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: С. М. Бокоч
Преподаватель:
Группа: М8О-204Б

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, трапеция, 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  //TArray.h
2  #include "memory"
3  const size_t DEFAULT_CAPACITY = 10;
4  template <class T>
5  class TArray {
6  public:
7      TArray();
8      explicit TArray(const size_t &);
9      TArray(TArray&);
10     void Push_back(std::shared_ptr<T> &);
11     bool Delete(const size_t);
12     bool Empty();
13     size_t Size() const;
14     size_t Capacity() const;
15     std::shared_ptr<T>& operator[](size_t);
16     std::shared_ptr<T>& operator[](size_t) const;
17     //     Figure.
18     template <class A> friend std::ostream &operator<<(std::ostream &, const TArray<A>
           &);
19     ~TArray();
20 private:
21     std::shared_ptr<T> *_data;
22     size_t _capacity;
23     size_t _size;
24 };
25 #include "TArray.hpp"
26 #endif // PROG_TArray_H
27 //TArray.cpp
28 #include <memory>
29 #include "TArray.h"
30 template <class T>
31 TArray<T>::TArray() {
32     _data = new std::shared_ptr<T>[DEFAULT_CAPACITY];
33     _capacity = DEFAULT_CAPACITY;
34     _size = 0;
35 }
36 template <class T>
37 TArray<T>::TArray(const size_t &sizeArr) {
38     _data = new std::shared_ptr<T>[sizeArr];
39     for (int i = 0; i < sizeArr; i++) {
40         _data[i] = nullptr;
41     }
42     _capacity = sizeArr;
43     _size = 0;
44 }
45 template <class T>
46 TArray<T>::TArray(TArray<T>& orig) {
```

```
47      _data = new std::shared_ptr<T>[orig._capacity];
48      this->_size = orig._size;
49      this->_capacity = orig._capacity;
50      for (size_t index = 0; index < _size; index++) {
51          _data[index] = orig._data[index];
52      }
53  }
54  template <class T>
55  bool TArray<T>::Empty() {
56      return _size == 0;
57  }
58  template <class T>
59  void TArray<T>::Push_back(std::shared_ptr<T> &temp) {
60      if (_size == _capacity) {
61          _capacity *= 2;
62          std::shared_ptr<T> *copyArr = new std::shared_ptr<T>[_capacity];
63          for (size_t index = 0; index < _size; ++index) {
64              copyArr[index] = this->_data[index];
65          }
66          delete [] _data;
67          _data = copyArr;
68      }
69      this->_data[_size++] = temp;
70  }
71  template <class T>
72  bool TArray<T>::Delete(const size_t index) {
73      std::shared_ptr<T> *tCopy = new std::shared_ptr<T>[_capacity];
74      int j = 0;
75      bool flag = false;
76      for (int i = 0; i < _size; i++) {
77          if (i!=index) {
78              tCopy[j++] = _data[i];
79          }
80          else {
81              flag = true;
82          }
83      }
84      _size--;
85      delete [] _data;
86      _data = tCopy;
87      return flag;
88  }
89  template <class T>
90  std::shared_ptr<T>& TArray<T>::operator[](size_t index) const{
91      return _data[index];
92  }
93  template <class T>
94  std::shared_ptr<T>& TArray<T>::operator[](size_t index) {
95      return _data[index];
```

```
}
template <class T>
size_t TArray<T>::Size() const{
    return this->_size;
}
template <class T>
size_t TArray<T>::Capacity() const {
    return this->_capacity;
}
template <class T>
TArray<T>::~TArray() {
    delete[] _data;
}
#include "Figure.h"
template class TArray<Figure>;
template std::ostream& operator<<(std::ostream &os, const TArray<Figure> &objArr){
    for (size_t index = 0; index < objArr._size; ++index) {
        if (&objArr[index] != nullptr) {
            os << index << "\t";
            objArr[index]->Print();
        }
    }
    return os;
}
```

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.