Московский авиационный институт (национальный исследовательский университет)

Факультет информационных технологий и прикладной математики

Кафедра вычислительной математики и программирования

Лабораторная работа №9 по курсу «Объектно-ориентированное программирование»

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

Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №9

Задача:
Целью лабораторной работы является:

1. Знакомство с лямбда-выражениями

Используя структуры данных, разработанные для лабораторной работы №6 (контейнер 1-ого уровня и классы-фигуры) необходимо разработать:

- Контейнер второго уровня с использованием шаблонов.
- Реализовать с помощью лямбда-выражений набор команд, совершающих операции над контейнером 1-ого уровня: генерация фигур со случайными значениями параметров, печать контейнера на экран, удаление элементов со значением площади меньше определенного числа.
- В контейнер второго уровня поместить цепочку команд.
- Реализовать цикл, который проходит по всем командам в контейнере второго уровня и выполняет их, применяя к конейнеру первого уровня.

Фигуры. Квадрат, треугольник, прямоугольник.
Контейнер первого уровня. Массив.
Контейнер первого уровня. Массив.

1 Теория

Лямбда-выражение – это удобный способ определения анонимного объекта-функции непосредственно в месте его вызова или передачи в функцию в качестве аргумента. Обычно лямбда-выражения используются для инкапсуляции нескольких строк кода, передаваемых алгоритмам или асинхронным методам. В итоге, мы получаем крайне удобную конструкцию, которая позволяет сделать код более лаконичным и устойчивым к изменениям.

Непосредственное объявление лямбда-функции состоит из трех частей.

Первая часть (квадратные скобки) позволяет привязывать переменные, доступные в текущей области видимости.

Вторая часть (круглые скобки) указывает список принимаемых параметров лямбда-функции.

Третья часть (фигурные скобки) содержит тело лямбда-функции.

2 Листинг

```
1  //main.cpp
2  #include <iostream>
3  #include <memory>
4  #include <cstdlib>
5  #include <cstring>
6  #include <random>
7  #include "Triangle.h"
8  #include "Rectangle.h"
9  #include "Square.h"
10 #include "TVector.h"
11
12
13 int main(void)
14 {
15     TVector <Figure> vect1;
16     TVector<std::shared_ptr<std::function<void(void)> > > vect2;
17     std::mutex mtx;
18
19     std::function<void(void)> LambdaInsert = [&]() {
20         std::lock_guard<std::mutex> guard(mtx);
21
22         for (int i = 0; i < 10; ++ i) {
23             std::cout << "std::function<void(void)>: Insert" << std::endl;
24
25             switch(rand()%3) {
26                 case 0: {
27                     std::cout << "Inserted Trapezoid" << std::endl;
28                     vect1.PushFirst(std::shared_ptr<Trapezoid>(new Trapezoid(rand()%300,
                           rand()%300, rand()%300)));
29                     break;
30                 }
31
32                 case 1: {
33                     std::cout << "Inserted Rectangle" << std::endl;
34                     vect1.PushFirst(std::shared_ptr<Rectangle>(new Rectangle(rand()%300,
                           rand()%300)));
35                     break;
36                 }
37                 case 2: {
38                     std::cout << "Inserted Square" << std::endl;
39                     vect1.PushFirst(std::shared_ptr<Square>(new Square(rand()%300)));
40                     break;
41                 }
42             }
43         }
44     };
45
```

```
46
47          std::function<void(void)> LambdaRemove = [&]() {
48          std::lock_guard<std::mutex> guard(mtx);
49          std::cout << "std::function<void(void)>: Remove" << std::endl;
50          if (vect1.IsEmpty()) {
51              std::cout << "vect1 is empty" << std::endl;
52          } else {
53              double sqr = rand()%50000;
54              std::cout << "Remove figure with area " << sqr << std::endl;
55
56              for (size_t i = 0; i < 5; ++i) {
57                  auto iter = vect1.begin();
58                  for (size_t k = 0; k < vect1.GetLength(); ++k) {
59                      if (iter->GetSquare() < sqr) {
60                          vect1.Pop(k);
61                          break;
62                      }
63                      ++iter;
64                  }
65              }
66          }
67      };
68
69      std::function<void(void)> LambdaPrint = [&]() {
70          std::lock_guard<std::mutex> guard(mtx);
71
72          std::cout << "std::function<void(void)>: Print" << std::endl;
73          if(!vect1.IsEmpty()) {
74              std::cout << vect1 << std::endl;
75          } else {
76              std::cout << "vect1 is empty." << std::endl;
77          }
78      };
79
80      vect2.Push(std::shared_ptr<std::function<void(void)>>(&LambdaPrint, [](std::
            function<void(void)>*){}));
81      vect2.Push(std::shared_ptr<std::function<void(void)>>(&LambdaRemove, [](std::
            function<void(void)>*){}));
82      vect2.Push(std::shared_ptr<std::function<void(void)>>(&LambdaPrint, [](std::
            function<void(void)>*){}));
83      vect2.Push(std::shared_ptr<std::function<void(void)>>(&LambdaInsert, [](std::
            function<void(void)>*){}));
84
85      //
86      while (!vect2.IsEmpty()) {
87          std::shared_ptr<std::function<void(void)>> Lambda = std::move(vect2.Top());
88          std::future<void> ft = std::async(*Lambda);
89          ft.get();
90          vect2.Pop();
```

```
91      }
92
93      return 0;
94  }
```

3 Выводы

В результате выполнения данной лабораторной работы я познакомился с лямбда-выражениями. Я также смог реализовть контейнер, в который поместил цепочку команд. Цепочка команд представляет из себя не что иное, как лямбда-выражения. Я понял, что небходимо необходимо больше тренироваться в написании лямбда выражений, потому как это является неотъемлимой частью современного програмирования.