ZPRO 12. cvičení

Knihovna <initializer_list>

„Moderní alternativa výpustky“

- Data musí být stejného typu

- Data musí být v množinové závorce (jako při inicializaci pole)

```cpp
#include <initializer_list>
#include <iostream>

class MyClass
{
public:
    MyClass(std::initializer_list<int> values)
    {
        for (auto value : values)
        {
            std::cout << value << ' ';
        }
        std::cout << '\n';
    }
};

int main()
{
    MyClass obj = { 1, 2, 3, 4, 5 }; // Output: 1 2 3 4 5
    return 0;
}
```

Příklad

- Přidání konstruktoru s initializer_list do Chytrého Pole

Přetěžování operátorů metod

- Příklad indexovací opeátor[]

Kontejnerový for

- Jak C++ překládá kontejnerový for

```cpp
#include <vector>
using namespace std;
int main()
{
    vector<int> v = { 1,2,3,4,5 };

    for (auto a : v)
        cout << a << " ";
    cout << endl;

    for (auto it = v.begin(); it != v.end(); ++it)
    {
        auto a = *it;
        cout << a << " ";
    }
    cout << endl;
}
```

Cíl

- Upravit třídu ChytrePole tak, aby na ní šel použít kontejnerový for cyklus.

Iterátor – Návrhový vzor

- Datová struktura umožňující iterovat posloupnostmi dat.
- Kontejner obsahující Iterátor musí mít metody **begin()** a **end()**, který vrátí iterátor ukazující na první resp. (za) poslední prvek posloupnosti kontejneru
- Iterátor musí obsahovat následující 4 metody:
  - Konstruktor
  - Přetížený operátor ++, který posune iterátor na další hodnotu posloupnosti
  - Přetížený operátor * (dereferencování) pro přístup k datům, na které aktuálně iterátor ukazuje
  - Přetížený operátor !=, který porovná dva Iterátory
- Pak můžeme použít na kontejner kontejnerový for cyklus

Příklad

- Implementace iterátoru v Chytrém Poli

Šablony tříd

```
// MyClass.h

template <typename T>
class MyClass {
  public:
    MyClass(T value);
    void setValue(T value);
    T getValue() const;

  private:
    T value_;
};
```

Příklad

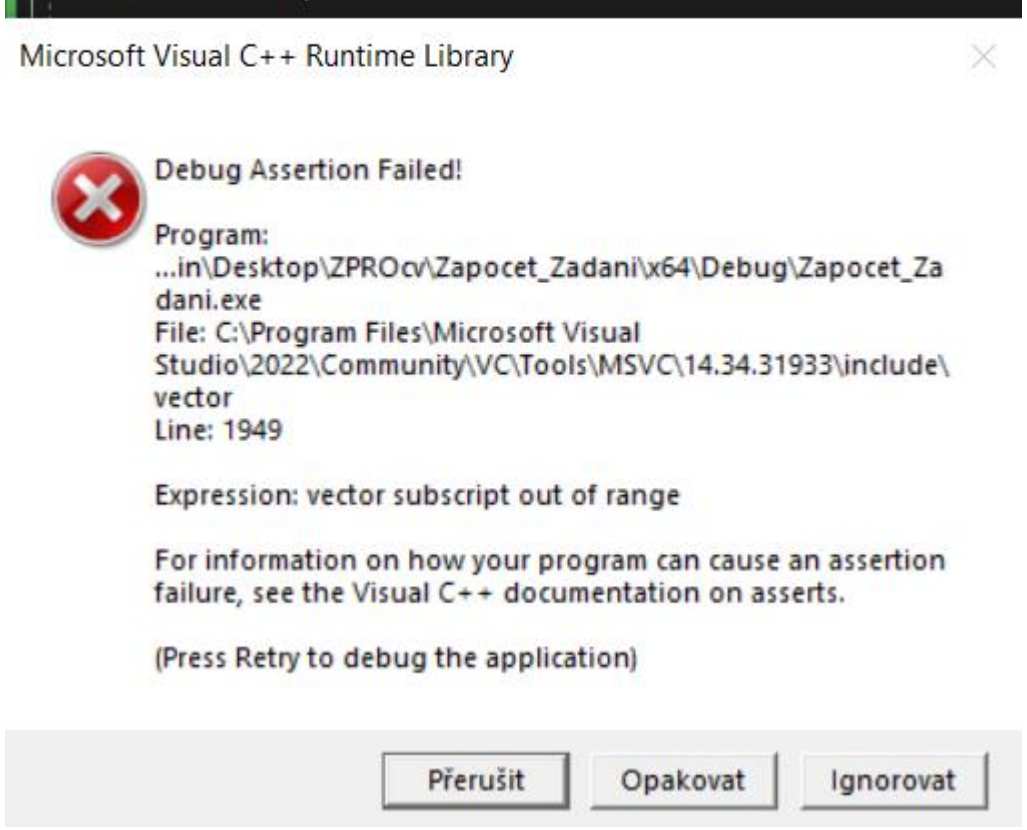
- Vylepšení třídy Chytré Pole šablonově

Aserce

- Makro assert
- `#include <assert.h>`

```
int main()
{
    vector<int> v = { 1,2,3,4,5 };

    cout << v[8];
```



Třídění šablonového kontejneru

- 1) Vlastní přetížení operátoru <
- 2) Použití ukazatele na porovnávací funkci
- 3) Použití komparátoru

```
struct Komparátor
{
    bool operator()(Třída a, Třída b)
    {
        return a.klíč < b.klíč;
    }
};
```

Dědění

- Příklad na dědění:

Vektor s metodou sort

```cpp
#include <vector>
#include <algorithm>

template <typename T>
class SortedVector : public std::vector<T>
{
public:
    // inherit constructors from std::vector
    using std::vector<T>::vector;

    // add a new method for sorting the vector
    void sort()
    {
        std::sort(this->begin(), this->end());
    }
};
```

Chytré ukazatele

- `#include <memory>`
- Objekt, který obsahuje ukazatel na jiný objekt, ale zároveň i spravuje životnost daného objektu.
- Tedy když ukazatel opustí oblast platnosti je uchovávaný objekt automaticky také zničen.
- Existuje několik různých typů inteligentních ukazatelů, každý s vlastním chováním a funkcemi.
- Některé z nejčastěji používaných typů jsou: `std::unique_ptr`, `std::shared_ptr` a `std::weak_ptr`.
- `std::unique_ptr` poskytuje exkluzivní vlastnictví ukazovaného objektu, což znamená, že je jediným objektem, který může uvolnit paměť.
- `std::shared_ptr` umožňuje více objektům sdílet vlastnictví ukazovaného objektu a paměť je uvolněna, když jsou zničeny všechny sdílené ukazatele, které na něj ukazují.
- `std::weak_ptr` je nevlastnící inteligentní ukazatel, který lze použít k pozorování objektu spravovaného bez ovlivnění `std::shared_ptr` jeho životnosti.