Technical University of Moldova
Inginerical department S.A.

Report

№: 7

OOP

Subject: Polymorfism and virtual functions

Author: Terman Emil FAF161
Prof: M. Kulev

Chisinau 2017

1 Objectives

- studierea necesităţii şabloanelor;
- studierea regulilor de definire şi utilizare a şabloanelor;
- studierea specializării şabloanelor;
- studierea potenţialelor probleme rezolvate cu ajutorul şabloanelor;

2 Main notions of theory and used methods

Templates are parametrized by one or more template parameters, of three kinds: type template parameters, non-type template parameters, and template template parameters.

When template arguments are provided, or, for function and class (since C++17) templates only, deduced, they are substituted for the template parameters to obtain a specialization of the template, that is, a specific type or a specific function lvalue. Specializations may also be provided explicitly: full specializations are allowed for both class and function templates, partial specializations are only allowed for class templates.

When a class template specialization is referenced in context that requires a complete object type, or when a function template specialization is referenced in context that requires a function definition to exist, the template is instantiated (the code for it is actually compiled), unless the template was already explicitly specialized or explicitly instantiated. Instantiation of a class template doesn't instantiate any of its member functions unless they are also used. At link time, identical instantiations generated by different translation units are merged.

The definition of a template must be visible at the point of implicit instantiation, which is why template libraries typically provide all template definitions in the headers (e.g. most boost libraries are header-only)

3 Task

1. Creaţi o funcţie şablon, care schimbă ordinea elementelor în felul următor: prima parte a listei se amestecă la urmă, dar a doua la început. De exemplu: 1 2 3 4 5 6 - 4 5 6 1 2 3. Funcţia trebuie să lucreze cu masive de lungimi diferite. Dacă numărul de elemente este impar, atunci elementul mijlociu nu trebuie de prelucrat.
2. Creaţi clasa parametrizată Stack. Clasa trebuie să conţină constructorii, destructorii, şi deasemenea funcţiile push, pop, empty, full şi operatorii de intrare/ieşire. Pentru alocarea memoriei să se utilizeze operatorul new.

4 Data analysis

4.1 Ex00

The code

```
template <typename T>
std :: vector<T> myShuffle(std :: vector<T> tab);
```

- `tab` is the target vector from which to make the shuffle;
- the function returns a new vector with the first half at the end;

4.2 Ex01

The code

The *Stack* class contains a 'stack' of *GenericNode*s remembered in the *last_* private field.

- `void Push(T newData);`

  Add a new element in the stack, increasing the *size_* value.

- `T Pop();`

  Remove the first element from the stack and return it.

5 Analysis of the results and conclusions

In this laboratory work, we studied Generic functions and classes. It's another, very critical feature of C++ that puts a big distance between C and C++, making easier to code.

– in comparison with C, the Generics feature introduced in C++ is very useful, because it would be necessary to use many *cast*s or *define*s to acomplish the same results in C (if possible);

– templates have a big drawback, it's possible to find the errors only at runtime. But it's natural, because the compiler has no way to know how much memory the *typename* needs;

– basically, templates are compiler friendly *define*s;

6 Anexes

CPP 1: main.cpp

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

template <typename T>
std::vector<T> myShuffle(std::vector<T> tab)
{
    std::vector<T> result;

    if (tab.size() <= 1)
        return std::vector<T>(tab);

    for (auto it = tab.begin() + tab.size() / 2; it < tab.end(); it++)
        result.push_back(*it);

    for (auto it = tab.begin(); it < tab.begin() + tab.size() / 2; it++)
        result.push_back(*it);
    return result;
}

template <typename T>
std::ostream & operator<<(std::ostream & o, std::vector<T> const & tab)
{
    for (T viktor: tab)
        o << viktor << "␣";
    return o;
}

int main()
{
    std::vector<int> tab[5] =
    {
        {1},
        {1, 2},
        {1, 2, 3},
        {1, 2, 3, 4},
        {1, 2, 3, 4, 5, 6}
    };

    for (int i = 0; i < 5; i++)
        std::cout << myShuffle<int>(tab[i]) << std::endl;

    std::vector<char> cTab[3] =
    {
        {'A'},
        {'A', 'B', 'C'},
        {'A', 'B', 'C', 'D'},
    };

    for (int i = 0; i < 3; i++)
        std::cout << myShuffle<char>(cTab[i]) << std::endl;
    return 0;
}
```

CPP 2: stack.h

```
#ifndef STACK_H
# define STACK_H

# include "generic_node.h"
# include <iostream>
# include <string>
# include <exception>
# include <ostream>

template <typename T>
class Stack
{
public:
    class IndexOutOfRange: public std::exception {
    public:
        virtual const char* what() const throw() {
            return "Index␣is␣out␣of␣range";
        }
    };

    class NoElements: public std::exception {
    public:
        virtual const char* what() const throw() {
            return "No␣elements␣in␣stack";
        }
    };

    int size() const { return size_; }
    bool isEmpty() const { return size() == 0; }

    Stack()
    {
        size_ = 0;
        last_ = nullptr;
    }

    ~Stack()
    {
        auto tmp = last_;

        while (last_ != nullptr)
        {
            tmp = last_;
            last_ = last_->prev();
            delete tmp;
        }
    }

    void Push(T newData)
    {
        auto new_node = new GenericNode<T>(newData);
        new_node->set_prev(last_);
        last_ = new_node;
        size_++;
    }

    T Pop()
    {
        if (last_ == nullptr)
            throw NoElements();

        auto target_node = last_;
        T result = target_node->data();
        last_ = last_->prev();

        delete(target_node);
        size_--;
        return result;
    }

    /*
    ** Operators
    */

    T& operator [] (int i)
    {
        auto node_iter = last_;

        while (i > 0)
            if (node_iter == nullptr)
                break;
            else
            {
                node_iter = node_iter->prev();
                i--;
            }

        if (node_iter == nullptr)
            throw IndexOutOfRange();

        return node_iter->data();
    }

    friend std::ostream& operator << (std::ostream& o, const Stack<T>& target)
    {
        auto node_iter = target.last_;

        o << "{";
        while (node_iter != nullptr)
        {
            o << node_iter->data();
            node_iter = node_iter->prev();

            if (node_iter != nullptr)
                o << ",␣";
        }
        o << "}";
        return o;
    }

    friend std::istream & operator >> (std::istream & is, Stack<T>& target)
    {
        int size;

        std::cout << "Elements␣to␣add:␣";
        is >> size;

        for (int i = 0; i < size; i++)
        {
            T new_data;

            std::cout << i << ")␣";
            is >> new_data;

            target.Push(new_data);
        }
        return is;
    }

private:
    int size_;
    GenericNode<T>* last_;
};

#endif
```

CPP 3: ‘generic˙node.h’

```
#ifndef _GENERIC_NODE_H_
# define _GENERIC_NODE_H_

# include <iostream>

template <typename T>
class GenericNode {
public:
    T& data() { return data_; }
    GenericNode* next() const { return next_; }
    GenericNode* prev() const { return prev_; }

    void set_data(T new_data) { data_ = new_data; }
    void set_next(GenericNode* new_next) { next_ = new_next; }
    void set_prev(GenericNode* new_prev) { prev_ = new_prev; }

    GenericNode(T data)
    {
        next_ = nullptr;
        prev_ = nullptr;
        data_ = data;
    }

private:
    GenericNode<T>* next_;
    GenericNode<T>* prev_;

    T data_;
};

#endif
```

CPP 4: main.cpp

```
#include "stack.h"
#include "generic_node.h"
#include <string>

int main()
{
    auto myStack = Stack<std::string>();
    myStack.Push("1");
    myStack.Push("2");
    myStack.Push("3");
    std::cout << "size(" << myStack.size() << ")␣" << myStack << std::endl;

    std::cout << myStack.Pop() << std::endl;
    std::cout << myStack << std::endl;

    std::cout << myStack.Pop() << std::endl;
    std::cout << myStack << std::endl;

    std::cin >> myStack;
    std::cout << myStack << std::endl;
    return 0;
}
```