# Московский авиационный институт (национальный исследовательский университет)

## Факультет информационных технологий и прикладной математики

## Кафедра вычислительной математики и программирования


**Лабораторная работа №4 по курсу «Объектно-ориентированное программирование»**


Студент: М. В. Спиридонов
Преподаватель:
Группа: М8О-206Б

Дата:
Оценка:
Подпись:

**Москва, 2017**

# Лабораторная работа №4

**Задача:** Необходимо спроектировать и запрограммировать на языке C++ шаблон класса-контейнера первого уровня, содержащий все три фигуры, согласно варианту задания Классы должны удовлетворять следующим правилам:

- Требования к классу фигуры аналогичны требованиям из лабораторной работы 1.
- Шаблон класса-контейнера должен содержать объекты, используя std::shared ptr<...>.
- Шаблон класса-контейнера должен иметь метод по добалению фигуры в контейнер.
- Шаблон класса-контейнера должен иметь методы по получению фигуры из контейнера.
- Шаблон класса-контейнера должен иметь метод по удалению фигуры из контейнера.
- Шаблон класса-контейнера должен иметь перегруженный оператор по выводу контейнера в поток ostream.
- Шаблон класса-контейнера должен иметь деструктор, удаляющий все элементы контейнера.
- Классы должны быть расположены в раздельных файлах: отдельно заголовки(.h), отдельно описание методов (.cpp).

**Фигуры.** Квадрат, треугольник, прямоугольник.
**Контейнер.** Массив.

# 1 Теория

Шаблоны (англ. template) — средство языка C++, предназначенное для кодирования обобщённых алгоритмов, без привязки к некоторым параметрам (например, типам данных, размерам буферов, значениям по умолчанию).

Шаблоны позволяют создавать параметризованные классы и функции. Параметром может быть любой тип или значение одного из допустимых типов.

Хотя шаблоны предоставляют краткую форму записи участка кода, на самом деле их использование не сокращает исполняемый код, так как для каждого набора параметров компилятор создаёт отдельный экземпляр функции или класса. Как следствие, исчезает возможность совместного использования скомпилированного кода в рамках разделяемых библиотек.

# 2 Листинг

```
1  //main.cpp
2  #include <iostream>
3  #include "TVector.h"
4  #include "Rectangle.h"
5  #include "Triangle.h"
6  #include "Square.h"
7
8  void Menu () {
9      std::cout << "Choose an operation:" << std::endl;
10     std::cout << "1) Add triangle" << std::endl;
11     std::cout << "2) Add rectangle" << std::endl;
12     std::cout << "3) Add square" << std::endl;
13     std::cout << "4) Get by Index" << std::endl;
14     std::cout << "5) Print vector" << std::endl;
15     std::cout << "0) Exit" << std::endl;
16 }
17
18 int main () {
19     int action = 0;
20     TVector<std::shared_ptr<Figure>> tVector(5);
21     std::shared_ptr <Figure> ptr;
22     do {
23         Menu();
24         std::cin >> action;
25         switch (action) {
26             case 1:
27                 ptr = std::make_shared <Triangle>(std::cin);
28                 tVector.FastPushBack(ptr);
29                 break;
30             case 2:
31                 ptr = std::make_shared <Rectangle>(std::cin);
32                 tVector.FastPushBack(ptr);
33                 break;
34             case 3:
35                 ptr = std::make_shared <TSquare>(std::cin);
36                 tVector.FastPushBack(ptr);
37                 break;
38             case 4:
39                 std::cin >> action;
40                 if (action <= tVector.Capacity() - 1)
41                     tVector[action]->Print();
42                 else {
43                     std::cout << "Index out of range";
44                 }
45                 break;
46             case 5:
47                 //std::cout << tVector << std::endl;
```

```
48                 for (int i = 0; i < tVector.Capacity(); ++i) {
49                     tVector[i]->Print();
50                 }
51                 break;
52             case 0:
53                 tVector.Clear();
54                 break;
55             default:
56                 std::cout << "Incorrect command" << std::endl;;
57                 break;
58         }
59     } while (action);
60     return 0;
61 }
62 //TVector.h
63 //
64 // Created by Maxim Spiridonov on 29.09.17.
65 //
66
67 #ifndef TVECTOR_H
68 #define TVECTOR_H
69
70 #include "iostream"
71 #include <memory>
72
73 template <class T>
74 class TVector {
75 public:
76
77     void FastPushBack (T &data);
78
79     void Clear ();
80
81     T &operator [] (int index);
82
83     int Size () const;
84
85     int Capacity () const;
86
87     TVector &operator = (const TVector <T> &inp);
88
89     TVector ();
90
91     TVector (const TVector <T> &inp);
92
93     friend std::ostream &operator << (std::ostream &os, const TVector <T> &tVector);
94
95     explicit TVector (int n);
96
```

```
97      ~TVector ();
98
99  private:
100     T *privateArray;
101     int privateArraySize;
102     int privateArrayOccupiedSize;
103     static int const SIZE_DIFFERENCE = 1;
104     static int const DEFAULT_INT_VALUE = 0;
105 };
106
107 #include "TVector.hpp"
108
109 #endif //TVECTOR_H
110 //TVector.hpp
111 #ifndef TVECTOR_HPP
112 #define TVECTOR_HPP
113
114 #include "TVector.h"
115
116 template <class T>
117 std::ostream &operator << (std::ostream &os, const TVector <T> &tVector) {
118     for (int i = 0; i < tVector.Capacity(); ++i) {
119         os << tVector[i] << std::endl;
120     }
121     return os;
122 }
123
124 template <class T>
125 void TVector <T>::FastPushBack (T &data) {
126     if (privateArraySize == privateArrayOccupiedSize) {
127         privateArraySize *= 2;
128         T *result = new T[privateArraySize];
129
130         for (int index = DEFAULT_INT_VALUE; index <= privateArrayOccupiedSize; index++)
              {
131             if (index != privateArrayOccupiedSize) {
132                 result[index] = privateArray[index];
133             } else {
134                 result[index] = data;
135                 break;
136             }
137         }
138         delete[] privateArray;
139         privateArray = result;
140         privateArrayOccupiedSize++;
141     } else {
142         privateArray[privateArrayOccupiedSize] = data;
143         privateArrayOccupiedSize++;
144     }
```

```
}

template <class T>
void TVector <T>::Clear () {
    delete[] privateArray;
    privateArraySize = DEFAULT_INT_VALUE;
    privateArrayOccupiedSize = DEFAULT_INT_VALUE;
    privateArray = new T[privateArraySize];
}

template <class T>
T &TVector <T>::operator [] (int index) {
    return this->privateArray[index];
}

template <class T>
int TVector <T>::Size () const {
    return privateArraySize;
}

template <class T>
int TVector <T>::Capacity () const {
    return this->privateArrayOccupiedSize;
}

template <class T>
TVector <T> &TVector <T>::operator = (const TVector <T> &inp) {
    return *this;
}

template <class T>
TVector <T>::TVector () {
    privateArraySize = DEFAULT_INT_VALUE;
    privateArrayOccupiedSize = DEFAULT_INT_VALUE;
    privateArray = new T[privateArraySize];
}

template <class T>
TVector <T>::TVector (const int n) {
    privateArraySize = n;
    privateArrayOccupiedSize = DEFAULT_INT_VALUE;
    privateArray = new T[privateArraySize];
}

template <class T>
TVector <T>::TVector (const TVector <T> &inp) {
    privateArraySize = inp.privateArraySize;
    privateArrayOccupiedSize = inp.privateArrayOccupiedSize;
    privateArray = inp.privateArray;
```

```
194 }
195
196 template <class T>
197 TVector <T>::~TVector () {
198     delete[] privateArray;
199     privateArraySize = DEFAULT_INT_VALUE;
200     privateArrayOccupiedSize = DEFAULT_INT_VALUE;
201 }
202
203 #endif // TVECTOR_HPP
```

# 3 Выводы

Шаблонное метапрограммирование в C++ страдает от множества ограничений, включая проблемы портируемости, отсутствие поддержки отладки или ввода/вывода в процессе инстанцирования шаблонов, длительное время компиляции, низкую читабельность кода, скудную диагностику ошибок и малопонятные сообщения об ошибках. Подсистема шаблонов C++ определяется как полный по Тьюрингу чистый функциональный язык программирования, но программисты в функциональном стиле считают это провокацией и не спешат признавать C++ успешным языком.