Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №3 по курсу «Объектно-ориентированное программирование»

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

Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №3

Задача: Необходимо спроектировать и запрограммировать на языке C++ класс-контейнер первого уровня, содержащий все три фигуры, согласно варианту задания. Классы должны удовлетворять следующим правилам:

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

Фигуры. Треугольник, Квадрат, прямоугольник.

Контейнер. Массив.

1 Теория

Умный указатель – класс (обычно шаблонный), имитирующий интерфейс обычного указателя и добавляющий некую новую функциональность, например, проверку границ при доступе или очистку памяти. В данной лабораторной работе я использовал **shared_ptr** из библиотеки STL. **shared_ptr** – умный указатель с подсчитанными ссылками. Используется, когда необходимо присвоить один необработанный указатель нескольким владельцам,например, когда копия указателя возвращается из контейнера, но требуется сохранить оригинал. Необработанный указатель не будет удален до тех пор, пока все владельцы shared_ptr не выйдут из области или не откажутся от владения

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 list" << std::endl;
15     std::cout << "6) Delete by Index" << std::endl;
16     std::cout << "0) Exit" << std::endl;
17 }
18
19 int main () {
20     int32_t action = 0;
21     TVector tVector(1);
22     std::shared_ptr <Figure> ptr;
23     do {
24         menu();
25         std::cin >> action;
26         switch (action) {
27             case 1:
28                 ptr = std::make_shared <Triangle>(std::cin);
29                 tVector.FastPushBack(ptr);
30                 break;
31             case 2:
32                 ptr = std::make_shared <Rectangle>(std::cin);
33                 tVector.FastPushBack(ptr);
34                 break;
35             case 3:
36                 ptr = std::make_shared <TSquare>(std::cin);
37                 tVector.FastPushBack(ptr);
38                 break;
39             case 4:
40                 std::cin >> action;
41                 if (action <= tVector.Capacity() - 1)
42                     tVector[action]->Print();
43                 else {
44                     std::cout << "Index out of range" << std::endl;
45                 }
46                 break;
47             case 5:
```

```
48                 std::cout << tVector << std::endl;
49                 break;
50                 case 6:
51                 std::cin >> action;
52                 if (action <= tVector.Capacity() - 1)
53                 tVector.Remove(action);
54             else {
55                 std::cout << "Index out of range" << std::endl;
56             }
57                 break;
58             case 0:
59                 tVector.Clear();
60                 break;
61             default:
62                 std::cout << "Incorrect command" << std::endl;;
63                 break;
64         }
65     } while (action);
66     return 0;
67 }
68 //TVector.h
69 #ifndef TVECTOR_H
70 #define TVECTOR_H
71
72 #include <memory>
73 #include "Figure.h"
74 #include <ctime>
75 #include <iostream>
76 #include <cstdio>
77
78 class TVector {
79 public:
80
81     void FastPushBack (std::shared_ptr<Figure> &data);
82
83     void Remove(int index);
84
85     void Clear ();
86
87     std::shared_ptr<Figure>& operator [] (int index);
88
89     std::shared_ptr<Figure>& operator [] (int index) const;
90
91     int Size () const;
92
93     int Capacity () const;
94
95     TVector &operator = (const TVector &inp);
96
```

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

```
144     privateArrayOccupiedSize = DEFAULT_INT_VALUE;
145     privateArray = new std::shared_ptr <Figure>[privateArraySize];
146 }
147 
148 std::shared_ptr <Figure> &TVector::operator [] (int index) {
149     return this->privateArray[index];
150 }
151 
152 std::shared_ptr <Figure> &TVector::operator [] (int index) const {
153     return this->privateArray[index];
154 }
155 
156 int TVector::Size () const {
157     return privateArraySize;
158 }
159 
160 int TVector::Capacity () const {
161     return privateArrayOccupiedSize;
162 }
163 
164 TVector &TVector::operator = (const TVector &inp) {
165     return *this;
166 }
167 
168 TVector::TVector () {
169     privateArraySize = DEFAULT_INT_VALUE;
170     privateArrayOccupiedSize = DEFAULT_INT_VALUE;
171     privateArray = new std::shared_ptr <Figure>[privateArraySize];
172 }
173 
174 TVector::TVector (const int n) {
175     privateArraySize = n;
176     privateArrayOccupiedSize = DEFAULT_INT_VALUE;
177     privateArray = new std::shared_ptr <Figure>[privateArraySize];
178 }
179 
180 TVector::TVector (const TVector &inp) {
181     privateArraySize = inp.privateArraySize;
182     privateArrayOccupiedSize = inp.privateArrayOccupiedSize;
183     privateArray = inp.privateArray;
184 }
185 
186 void TVector::Remove (int index){
187     bool deleted = false;
188     for(int i = 0; i<privateArrayOccupiedSize;++i){
189         if(i==index && !deleted){
190             deleted =true;
191         }else if(i!=index && deleted){
192             privateArray[i-1] = privateArray[i];
```

```
193         }
194     }
195     --privateArrayOccupiedSize;
196 }
197 
198 TVector::~TVector () {
199     delete[] privateArray;
200     privateArraySize = DEFAULT_INT_VALUE;
201     privateArrayOccupiedSize = DEFAULT_INT_VALUE;
202 }
203 
204 std::ostream &operator << (std::ostream &os, const TVector &obj) {
205     for (int i = 0; i < obj.Capacity(); ++i) {
206         obj[i]->Print();
207     }
208     return os;
209 }
```

3 Выводы

Умные указатели, безусловно, являются необходимыми инструментами, если приходится иметь дело с динамическим выделением памяти. Хоть я и понимаю как они работают и для себя делал несколько реализаций, все равно почему с ними программа работает без ошибок – остается загадкой. Также важно знать чем указатели отличаются друг от друга, потому что бывают моменты, когда обойтись одним лишь shared_ptr нельзя.