Московский Авиационный Институт
(Национальный исследовательский Университет)

Факультет: «Информационные технологии и прикладная математика»
Кафедра: 806 «Вычислительная математика и программирование»

Лабораторная работа
по курсу «ООП»

Тема:
Асинхронное программирование.

Студент:	Касимов М.М.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	6
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

figure.h

```
#ifndef OOP_EXERCISE_08_FIGURE_H
#define OOP_EXERCISE_08_FIGURE_H
#include <iostream>
#include "point.h"
#include <fstream>
struct figure {
   virtual point center() const = 0;
   virtual void print(std::ostream&) const = 0 ;
   virtual void printFile(std::ofstream&) const = 0 ;
   virtual double square() const = 0;
   virtual ~figure() = default;
};

#endif //OOP_EXERCISE_08_FIGURE_H
```

point.h

```
#ifndef OOP_EXERCISE_08_POINT_H
#define OOP_EXERCISE_08_POINT_H

#include <iostream>

struct point {
   double x, y;
   point (double  a,double b) { x = a, y = b;};
   point() = default;
};

std::istream& operator >> (std::istream& is,point& p );
std::ostream& operator << (std::ostream& os,const point& p);

#endif //OOP_EXERCISE_08_POINT_H
```

point.cpp

```
#include "point.h"

std::istream& operator >> (std::istream& is,point& p ) {
   return  is >> p.x >> p.y;
}

std::ostream& operator << (std::ostream& os,const point& p) {
   return os << p.x <<' '<< p.y;
}
```

pentagon.h

```
#ifndef OOP_EXERCISE_08_PENTAGON_H
#define OOP_EXERCISE_08_PENTAGON_H

#include "figure.h"

struct pentagon : figure{
```

```
private:
    point a1,a2,a3,a4,a5;
public:
    point center() const override ;
    void print(std::ostream&) const  override ;
    void printFile(std::ofstream&) const  override ;
    double square() const override ;
    pentagon() = default;
    pentagon(std::istream& is);
    pentagon(std::ifstream& is);
};

#endif //OOP_EXERCISE_08_PENTAGON_H
```

pentagon.cpp

```
#include "pentagon.h"

#include <cmath>
#include "point.h"

point pentagon::center() const {
    double x,y;
    x = (a1.x + a2.x + a3.x + a4.x + a5.x) / 5;
    y = (a1.y + a2.y + a3.y + a4.y + a5.y) / 5;
    point p(x,y);
    return p;
}

void pentagon::print(std::ostream& os) const {
    os << "pentagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 << '\n';
}

void pentagon::printFile(std::ofstream& of) const {
    of << "pentagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 << '\n';
}

double pentagon::square() const{
    //метод Гаусса(алгоритм шнурования)
    return (-0.5) * ((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a1.y) - ( a1.y*a2.x +
a2.y*a3.x + a3.y*a4.x + a4.y*a5.x + a5.y*a1.x ));
}

pentagon::pentagon(std::istream& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5;
}

pentagon::pentagon(std::ifstream& is) {
    is >> a1 >> a2 >> a3 >> a4 >> a5;
}
```

hexagon.h

```
#ifndef OOP_EXERCISE_08_HEXAGON_H
```

```
#define OOP_EXERCISE_08_HEXAGON_H

#include "figure.h"

struct hexagon : figure{
private:
   point a1,a2,a3,a4,a5,a6;
public:
   point center() const override ;
   void print(std::ostream&) const  override ;
   void printFile(std::ofstream&) const  override ;
   double square() const override ;
   hexagon() = default;
   hexagon(std::istream& is);
   hexagon(std::ifstream& is);
};
```

hexagon.cpp

```
#include "hexagon.h"

point hexagon::center() const {
   double x,y;
   x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x) / 6;
   y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y) / 6;
   point p(x,y);
   return p;
}
void hexagon::print(std::ostream& os) const {
   os << "hexagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 << '\n' << a6 <<
"\n";
}

void hexagon::printFile(std::ofstream &of) const {
   of << "hexagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 << '\n' << a6 <<
"\n";
}

double hexagon::square() const {
   return (-0.5) * ((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a6.y + a6.x*a1.y) - (
a1.y*a2.x + a2.y*a3.x + a3.y*a4.x + a4.y*a5.x + a5.y*a6.x + a6.y*a1.x ));
}

hexagon::hexagon(std::istream& is) {
   is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;
}

hexagon::hexagon(std::ifstream& is) {
   is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6;
}

#endif //OOP_EXERCISE_08_HEXAGON_H
```

octagon.h

```
#ifndef OOP_EXERCISE_08_OCTAGON_H
#define OOP_EXERCISE_08_OCTAGON_H

#include "figure.h"

struct octagon : figure{
private:
   point a1,a2,a3,a4,a5,a6,a7,a8;
public:
   point center() const override ;
   void print(std::ostream&) const  override ;
   void printFile(std::ofstream&) const  override ;
   double square() const override ;
   octagon() = default;
   octagon(std::istream& is);
   octagon(std::ifstream& is);
};


#endif //OOP_EXERCISE_08_OCTAGON_H
```

octagon.cpp

```
#include "octagon.h"

point octagon::center() const {
   double x,y;
   x = (a1.x + a2.x + a3.x + a4.x + a5.x + a6.x + a7.x + a8.x) / 8;
   y = (a1.y + a2.y + a3.y + a4.y + a5.y + a6.y + a7.y + a8.x) / 8;
   point p(x,y);
   return p;
}
void octagon::print(std::ostream& os) const {
   os << "octagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 << '\n' << a6 <<
"\n" << a7 << '\n' << a8 << '\n';
}

void octagon::printFile(std::ofstream& of) const {
   of << "octagon\n"<< a1 << '\n' << a2 << '\n' << a3 << '\n' << a4 << '\n' << a5 << '\n' << a6 <<
"\n" << a7 << '\n' << a8 << '\n';
}

double octagon::square() const {
   return (-0.5) * ((a1.x*a2.y + a2.x*a3.y + a3.x*a4.y + a4.x*a5.y + a5.x*a6.y + a6.x*a7.y +
a7.x*a8.y + a8.x*a1.y) - ( a1.y*a2.x + a2.y*a3.x + a3.y*a4.x + a4.y*a5.x + a5.y*a6.x + a6.y*a7.x +
a7.y*a8.x + a8.y*a1.x));
}

octagon::octagon(std::istream& is) {
   is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;
}
```

```
octagon::octagon(std::ifstream& is) {
   is >> a1 >> a2 >> a3 >> a4 >> a5 >> a6 >> a7 >> a8;
}
```

factory.h

```
#ifndef OOP_EXERCISE_08_FACTORY_H
#define OOP_EXERCISE_08_FACTORY_H

#include <memory>
#include <iostream>
#include <fstream>
#include "hexagon.h"
#include "octagon.h"
#include "pentagon.h"
#include <string>

struct factory {
   std::shared_ptr<figure> FigureCreate(std::istream& is);
   std::shared_ptr<figure> FigureCreateFile(std::ifstream& is);
};
#endif //OOP_EXERCISE_08_FACTORY_H
```

factory.cpp

```
#include "factory.h"

std::shared_ptr<figure> factory::FigureCreate(std::istream &is) {
   std::string name;
   is >> name;
   if ( name == "pentagon" ) {
      return std::shared_ptr<figure> ( new pentagon(is));
   } else if ( name == "hexagon") {
      return std::shared_ptr<figure> ( new hexagon(is));
   } else if ( name == "octagon") {
      return std::shared_ptr<figure> ( new octagon(is));
   } else {
      throw std::logic_error("There is no such figure\n");
   }
}

std::shared_ptr<figure> factory::FigureCreateFile(std::ifstream &is) {
   std::string name;
   is >> name;
   if ( name == "pentagon" ) {
      return std::shared_ptr<figure> ( new pentagon(is));
   } else if ( name == "hexagon") {
      return std::shared_ptr<figure> ( new hexagon(is));
   } else if ( name == "octagon") {
      return std::shared_ptr<figure> ( new octagon(is));
   } else {
      throw std::logic_error("There is no such figure\n");
   }
}
```

subscriber.h

```
#ifndef OOP_EXERCISE_08_SUBSCRIBER_H
#define OOP_EXERCISE_08_SUBSCRIBER_H

#include <iostream>
#include <condition_variable>
#include <thread>
#include <vector>
#include <mutex>
#include "factory.h"
#include "figure.h"
#include "processor.h"

struct subscriber {
    void operator()();
    std::vector<std::shared_ptr<processor>> processors;
    std::shared_ptr<std::vector<std::shared_ptr<figure>>> buffer;
    std::mutex mtx;
    std::condition_variable cv;
    bool end = false;

};

#endif //OOP_EXERCISE_08_SUBSCRIBER_H
```

subscriber.cpp

```
#include "subscriber.h"

void subscriber::operator()() {
    while(true) {
        std::unique_lock<std::mutex> lock(mtx);
        //std::cout << "thread lock\n";
        cv.wait(lock,[&]{ return (buffer != nullptr || end);});
        //std::cout << "thread unlock\n";
        if (end) {
            break;
        }
        for (const auto& processor_elem: processors) {
            processor_elem->process(buffer);
        }
        buffer = nullptr;
        cv.notify_all();
    }
}
```

processor.h

```
#ifndef OOP_EXERCISE_08_PROCESSOR_H
#define OOP_EXERCISE_08_PROCESSOR_H
#include <iostream>
#include <condition_variable>
#include <thread>
#include <vector>
#include <mutex>
#include "factory.h"
#include "figure.h"

struct processor {
    virtual void process(std::shared_ptr<std::vector<std::shared_ptr<figure>>> buffer) = 0;

};

struct stream_processor : processor {
    void process(std::shared_ptr<std::vector<std::shared_ptr<figure>>> buffer) override;
};

struct file_processor : processor {
    void process(std::shared_ptr<std::vector<std::shared_ptr<figure>>> buffer) override;
private:
    uint64_t counter = 0;
};
#endif //OOP_EXERCISE_08_PROCESSOR_H
```

processor.cpp

```
#include "processor.h"

void stream_processor::process(std::shared_ptr<std::vector<std::shared_ptr<figure>>> buffer) {
    for (const auto& figure : *buffer) {
        figure->print(std::cout);
    }
}

void file_processor::process(std::shared_ptr<std::vector<std::shared_ptr<figure>>> buffer) {
    std::ofstream fout;
    fout.open(std::to_string(counter) + ".txt");
    ++counter;
    if (!fout.is_open()) {
        std::cout << "File not opened\n";
        return;
    }
    for (const auto& figure : *buffer) {
        figure->printFile(fout);
    }

}
```

2. Ссылка на репозиторий на GitHub.

https://github.com/magomed2000kasimov/oop_exercise_08

3. Набор тестов.

test_01.test:

```
add pentagon 1 1 1 1 1 1 1 1 1 1
add pentagon 2 2 2 2 2 2 2 2 2 2
add pentagon 3 3 3 3 3 3 3 3 3 3
add pentagon 5 5 5 5 5 5 5 5 5 5
add hexagon 6 6 6 6 6 6 6 6 6 6 6 6
add octagon 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8 8
exit
```

test_02.test:

```
remove
add triangle
end
exit
```

4. Результаты выполнения тестов.

test_01.result:

```
add - adding a new shape
exit - the end of the program
begin
begin
begin
pentagon
1 1
1 1
1 1
1 1
1 1
pentagon
2 2
2 2
2 2
2 2
2 2
pentagon
3 3
3 3
3 3
3 3
3 3
```

begin
begin
begin
pentagon
5 5
5 5
5 5
5 5
5 5
hexagon
6 6
6 6
6 6
6 6
6 6
6 6
octagon
8 8
8 8
8 8
8 8
8 8
8 8
8 8
8 8
begin
test_02.result:
add - adding a new shape
exit - the end of the program
begin
unknown command
begin
There is no such figure

begin
unknown command
begin

5. Объяснение результатов работы программы.

1) Метод center() const возвращает точку с x –деление суммы иксов всех точек данной фигуры на их количество, y – аналогично x.

2) Метод print(std::ostream&) const печатает координаты всех точек данной фигуры.
3) Метод square() const вычисляет площадь данной фигуры по методу Гаусса (формула землемера, метод шунтирования) и возвращает это значение.
4) Удаление в main.cpp фигуры из вектора по индексу происходит:
- удаляается фигура с помощью delete.
- элементы вектора сдвигаются влево циклом for , чтобы закрыть индекс удаленного элемента.
- используется метод вектора pop_back();.
5) работают 2 потока, один считывает и сохраняет в буфер фигуры, а другой печатает их в файл и на консоль.

6. Вывод.

Выполняя данную лабораторную работу, я обрел базовые навыки многопоточного программирования, научился использовать мьютексы и условные переменные. Понял, как нужно работать с потоками и какие могут при этом возникнуть трудности.

7. Литература.

1)лекции по ООП МАИ.
2) https://habr.com/ru/post/182626/