X36PJC
4. přednáška

Základní datové typy
ze std: string, vector, iterátory

Namespace I.

- Jmenný prostor
- Namespace problém
- Řešení
  - Prefixace:

    class X36PJC_colections_vector {}
  - Deklarace namespace:

    namespace X36PJC{

    namespace collections{

    class vector {}; }}

Namespace II.

- Přístup pomocí operátoru ::
  - X36PJC::collections::vector vektor;
- Přístup pomocí using

  using namespace X36PJC::collections;

  vector vektor;
- nebo

  using namespace X36PJC;

  collections::vector vektor;
- nebo

  using namespace X36PJC::collections::vector;

string I.

- Řetězec znaků s proměnnou délkou
- Odstiňuje programátora od práce s pamětí
- Efektivní pro obecné použití
- Nabízí více než jen uložení textu
- Pro použití v kódu je třeba

  #include <string>
- a pro ulehčení práce

  using namespace std::string;

string II.

- Definice a inicializace

  string s1;

  string s2(s1);

  string s3(„text“);

  string s4(n, 'c'); // n je typu string::size_type

string III.

- Vstup a výstup pomocí string

  // vstupem je Hello world!

  cin >> s1 >> s2;

  // vystupem je Helloworld!

  cout << s1 << s2;
- Načtení celé řádky

  std::getline(cin, s3);

string IV.

- Operace se stringy

  s.empty();

  s.size(); // vraci typ string::size_type

  s[n];

  s1 + s2;

  s1 = s2;

  s1 == s2;

  !=, <, <=, >, >=

  s.c_str()

string V.

- Vybrané operace se stringy

```
s1 += s2;
s3 = s1 + ", " + s2 + "\n";
s4 = "hello" + ", "; // chyba
s5 = "hello" + ", " + s2; // chyba
s6 = string(„hello“) + ", "; // spravne
string s7("some string");
for (string::size_type ix = 0; ix != s7.size(); ++ix)
        cout << s7[ix] << endl;
s7[4] = '_';  // v s7 je text „some_string“
```

string VI.

- Operace se znaky
  - Dotazovací

    isalnum(c), isalpha(c), islower(c), …
  - Konvertovací

    tolower(c), toupper(c)

vector I.

- Univerzální objekt pro uložení skupiny prvků stejného typu
- Kontejnerovou třídu ze STL
- K prvkům lze přistupovat pomocí indexu
- Datový typ definovaný pomocí šablony (viz přednáška 10)
- Použití

  ```
  #include <vector>

  vector<type_name> v;
  ```

Vector II.

- Definice vektoru

  vector<int> v1;

  vector<string> v2(v1); // Chyba

  vector<int> v3(n, 20);

  vector<string> v4(10); // Co tam bude?
- Vektory jsou dynamické, není třeba definovat jejich velikost při definici

Vector III.

- Operace s vektorem

  v.empty()

  v.size()

  v.push_back()

  v[n]

  v1 = v2

  v1 == v2

  !=, <, <=,>, >=

vector IV.

- Velikost vektoru

  v.size()
- ...vrací datový typ vector::size_type
- Pozor!

```
vector<int> ivec; // Prazdny vektor
for (vector<int>::size_type ix = 0; ix != 10; ++ix)
        ivec[ix] = ix; // Chyba
```

- Přístup přes závorku nepřidá nové prvky!

Kontejnerové třídy

- Součástí standardní knihovny C++
- Typickým zástupcem je vector
- Obecně slouží pro uložení určitého množství objektů
- Rozlišení podle způsobu ukládání, vybírání a uložení objektů
- bitset, deque, list, map, multimap, multiset, queue, priority_queue, set, stack
- Kromě třídy vektor nemají přístup k prvkům přes závorky

Iterátory I.

- Slouží jako obecný přístup k jednotlivým prvkům uložených v kontejnerových třídách
- Nejsou konkurencí přístupu přes závorky
- Pomáhají zabránit programátorským chybám

  vector<int> ivec; // prázdný vektor

  cout << ivec[0];
- Datový typ iterátoru je dán typem kontejnerové třídy

  vector<type>::iterator iter;

Iterátory II.

- Získání iterátoru

```
vector<int> vec(10);
vector<int>::iterator i1 = vec.begin();
vector<int>::iterator i2 = vec.end();
```

- Dereference iterátoru

```
cout << *i1; // cout << vec[0];
cout << *i2; // cout << vec[9];
*i1 = 100; // vec[0] = 100;
```

Iterátory III.

- Operace s iterátory

  ==, !=

  i1++, ++i1, i1--, --i1

  i2 + n, i2 – n

  i1 – i2 // oba iteratory patri stejne instanci

- Cyklus s iterátory

```
for (vector<int>::iterator iter = ivec.begin();
                                    iter != ivec.end(); ++iter)
        *iter = 0;
```

Konstantní iterátory I.

- Slouží jen ke čtení objektů z kontejnerů
- vector<type>::const_iterator iter;
- Pozor! Neplést si s konstantou iterátoru:

  const vector<type>::iterator iter;
- Typické chyby

  vector<int>::const_iterator it = vec.begin();

  *it = 10; // Problem, const_iterator

Konstantní iterátory II.

- Typické chyby

```
const vector<int>::iterator i1 = v1.begin();
i1++; // Chyba, it je konstanta
const vector<int> v2(10);
// Chyba, proc?
const vector<int>iterator i2 = v2.begin();
// Spravna verze
const vector<int>const_iterator i3 = v2.begin();
```

Na závěr ukázka

- Jak na dvourozměrný vektor