

**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: Болотин А.Н.
Преподаватель: Поповкин А. В.
Группа: 08-207
Вариант: 19
Дата:
Оценка:
Подпись:

Москва, 2017

Лабораторная работа №9

1 Цель работы

- Знакомство с лямбда-выражениями

2 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 контейнеру первого уровня.

Для создания потоков использовать механизмы:

- future
- packaged task/async

Для обеспечения потокобезопасности структур использовать механизмы:

- mutex
- lock quard

Фигуры: Прямоугольник, ромб, трапеция

Контейнер 1: N-дерево **Контейнер 2:** Массив

3 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.

4 Исходный код

```
1 | int main()
2 | {
3 |     Tree<Rectangle> treeRect;
4 |     typedef std::function<void(void)> command;
5 |     Vector<command> vecCmd;
6 |
7 |
8 |
9 |     command cmdInsert = [&]() {
10 |         std::cout << "Creating random tree" << std::endl;
11 |         std::default_random_engine generator(time(NULL));
12 |         std::uniform_int_distribution<int> distribution(1, 20);
13 |         for (int i = 0; i < 10; i++) {
14 |             int side_a = distribution(generator);
15 |             int side_b = distribution(generator);
16 |             std::shared_ptr<Rectangle> ptr = std::make_shared<Rectangle>(side_a, side_b);
```

```

17     int key = distribution(generator);
18     treeRect.insert(ptr, 0, key);
19 }
20 };
21
22 command cmdPrint = [&]() {
23     std::cout << "Printing tree" << std::endl;
24     treeRect.print();
25 };
26
27 command removeIfEqualN = [&]() {
28     if (treeRect.empty()) {
29         std::cout << "Tree is empty" << std::endl;
30     }
31     else {
32         size_t n;
33         std::cout << "Enter N: ";
34         std::cin >> n;
35         treeRect.remove(n);
36     }
37 };
38
39
40 vecCmd.pushBack(std::shared_ptr<command>(&cmdInsert, [](command*){}));
41 vecCmd.pushBack(std::shared_ptr<command>(&cmdPrint, [](command*){}));
42 vecCmd.pushBack(std::shared_ptr<command>(&removeIfEqualN, [](command*){}));
43 vecCmd.pushBack(std::shared_ptr<command>(&cmdPrint, [](command*){}));
44
45 while (!vecCmd.empty()) {
46     std::shared_ptr<command> cmd = vecCmd.remove(0);
47     std::future<void> ft = std::async(*cmd);
48     ft.get();
49 }
50
51 system("pause");
52 return 0;
53 }

```

5 Выводы

В этой лабораторной работе были получены базовые понятия об реализации лямбда-выражений в C++. Они были помещены в ранее реализованный вектор, и выполнялись поочередно. Правильно реализованные лямбда-выражения упрощают код и дают прирост производительности.