Šablony funkcí a tříd (Templates)

Genericita

Šablony funkcí

Motivace

- přetížíme funkci pro výpočet minima ze dvou hodnot:

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

float minimum(float a, float b)
{
  return (a<=b)?a:b;
}
```

- obě funkce mají stejné tělo, liší se pouze typem parametrů
- nevýhoda:
  - píšeme dvakrát stejný kód
- jazyk C++ poskytuje možnost napsat tzv. ***šablonu funkce (template function, generickou funkci)***
  - tj. kód bez specifikace konkrétního typu parametrů a výsledku (nespecifikovaný parametr je generický parametr)

  - konkrétní funkce (generická instance) se vytvoří automaticky při překladu (při volání se skutečnými parametry)

- definice generické funkce (šablony)

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

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

void main(void)
{
  int a=5,b=7,c;
  float x=4.5,y=1.1,z;

  c = minimum(a,b);
  z = minimum(x,y);
}
```

vytvoří se a použije se funkce
int minimum(int,int)

vytvoří se a použije se funkce
float minimum(float,float)

Poznámka:

```
int x;
char c;
minimum(x,c)
```

- nelze použít, protože minimum(int,char) nelze vytvořit; šablona je definována tak, že oba parametry musí být stejného typu

- řešení:
  - definovat jinou šablonu, kde každý parametr je jiného typu

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

void main(void)
{
  int a=5,x;
  char c='A';

  x = minimum(a,c);
}
```

- pro parametry typu např. `int*` tato šablona nevyhovuje, protože by kód porovnával ukazatele; potlačit generování instance ze šablony je možné potlačit definicí negenerické funkce

```
int minimum(int *a, int *b)
{
  return (*a<=*b)? *a:*b;
}
```

```
void main(void)
{
  int *a = new int;
  int *b = new int;
  int x;
  *a = 3; *b = 4;
  x = minimum(a,b);
  delete a;
  delete b;
}
```

zde se nevytvoří funkce podle šablony, ale použije se negenerická

Šablony tříd

- definujeme třídu zásobník pro ukládání celých čísel

```
class TZas
{
  int delka,vrchol;
  int *zasob;
 public:
  int je_prazdny();
  int je_plny();
  int push(int prvek);
  int pop();
  int top();
  TZas();
  TZas(int velikost);
  ~TZas();
};
```

- pokud bychom chtěli využívat zásobník pro ukládání čísel typu **`float`:**
  - *řešení č. 1:*
    - definice nové třídy `TZasfloat`, která se bude lišit pouze typem atributu `zasob` (`float *zasob`) a napsání shodného kódu pro jednotlivé metody
  - *řešení č. 2:*
    - jazyk C++ umožňuje aplikovat princip genericity na třídy, tj. vytvořit ***šablonu třídy (generickou třídu)***
    - definice generické třídy pro zásobních s generickým parametrem pro prvek

```
template<class T>
class TZas
{
  int delka,vrchol;
  T *zasob;
 public:
  int je_prazdny();
  int je_plny();
  int push(T prvek);
  T pop();
  T top();
  TZas();
  TZas(int velikost);
  ~TZas();
};
```

- metody implementujeme podobně jako generické funkce; před každou musí být „informace“ o šabloně:

```
template <class T>
TZas<T>::TZas()
{
  delka = 80;
  vrchol = -1;
  zasob = new T[80];
}
```

```
template <class T>
int TZas<T>::push(T prvek)
{
  if (vrchol == delka-1) return false;
  zasob[++vrchol] = prvek;
  return true;
}
```

- atd

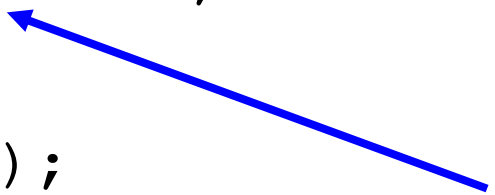
Použití:

```
void main(void)
{
   TZas<char> z1;
   TZas<float> z2;
   TZas<TKomplex> z3;

   z1.push('A');
   z2.push(1.5);
}
```

- generický parametr může zastupovat i konstantu

```
template<class T, int vel>
class TZas
{
  int delka,vrchol;
  T zasob[vel];
 public:
  int je_prazdny();
  int je_plny();
  atd…
  TZas()
  ~TZas();
};
```

```
void main(void)
{
   TZas<char,20> z1;

   z1.push('A');
}
```



konstanta musí být známa při překladu

POZOR

- definice metod generické třídy musí být „vidět“ v místě vytvoření generické instance
  - musí být v *.h* souboru
  - nebo v hlavním programu musíme vložit i soubor .cpp s kódem:

```
#include "TZas.h"
#include "TZas.cpp"
```

STL

Standard Template Library

Motivace

- třídy pro implementaci spojového seznamu, zásobníku, fronty, vektoru (abstraktní datové typy ADT) potřebuje každý programátor
- ADT – data + operace
- mechanismus šablon tříd v C++ umožňuje naprogramovat ADT univerzálně

Bylo by pěkné mít hotovou takovou knihovnu …

STL

- STL je knihovna šablon (generických tříd) implementující zmíněné datové typy
  - fronta, zásobník, seznam, vektor
- je součástí normy C++

- v STL jsou definovány tři skupiny generických tříd:
  - **kontejnery**
    - představují vlastní ADT – vektor, zásobník, fronta, string
  - **iterátory**
    - třídy, jejichž metody umožňují procházet kontejnery, např. nastavit se na první nebo poslední prvek, postoupit na další atd.
  - **algoritmy**
    - provádějí inicializaci, třídění, vyhledávání nad kontejnery

- každý kontejner má definován vlastní alokátor
  - alokátor řídí přidělování paměti pro kontejnery
  - standardní alokátor je instance třídy *allocator*, která je definována v STL
  - uživatel si může definovat vlastní alokátory

Základní kontejnery

Kontejner	Popis	Hlav. soubor
bitset	sada bitů	`<bitset>`
deque	oboustr. zakončená fronta	`<deque>`
list	lineární seznam	`<list>`
map	ukládá páry klíč/hodnota	`<map>`
multimap	ukládá páry klíč/hodnoty	`<multimap>`
multiset	sada (množina)	`<multiset>`
priority_queue	prioritní fronta	`<queue>`
queue	fronta	`<queue>`
set	sada, každý prvek jednozn.	`<set>`
stack	zásobník	`<stack>`
vector	vektor (dynamické pole)	`<vector>`

Iterátory

- objekty, jejichž metody slouží k „pohybu“ po prvcích kontejnerů
  - umožňují např. stejným způsobem procházet vektor, frontu od začátku do konce, od konce k začátku atd.
  - pracuje se s nimi jako s ukazateli (mají přetížené operátory *,++,--) – lze je dekrementovat, inkrementovat,…

- typy iterátorů:
  - dopředný (ForIter)
    - ukládá a získává hodnoty, pohyb vpřed
  - obousměrný (BiIter)
    - ukládá a získává hodnoty, pohyb vpřed i vzad
  - vstupní (InIter)
    - získává, ale neukládá hodnoty, pohyb pouze vpřed
  - výstupní (OutIter)
    - ukládá, ale nezískává hodnoty, pohyb pouze vpřed
  - přímý přístup (RandIter)
    - získává a ukládá hodnoty, přímý přístup k prvkům jako u pole

Poznámky:

- dopředný
  - nemá přetížený operátor **––**
- vstupní
  - nemůže být použit na levé straně přiřazovacího příkazu

Vektor

- deklarace v souboru <vector>:

```
template<class _Ty,class _Ax = allocator<_Ty> >
class vector
```

- konstruktory

```
vector(const _Alloc& _Al)
vector(size_type _Count, const _Ty& _Val)
vector(size_type _Count, const _Ty& _Val, const
    _Alloc& _Al)
```

Příklad metod

`reference front()`

- vrací odkaz na první prvek vektoru (reference je datový typ představující odkaz)
  - `typedef T& reference;`

`void clear()`

- odstraní všechny prvku vektoru

`iterator insert(iterator _Where, const _Ty& _Val)`

- vloží prvek Val před (?) prvek označený Where
- insert je 3x přetíženo

- `void insert(iterator _Where, const Ty& val = Ty());`
  - vloží 1 prvek
- `void insert(iterator _Where, size_type n, const Ty& val = Ty())`
  - vloží n kopií prvku
- `template<class InputIterator> void insert(iterator _Where, InputIterator zacatek, InputIterator konec);`
  - vloží prvky z jiného iterátoru

`reference back()`

- vrací referenci na poslední prvek

`void push_back()`

- vloží nový prvek "za" konec

`void pop_back()`

- odebere prvek z konce vektoru

Použití

```
#include <vector>
using namespace std;
void main(void)
{
  vector<int> v;
  // vytvori prazdny vektor
  for(int i=0;i<10;i++)  v.push_back(i);
  for(int i=0;i<10;i++)
    cout << v[i] << ' ';
}
```

Použití

```
include <vector>
using namespace std;
void main(void)
{
  vector<int> v(10);
  // vytvori vektor o 10 polozkach
  for(int i=0;i<10;i++)  v[i] = i;
  for(int i=0;i<10;i++)
    cout << v[i] << ' ';
}
```

Použití

```
include <vector>
using namespace std;
void main(void)
{
  vector<int> v(10,5);
  // nastavi vsechny prvky vektoru na 5
  for(int i=0;i<10;i++)  v[i] = i;
  for(int i=0;i<10;i++)
    cout << v[i] << ' ';
  cout << "\nPocet prvku ve vektoru: ";
  cout << v.size();
}
```

- přístup přes iterátor

```
vector<int>::iterator p=v.begin();
while(p != v.end())
{
  cout << *p << ' ';
  p++;
}
```

Poznámky:

- všechny kontejnery jsou ve jmenném prostoru std
- parametrem <bitset> je počet bitů, nikoliv typ

```
template <size_t N>
class  bitset
{
...
}
```

- každý kontejner má veřejné metody:
  - begin – vrací iterátor na počáteční prvek.
  - empty - vrací true, jestliže je kontejner prázdný.
  - end – vrací iterátor za poslední prvek
  - max_size - vrací maximální možnou velikost kontejneru.
  - size - vrací aktuální velikost kontejneru.
  - swap - zajistí výměnu prvků s jiným kontejnerem

Algoritmy

- funkce jsou definovány v hlavičkovém souboru `algorithm`
- dělení
  - algoritmy nepracující s kontejnery
    - min, max, swap – parametry jsou reference na typ
  - kopírující algoritmy
    - copy, copy_backward – kopírují části kontejnerů určené iterátory

- přesouvací algoritmy
  - remove, remove_if – odstraňují prvky
  - remove_copy, remove_copy_if – kopíruje prvky do jiného kontejneru
- vyhledávací algoritmy
  - find
- a další
  - generování permutací
    - `next_permutation(BidIt f, BidIt l)`

- pěkný přehled je na

  http://www.cplusplus.com/

Mapa

```
map<string, string> tabulka;

tabulka["420525/1234"]="Josef Novák";
tabulka["735510/1275"]="Marie Vokatá";
```

Klíč	Hodnota
420525/1234	Josef Novák
735510/1275	Marie Vokatá

- nelze přistupoval přes indexy `tabulka[0] tabulka[1]`, ale přes hodnotu klíče: `tabulka["420525/1234"]`
- projít všechny záznamy mapy lze pouze pomocí iterátoru

Mapa

```
map<string,string>::iterator p = tabulka.begin();

while (p!=tabulka.end())
{
  cout << (*p).first << " " << (*p).second;
  cout << endl;
  p++;
}
```

Ukázka

- generování permutací nad prvky v poli

Úkoly

Napište program, který využívá generický zásobník (stack) pro uložení znaků zadávaných z klávesnice (hlavičkový soubor stack). Vkládejte znaky zadávané z klávesnice na zásobník (posloupnost ukončete klávesou Enter), pak vypište jejich počet (velikost zásobníku) pomocí metody size. Odeberte všechny prvky ze zásobníku a vypište je.

Úkoly

Napište program, který vypočte součet dvou vektorů. Vektory načte z textového souboru. Jejich dimenze není v souboru zaznamenána, soubor obsahuje dva řádky čísel:

```
1 2 3 4
5 6 1 5
```

Návod:

Deklarujte dva vektory typu `vector<double>`. Načtete ze souboru první řádek do řetězce funkcí `getline` a pomocí objektu typu istringstream přečtěte z řetězce cyklem s podmínkou na začátku všechny hodnoty (testujte pomocí metody good()). Do vektoru je přidejte na konec metodou push_back(). Totéž proveďte pro

druhý vektor (druhý řádek v souboru). Testujte správnost konverze metodou fail(). Před vlastním sčítáním srovnejte dimenzi vektorů *a, b* (metodou size()) – pokud není stejná, ukončete program. Pak vektory sečtěte a vypište všechny na obrazovku.