koristimo.

Generičke funkcije

Unutar funkcije smijemo deklarirati i nove varijable tipa `Type`.

```
template <class Type>
Type min_polje( Type *polje, int duljina )
{
    Type ret = polje[0];
    for( int i = 1; i < duljina; ++i )
        if( polje[i] < ret )
            ret = polje[i];
    return ret;
}
...
int a[] = { 9, 5, 7 };
int m = min_polje( a, 3 ); // ok: Type = int
```

Budući da već prilikom kompajliranja mora biti poznata cijela implementacija generičke funkcije, ako razdvajamo program u više datoteka onda se i definicija i implementacija takvih funkcija stavlja u .h datoteku!

Zadatak 2

1. Napišite generičku implementaciju (odnosno, funkcijski predložak) funkcije `swap` koja zamjenjuje vrijednosti dva dobivena argumenta.

2. Testirajte napisanu funkciju na varijablama tipa `int`, `string`.

3. Zamjenjuje li ispravno funkcija varijable koji su C-ovski stringovi?

Zadatak 3

1. Napišite generičku implementaciju (u obliku funkcijskog predloška) funkcije `sort` koja prima dva parametra.
   - Prvi parametar je pokazivač na proizvoljan tip, a drugi je tipa `int`.
   - Prvi parametar sadrži adresu nultog elementa polja, a drugi predstavlja broj elemenata polja.

   Funkcija treba uzlazno sortirati elemente polja.

2. Testirajte napisanu funkciju na poljima elemenata tipa `int` i `string`.

Generičke strukture

Na analogan način moguće je napraviti i generičke strukture.

```
template<class T>
struct stack
{
    int size;
    T element[100];

    stack() { size = 0; }
    T top() { return element[size-1]; }
    void push( T data );
};

template<class T>
void stack<T>::push( T data )
{
    element[size++] = data;
}
```

Generičke strukture

Generičke funkcije

- Tip je obično moguće odrediti iz samog poziva funkcije, no možemo ga i eksplicitno navesti.

```
a = max( 3, 2 );          // Type = int
b = max<int>( 3.0, 2 );   // Type = int
```

Generičke strukture

- Tip treba eksplicitno navesti prilikom deklaracije.

```
stack<int> stogIntova;          // T = int
stack<string> stogStringova;    // T = string
```

- Nakon deklaracije je tip poznat, pa ga prilikom korištenja funkcija članica više ne treba navoditi.

```
stogIntova.push( 10 );
stogStringova.push( "RP1" );
```

Zadatak 4

- Implementirajte generičku strukturu `par` predstavlja uređen par koji se sastoji od dva podatka. Ta dva podatka mogu biti različitih tipova (npr. jedan `int` i jedan `string`).
- Struktura treba imati konstruktor koji prima dva parametra. Prvi će spremiti kao prvi član uređenog para, a drugi kao drugi član.
- Struktura treba imati funkcije `prvi()` i `drugi()` koje vraćaju prvi, odnosno, drugi član uređenog para.

Strukture i funkcije mogu biti parametrizirane i s više tipova.

```
template<class Type1, class Type2, class Type3>
struct Example
{
    ...
};

Example<int, string, string> X; // deklaracija varijable
```

Složeni tipovi u generičkoj strukturi

Generička struktura može biti parametrizirana bilo kojim tipom, pa i drugom generičkom strukturom.

```
// stog na kojem se nalaze uredjeni parovi int-a i string-a
stack<par<int, string>> S;

// par ciji je prvi clan stog int-ova, a drugi string
par<stack<int>, string> P;

// uf...kompliciranо, ali dozvoljeno
par<par<int, string>, stack<par<string, stack<int>>>> X;
```

C++11 - Dozvoljava >> u deklaraciji ugniježđene strukture.
U ranijim standardima treba dodati razmak: > >.

```
par<int, stack<string>>  P; // ok u C++11, nije ok prije
par<int, stack<string> > P; // ok u svim standardima
```

Složeni tipovi u generičkoj strukturi

Kod korištenja samo treba biti pažljiv s tipovima i raditi postepeno.

```
// "Komplicirani stog".
stack<par<int, stack<string>>> KS;

// Pripremimo element koji cemo staviti na KS.
// Tip tog elementa je par<int, stack<string>>.
// Da napravimo par, prvo nam treba int.
int a = 5;

// Drugi clan para je stog stringova.
stack<string> S;
S.push( "abc" ); S.push( "def" );

// Napravimo par.
par<int, stack<string>> P( a, S );

// Stavimo ga na "komplicirani stog".
KS.push( P );
```