МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСОЙ ФЕДЕРАЦИИ МОСКОВСКИЙ АВЦИАЦИОННЫЙ ИНСТИТУТ

(НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЬЕЛЬСКИЙ УНИВЕРСТИТЕТ)

ЛАБОРАТОРНАЯ РАБОТА №8

по курсу объектно-ориентированное программирование I семестр, 2019/20 уч. год

Студент *Артемьев Дмитрий Иванович, группа М8О-206Б-18*

Преподаватель *Журавлёв Андрей Андреевич*

Условие

Задание №1: создать шаблонный класс коллекции стек, по заданию содержащей треугольники, с возможностью

1. push
2. pop
3. top
4. insert
5. erase
6. count_if
7. print

Реализовать аллокатор, который выделяет фиксированный размер памяти. Внутри аллокатор должен хранить указатель на используемый блок памяти и динамическую коллекцию указателей на свободные блоки. Динамическая коллекция в моём варианте - динамический массив.

Описание программы

Исходный код лежит в 8 файлах:

1. src/main.cpp: основная интерактивная программа, возможность работать со стеком
2. include/stack.hpp: описание и реализация класса стека
3. include/figure.hpp: описание и реализация класса обобщённой фигуры
4. include/triangle.hpp: описание и реализация класса треугольника
5. include/template.hpp: разные шаблонные вещи
6. include/point.hpp: описание и реализация класса точки
7. include/allocator.hpp: описание и реализация класса аллокатора
8. include/vector.hpp: описание и реализация класса вектора

Дневник отладки

Недочёты

Выводы

Я научился работать с потоками в языке C++, использовать Continuous Integration.

Исходный код

figure.hpp

```
#pragma once

#include <iostream>

#include <point.hpp>

struct Figure {
    virtual Point figureCenter() const = 0;
    virtual double figureArea() const = 0;
    virtual void printPoints(std::ostream& ) const = 0;

    virtual ~Figure() = default;
};

std::ostream& operator<< (std::ostream& out, Figure const& fig);
std::istream& operator>> (std::istream& in, Figure*& fig);
```

point.hpp

```
#pragma once

#include <iostream>

struct my_double {
    double v;
    my_double() {};
    my_double(double x) : v(x) {};
};

struct Point {
    my_double x, y;
    Point() {};
    Point(double x, double y) : x(my_double(x)), y(my_double(y)) {};
};

std::istream& operator>> (std::istream& , Point& );
std::ostream& operator<< (std::ostream& , Point const& );
Point operator+ (Point const& , Point const& );
Point operator- (Point const& , Point const& );
Point operator* (Point const& , double const& );
Point operator/ (Point const& , double const& );

my_double operator+ (my_double const& , my_double const& );
my_double operator- (my_double const& , my_double const& );
my_double operator* (my_double const& , my_double const& );
my_double operator/ (my_double const& , my_double const& );
bool operator<  (my_double const& a, my_double const& b);
bool operator>  (my_double const& a, my_double const& b);
bool operator== (my_double const& a, my_double const& b);

bool operator< (Point const& , Point const& );
bool operator> (Point const& , Point const& );
bool operator<= (Point const& , Point const& );
bool operator>= (Point const& , Point const& );
bool operator== (Point const& , Point const& );

my_double dist(const Point& , const Point& );
```

triangle.hpp

```cpp
#pragma once

#include <iostream>
#include <vector>

#include <figure.hpp>

class Triangle : public Figure{
public:
    Triangle(std::vector<Point> const& );

    Point figureCenter() const override;
    double figureArea() const override;
    void printPoints(std::ostream& ) const override;

    ~Triangle() = default;
protected:
    static const int NUM = 3;
    Point pts[NUM];
};
```

main.cpp

```cpp
#include <iostream>
#include <memory>
#include <vector>
#include <thread>
#include "processor.hpp"
#include "subscriber.hpp"
#include "figure.hpp"
#include "triangle.hpp"
#include "square.hpp"
#include "rectangle.hpp"

std::shared_ptr<Figure> getFigure(std::istream& is) {
    Figure* fig;
    is >> fig;
    return std::shared_ptr<Figure>(fig);
}

int main(int argc, char* argv[]) {
    if (argc != 2) {
        std::cout << "Usage: " << argv[0] << " <buffer size>\n";
        return 1;
    }
    const int buf_size = std::stoi(argv[1]);

    std::shared_ptr<std::vector<std::shared_ptr<Figure>>> buffer =
        std::make_shared<std::vector<std::shared_ptr<Figure>>>();
    buffer->reserve(buf_size);

    subscriber sub;
    sub.processors.push_back(std::make_shared<stream_processor>());
    sub.processors.push_back(std::make_shared<file_processor>());
    std::thread sub_thread(std::ref(sub));

    while(true) {
        std::unique_lock<std::mutex> guard(sub.mtx);
        std::string cmd;
        std::cin >> cmd;
        if (cmd == "add") {
            try {
                buffer->push_back(getFigure(std::cin));
            }
```

```
            catch (std::logic_error& e) {
                std::cout << e.what() << "\n";
                continue;
            }
            if (buffer->size() == buf_size) {
                sub.buffer = buffer;
                sub.cv.notify_all();
                sub.cv.wait(guard, [&]() { return sub.buffer == nullptr; });
                buffer->clear();
            }
        }
        else if (cmd == "exit"){
            break;
        }
        else {
            std::cout << "Wrong command!\n";
        }
    }
    sub.end = true;
    sub.cv.notify_all();
    sub_thread.join();

    return 0;
}
```