Лабораторная работа №9

Задача: 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.

Для создания потоков ипользовать механизмы:

- future
- packaged task/async

Для обеспечения потокобезопасности структур использовать механизмы:

- mutex
- lock quard

Фигуры: трапеция, ромб, пятиугольник.
Контейнер 1-ого уровня: связный список.
Контейнер 2-ого уровня: стек.

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.
В настоящее время, учитывая, что достигли практически потолка по тактовой частоте и дальше идет рост количества ядер, появился запрос на параллелизм. В результате снова в моде стал функциональный подход, так как он очень хорошо работает в условиях параллелизма и не требует явных синхронизаций. Поэтому сейчас усиленно думают, как задействовать растущее число ядер процессора и как обеспечить автоматическое распараллеливание. А в функциональном программировании практически основа всего – лямбда. Учитывая, что функциональные языки переживают второе рождение, было бы странным, если бы функциональный подход не добавляли во все популярные языки. C++ – язык, поддерживающий много парадигм, поэтому нет ничего странного в использовании лямбда-функций и лямбда-выражений в нем.

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      TStack<std::shared_ptr<Command>> stack;
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 < 10; ++ i) {
18              std::cout << "Command: Insert" << std::endl;
19
20              switch(distFigureType(generator)) {
21                  case 1: {
22                      std::cout << "Inserted trapeze" << std::endl;
23
24                      int32_t big_base = distFigureParam(generator);
25                      int32_t small_base = distFigureParam(generator);
26                      int32_t l_side = distFigureParam(generator);
27                      int32_t r_side = distFigureParam(generator);
28
29                      list.PushFirst(std::shared_ptr<Trapeze>(new Trapeze(small_base,
                            big_base, l_side, r_side)));
30
31                      break;
32                  }
33
34                  case 2: {
35                      std::cout << "Inserted rhomb" << std::endl;
36
37                      int32_t side = distFigureParam(generator);
38                      int32_t small_angle = distFigureParam(generator);
39
40                      list.PushFirst(std::shared_ptr<Rhomb>(new Rhomb(side, small_angle)))
                            ;
41
42                      break;
43                  }
44
45                  case 3: {
46                      std::cout << "Inserted pentagon" << std::endl;
47
48                      int32_t side = distFigureParam(generator);
49
50                      list.PushFirst(std::shared_ptr<Pentagon>(new Pentagon(side)));
51
52                      break;
53                  }
54              }
55          }
56      };
57
58
59      Command cmdRemove = [&]() {
```

```
60          std::lock_guard<std::mutex> guard(mtx);
61
62          std::cout << "Command: Remove" << std::endl;
63
64          if (list.IsEmpty()) {
65              std::cout << "List is empty" << std::endl;
66          } else {
67              uint32_t seed = std::chrono::system_clock::now().time_since_epoch().count()
                    ;
68
69              std::default_random_engine generator(seed);
70              std::uniform_int_distribution<int> distSquare(1, 150);
71              double sqr = distSquare(generator);
72              std::cout << "Lesser than " << sqr << std::endl;
73
74              for (int32_t i = 0; i < 10; ++i) {
75                  auto iter = list.begin();
76                  for (int32_t k = 0; k < list.GetLength(); ++k) {
77                      if (iter->Square() < sqr) {
78                          list.Pop(k);
79                          break;
80                      }
81                      ++iter;
82                  }
83              }
84          }
85      };
86
87      Command cmdPrint = [&]() {
88          std::lock_guard<std::mutex> guard(mtx);
89
90          std::cout << "Command: Print" << std::endl;
91          if(!list.IsEmpty()) {
92              std::cout << list << std::endl;
93          } else {
94              std::cout << "List is empty." << std::endl;
95          }
96      };
97
98      stack.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
99      stack.Push(std::shared_ptr<Command>(&cmdRemove, [](Command*){}));
100     stack.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
101     stack.Push(std::shared_ptr<Command>(&cmdInsert, [](Command*){}));
102
103     while (!stack.IsEmpty()) {
104         std::shared_ptr<Command> cmd = stack.Top();
105         std::future<void> ft = std::async(*cmd);
106         ft.get();
107         stack.Pop();
```

```
108     }
109
110     return 0;
111 }
```

3 Консоль

```
karma@karma:~/mai_study/OOP/lab9$ ./run
Command: Insert
Inserted trapeze
Command: Insert
Inserted trapeze
Command: Insert
Inserted pentagon
Command: Insert
Inserted rhomb
Command: Insert
Inserted rhomb
Command: Insert
Inserted pentagon
Command: Insert
Inserted trapeze
Command: Insert
Inserted trapeze
Command: Insert
Inserted rhomb
Command: Insert
Inserted trapeze
Command: Print
idx: 0   Smaller base = 6,bigger base = 9,left side = 10,right side = 10,square
= 16.8869,type: trapeze

idx: 1   Side = 3,smaller_angle = 2,square = 0.314095,type: rhomb

idx: 2   Smaller base = 7,bigger base = 9,left side = 5,right side = 8,square
= -1,type: trapeze

idx: 3   Smaller base = 6,bigger base = 9,left side = 4,right side = 3,square
= 10.1225,type: trapeze
```

```
idx: 4   Sides =  6,square = 61.9372,type: pentagon

idx: 5   Side = 1,smaller_angle = 1,square = 0.0174524,type: rhomb

idx: 6   Side = 2,smaller_angle = 2,square = 0.139598,type: rhomb

idx: 7   Sides =  6,square = 61.9372,type: pentagon

idx: 8   Smaller base = 2,bigger base = 10,left side = 3,right side = 4,square
= -1,type: trapeze

idx: 9   Smaller base = 1,bigger base = 8,left side = 8,right side = 5,square
= 9.2915,type: trapeze


Command: Remove
Lesser than 16
Command: Print
idx: 0   Smaller base = 6,bigger base = 9,left side = 10,right side = 10,square
= 16.8869,type: trapeze

idx: 1   Sides =  6,square = 61.9372,type: pentagon

idx: 2   Sides =  6,square = 61.9372,type: pentagon
```