Московский Авиационный Институт
(Национальный Исследовательский Университет)

Кафедра: 806 «Вычислительная математика и программирование»
Факультет: «Информационные технологии и прикладная математика»
Дисциплина: «Объектно-ориентированное программирование»

Лабораторная работа №9.

Группа: М8О – 104Б-16
Студент: Чекушкин Денис Игоревич
Преподаватель: Поповкин Александр Викторович
Вариант: №18

Москва
2017

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:

• Знакомство с лямбда-выражениями

ЗАДАНИЕ

Используя структуры данных, разработанные для лабораторной работы No6 (контейнер первого уровня и классы-фигуры) необходимо разработать:

• Контейнер второго уровня с использованием шаблонов.

• Реализовать с помощью лямбда-выражений набор команд, совершающих операции над

контенйром 1-го уровня:

o Генерация фигур со случайным значением параметров;

o Печать контейнера на экран;

o Удаление элементов со значением площади меньше определенного числа;

• В контенер второго уровня поместить цепочку команд.

• Реализовать цикл, который проходит по всем командам в контенере второго уровня и выполняет их, применяя к контейнеру первого уровня.

Для создания потоков использовать механизмы:

• future

• packaged_task/async

Для обеспечения потоко-безопасности структур данных использовать:

• mutex

• lock_guard

Теория:

Лямбда-выражение в программировании — специальный синтаксис для определения функциональных объектов, заимствованный из λ-исчисления. Применяется как правило для объявления анонимных функций по месту их использования, и обычно допускает замыкание на лексический контекст, в котором это выражение использовано. Используя лямбда-выражения, можно объявлять функции в любом месте кода.

Листинг

main.cpp

```
#include <functional>
#include <random>
#include <chrono>
```

```
#include <string>
#include <mutex>
#include <thread>
#include "btree.h"
#include "square.h"
#include "rectangle.h"
#include "trapezoid.h"
#include <iostream>

typedef std::function<void(void)> Command;
int main()
{
  Btree<Figure> b;
  Stack<Command> cmds;
  Stack<std::string> cmdsNames;
  Stack<Command> cmds3;
  Stack<Command> cmds2;
  Stack<std::string> cmdsNames2;
  std::mutex mtx;
  Command cmdInsert = [&]()
  {
        std::lock_guard<std::mutex> guard(mtx);
        unsigned int seed =
std::chrono::system_clock::now().time_since_epoch().count();
        std::default_random_engine generator(seed);
        std::uniform_int_distribution<int> distrFigureType(1, 3);
        std::uniform_int_distribution<int> distrFigureParam(1, 10);
        std::cout << "================" << std::endl;
        std::cout << "Command: insert" << std::endl;
        switch (distrFigureType(generator))
        {
              case 1:
              {
                    std::cout << "================" << std::endl;
                    std::cout << "Inserted: square" << std::endl;

                    double side = distrFigureParam(generator);
```

```
                        b.bstInsert(std::shared_ptr<Square>(new Square(side)));

                        break;
                }
                case 2:
                {
                        std::cout << "================" << std::endl;
                        std::cout << "Inserted: rectangle" << std::endl;
                        double sideA = distrFigureParam(generator);
                        double sideB = distrFigureParam(generator);
                        b.bstInsert(std::shared_ptr<Rectangle>(new Rectangle(sideA,
sideB)));
                        break;
                }
                case 3:
                {
                        std::cout << "================" << std::endl;
                        std::cout << "Inserted: trapezoid" << std::endl;
                        double sideA = distrFigureParam(generator);
                        double sideB = distrFigureParam(generator);
                        double height = distrFigureParam(generator);
                        b.bstInsert(std::shared_ptr<Trapezoid>(new Trapezoid(sideA,
sideB, height)));
                        break;
                }
        }
    };
    Command cmdErase = [&]()
    {
        std::lock_guard<std::mutex> guard(mtx);
        const double AREA = 24.0;
        std::cout << "================" << std::endl;
        std::cout << "Command: erase" << std::endl;

        if (b.size() == 0)
        {
                std::cout << "================" << std::endl;
                std::cout << "Stack is empty" << std::endl;
        }
```

```
		else
		{
			std::shared_ptr<Figure> first = b.front();
			while (true)
			{
				bool isRemoved = false;

				for (auto figure : b)
				{
					if (figure->area() < AREA)
					{
						std::cout << "================" << std::endl;
						std::cout << "Removed" << std::endl;
						figure->print();
						std::cout << "Area: " << figure->area() <<
std::endl;

						b.bstInsert(b.front());
						isRemoved = true;

						break;
					}
				}
				if (!isRemoved)
					break;
		}	}
	};
	Command cmdPrint = [&]()
	{
		std::lock_guard<std::mutex> guard(mtx);

		std::cout << "================" << std::endl;
		std::cout << "Command: print" << std::endl;

		for (auto figure : b)
		{
			figure->print();
			std::cout << "Area: " << figure->area() << std::endl;
```

```
	}
};

while (true)
{
	unsigned int action;

	std::cout << "================" << std::endl;
	std::cout << "Menu:" << std::endl;
	std::cout << "1) Add command" << std::endl;
	std::cout << "2) Erase command" << std::endl;
	std::cout << "3) Execute commands" << std::endl;
	std::cout << "4) Print commands" << std::endl;
	std::cout << "0) Quit" << std::endl;
	std::cin >> action;
	if (action == 0)
		break;
	if (action > 4)
	{
		std::cout << "Error: invalid action" << std::endl;
		continue;
	}
	switch (action)
	{
		case 1:
		{
			unsigned int commandType;
			std::cout << "================" << std::endl;
			std::cout << "1) Insert" << std::endl;
			std::cout << "2) Erase" << std::endl;
			std::cout << "3) Print" << std::endl;
			std::cout << "0) Quit" << std::endl;
			std::cin >> commandType;
			if (commandType > 0)
			{
				if (commandType > 3)
				{
					std::cout << "Error: invalid command type" <<
```

```
  std::endl;

                                        continue;
                                }
                                switch (commandType)
                                {
                                        case 1:
                                        {
cmds.Push(std::shared_ptr<Command>(&cmdInsert, [](Command*){}));
cmdsNames.Push(std::shared_ptr<std::string>(new std::string("Insert")));
                                                break;
                                        }
                                        case 2:
                                        {
cmds.Push(std::shared_ptr<Command>(&cmdErase, [](Command*){}));
cmdsNames.Push(std::shared_ptr<std::string>(new std::string("Erase")));
                                                break;
                                        }
                                        case 3:
                                        {
cmds.Push(std::shared_ptr<Command>(&cmdPrint, [](Command*){}));
cmdsNames.Push(std::shared_ptr<std::string>(new std::string("Print")));
                                                break;
                                        }
                                }
                        }
                        break;
                }
                case 2:
                {
                        std::cout << "================" << std::endl;
                        std::cout << "Front command erased" << std::endl;
                        cmds.Pop();
                        cmdsNames.Pop();
```

```
                        break;
                    }
                    case 3:
                        {
                        Stack<std::thread> ths;

                        for (auto cmd : cmds)
                            ths.Push(std::shared_ptr<std::thread>(new
std::thread(*cmd)));
                        for (auto th : ths)
                            th->join();
                     break;
                    }
                    case 4:
                    {
                        std::cout << "=================" << std::endl;

                        if (cmds.size() == 0)
                            {std::cout << "Commands list is empty" << std::endl;}
                        else {
                             cmdsNames2=cmdsNames;
                            while (cmdsNames2.size()>0) {
                                printf("=========\n");
                                    std::cout << *cmdsNames2.front() <<
std::endl;
                                    cmdsNames2.Pop();
                            }
                        }
                        break;
                    }
            }
        }
        return 0;
    }
```

Выводы: в данной лабораторной работе я получил навыки работы с лямбда-выражениями. Добавил с помощью лямбда-выражений команды, выполняющие операции с контейнером 1-го уровня.

https://github.com/israelcode/oop/tree/master/sem2/lab9