Лабораторная работа №9

Целью лабораторной работы является:

- Знакомство с лямбда-выражениями

1 Описание

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 Исходный код

Описание классов фигур и классов-контейнеров, определенных раннее, остается неизменным.

```
1
2  int main(void)
3  {
4      TList<Figure> list;
5      typedef std::function<void(void)> Command;
6      TNList<std::shared_ptr<Command>> nlist;
7      std::mutex mtx;
8
9      Command cmdInsert = [&]() {
10         std::lock_guard<std::mutex> guard(mtx);
11
12         uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count();
13
14         std::default_random_engine generator(seed);
15         std::uniform_int_distribution<int> distFigureType(1, 3);
```

```
16          std::uniform_int_distribution<int> distFigureParam(1, 10);
17          for (int i = 0; i < 5; ++ i) {
18              std::cout << "Command: Insert" << std::endl;
19
20              switch(distFigureType(generator)) {
21                  case 1: {
22                      std::cout << "Inserted Triangle" << std::endl;
23
24                      int side_a = distFigureParam(generator);
25                      int side_b = distFigureParam(generator);
26                      int side_c = distFigureParam(generator);
27
28                      std::shared_ptr<Figure> ptr = std::make_shared<Triangle>(Triangle(
                            side_a, side_b, side_c));
29                      list.PushFirst(ptr);
30                      break;
31                  }
32
33                  case 2: {
34                      std::cout << "Inserted Octagon" << std::endl;
35
36                      int side = distFigureParam(generator);
37                      std::shared_ptr<Figure> ptr = std::make_shared<Octagon>(Octagon(side
                            ));
38                      list.PushFirst(ptr);
39
40                      break;
41                  }
42
43                  case 3: {
44                      std::cout << "Inserted Foursquare" << std::endl;
45                      int side = distFigureParam(generator);
46                      std::shared_ptr<Figure> ptr = std::make_shared<Foursquare>(
                            Foursquare(side));
47
48                      list.PushFirst(ptr);
49
50                      break;
51                  }
52              }
53          }
54      };
55
56
57      Command cmdRemove = [&]() {
58          std::lock_guard<std::mutex> guard(mtx);
59
60          std::cout << "Command: Remove" << std::endl;
61
```

```
62          if (list.IsEmpty()) {
63              std::cout << "List is empty" << std::endl;
64          } else {
65              uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count()
                    ;
66
67              std::default_random_engine generator(seed);
68              std::uniform_int_distribution<int> distSquare(1, 150);
69              double sqr = distSquare(generator);
70              std::cout << "Lesser than " << sqr << std::endl;
71
72              for (int32_t i = 0; i < 5; ++i) {
73                  auto iter = list.begin();
74                  for (int32_t k = 0; k < list.GetLength(); ++k) {
75                      if (iter->Square() < sqr) {
76                          list.Pop(k);
77                          break;
78                      }
79                      ++iter;
80                  }
81              }
82          }
83      };
84
85      Command cmdPrint = [&]() {
86          std::lock_guard<std::mutex> guard(mtx);
87
88          std::cout << "Command: Print" << std::endl;
89          if(!list.IsEmpty()) {
90              std::cout << list << std::endl;
91          } else {
92              std::cout << "List is empty." << std::endl;
93          }
94      };
95      nlist.Push(std::shared_ptr<Command>(&cmdInsert, [](Command*){}));
96      nlist.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
97      nlist.Push(std::shared_ptr<Command>(&cmdRemove, [](Command*){}));
98      nlist.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
99
100
101     while (!nlist.IsEmpty()) {
102         std::shared_ptr<Command> cmd = nlist.Top();
103         std::future<void> ft = std::async(*cmd);
104         ft.get();
105         nlist.Pop();
106     }
107     retunr 0;
108 }
```

3 Консоль

```
denis@ubuntu:~/Desktop/OOP/oop_lab9$ ./run
Command: Insert
Inserted Triangle
Command: Insert
Inserted Foursquare
Command: Insert
Inserted Foursquare
Command: Insert
Inserted Triangle
Command: Insert
Inserted Octagon
Command: Print
idx: 0   Side = 6,type: Octagon

idx: 1   Side A = 3,side B = 2,side C = 5,type: Triangle

idx: 2   Side = 1,type: Foursquare

idx: 3   Side = 7,type: Foursquare

idx: 4   Side A = 6,side B = 3,side C = 5,type: Triangle


Command: Remove
Lesser than 70
Command: Print

denis@ubuntu:~/Desktop/OOP/oop_lab9$
```

4 Выводы

В данной лабораторной работе я познакомился с лямбда-выражениями. Теперь действия над контейнером первого уровня генерируются в виде команд, которые помещаются в контейнер второго уровня. Это очень удобно. Удобство использования лямбда-выражений - несомненно в сфере тестирования программ.