Московский Авиационный Институт
(Национальный исследовательский Университет)

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

Лабораторная работа
по курсу «ООП»

Тема:
Наследование, полиморфизм.

Студент:	Козлов А.Д.
Группа:	М80-206Б-18
Преподаватель:	Журавлев А.А.
Вариант:	7
Оценка:	
Дата:	

Москва
2019

1. Код программы на языке C++:

Figure.hpp:

```
#ifndef FIGURE_HPP
#define FIGURE_HPP

#include <iostream>

struct Point {
   double X;
   double Y;
};
std::istream& operator >>(std::istream&, Point&);
std::ostream& operator <<(std::ostream&, Point&);

class Figure {
public:
   Figure() {};
   ~Figure() {};

   virtual int GetSize() const = 0;

   virtual void ReadFigure(std::istream&) = 0;
   virtual void WriteFigure(std::ostream&) const = 0;

   virtual double FigureArea() = 0;
   virtual Point GeomCenter() = 0;
protected:
   static double PolygonArea(const Point*, const int);
   static Point PolygonCenter(const Point*, const int);
};

std::istream& operator >>(std::istream&, Figure&);
std::ostream& operator <<(std::ostream&, Figure&);

#endif
```

Figure.cpp:

```
#include "../include/Figure.hpp"

std::istream& operator >>(std::istream& is, Point& p) {
   is >> p.X >> p.Y;
   return is;
}
std::ostream& operator <<(std::ostream& os, Point& p) {
   os << "(" << p.X << ", " << p.Y << ")";
```

```
    return os;
}

double Figure::PolygonArea(const Point* Peaks, const int Size) {
    //std::cout << "Size = " << Size << "\n";
    double A = 0.0;
    if(Size < 3) {
        return A;
    }
    for(int i = 0; i < Size - 1; ++i) {
        A = A + (Peaks[i].X * Peaks[i + 1].Y);
    }
    A = A + (Peaks[Size-1].X * Peaks[0].Y);
    for(int i = 0; i < Size - 1; ++i) {
        A = A - (Peaks[i+1].X * Peaks[i].Y);
    }
    A = A - (Peaks[0].X * Peaks[Size-1].Y);
    return A >= 0 ? (A * 0.5) : (-A * 0.5);
}
Point Figure::PolygonCenter(const Point* Peaks, const int Size) {
    Point res;
    double x = 0;
    double y = 0;
    for(int i = 0; i < Size; ++i) {
        x = x + Peaks[i].X;
        y = y + Peaks[i].Y;
    }
    res.X = x / Size;
    res.Y = y / Size;
    return res;
}

std::istream& operator >>(std::istream& is, Figure& f) {
    f.ReadFigure(is);
    return is;
}
std::ostream& operator <<(std::ostream& os, Figure& f) {
    f.WriteFigure(os);
    return os;
}
```

Hexagon.hpp:

```
#ifndef HEXAGON_HPP
#define HEXAGON_HPP

#include <iostream>
#include "Figure.hpp"

class Hexagon : public Figure {
private:
   Point Peaks[6];
   const int Size = 6;
public:
   Hexagon() {};
   ~Hexagon() {};

   int GetSize() const override;

   void ReadFigure(std::istream&) override;
   void WriteFigure(std::ostream&) const override;

   double FigureArea() override;
   virtual Point GeomCenter() override;
};

std::istream& operator >>(std::istream&, Hexagon&);
std::ostream& operator <<(std::ostream&, Hexagon&);

#endif
```

Hexagon.cpp:

```
#include "../include/Hexagon.hpp"

int Hexagon::GetSize() const {
   return Size;
}

void Hexagon::ReadFigure(std::istream& is) {
   for(int i = 0; i < Size; ++i) {
      is >> Peaks[i].X;
      is >> Peaks[i].Y;
   }
}
void Hexagon::WriteFigure(std::ostream& os) const {
   for(int i = 0; i < Size; ++i) {
      os << "(" << Peaks[i].X << ", " << Peaks[i].Y << ")";
```

```
    }
}

double Hexagon::FigureArea() {
    return PolygonArea(Peaks, Size);
}
Point Hexagon::GeomCenter() {
    return PolygonCenter(Peaks, Size);
}

std::istream& operator >>(std::istream& is, Hexagon& x) {
    x.ReadFigure(is);
    return is;
}
std::ostream& operator <<(std::ostream& os, Hexagon& x) {
    x.WriteFigure(os);
    return os;
}
```

Octagon.hpp:

```
#ifndef OCTAGON_HPP
#define OCTAGON_HPP

#include <iostream>
#include "Figure.hpp"

class Octagon : public Figure {
private:
   Point Peaks[8];
   const int Size = 8;
public:
   Octagon() {};
   ~Octagon() {};

   int GetSize() const override;

   void ReadFigure(std::istream&) override;
   void WriteFigure(std::ostream&) const override;

   double FigureArea() override;
   virtual Point GeomCenter() override;
};

std::istream& operator >>(std::istream&, Octagon&);
std::ostream& operator <<(std::ostream&, Octagon&);

#endif
```

Octagon.cpp:

```
#include "../include/Octagon.hpp"

int Octagon::GetSize() const {
   return Size;
}

void Octagon::ReadFigure(std::istream& is) {
   for(int i = 0; i < Size; ++i) {
      is >> Peaks[i].X;
      is >> Peaks[i].Y;
   }
}
void Octagon::WriteFigure(std::ostream& os) const {
   for(int i = 0; i < Size; ++i) {
      os << "(" << Peaks[i].X << ", " << Peaks[i].Y << ")";
```

```
    }
}

double Octagon::FigureArea() {
    return PolygonArea(Peaks, Size);
}
Point Octagon::GeomCenter() {
    return PolygonCenter(Peaks, Size);
}

std::istream& operator >>(std::istream& is, Octagon& oc) {
    oc.ReadFigure(is);
    return is;
}
std::ostream& operator <<(std::ostream& os, Octagon& oc) {
    oc.WriteFigure(os);
    return os;
}
```

Triangle.hpp:

```
#ifndef TRIANGLE_HPP
#define TRIANGLE_HPP

#include <iostream>
#include "Figure.hpp"

class Triangle : public Figure {
private:
   Point Peaks[3];
   const int Size = 3;
public:
   Triangle() {};
   ~Triangle() {};

   int GetSize() const override;

   void ReadFigure(std::istream&) override;
   void WriteFigure(std::ostream&) const override;

   double FigureArea() override;
   virtual Point GeomCenter() override;
};

std::istream& operator >>(std::istream&, Triangle&);
std::ostream& operator <<(std::ostream&, Triangle&);

#endif
```

Triangle.cpp:

```
#include "../include/Triangle.hpp"

int Triangle::GetSize() const {
   return Size;
}

void Triangle::ReadFigure(std::istream& is) {
   for(int i = 0; i < Size; ++i) {
      is >> Peaks[i].X;
      is >> Peaks[i].Y;
   }
}
void Triangle::WriteFigure(std::ostream& os) const {
   for(int i = 0; i < Size; ++i) {
      os << "(" << Peaks[i].X << ", " << Peaks[i].Y << ")";
```

```
    }
}

double Triangle::FigureArea() {
    return PolygonArea(Peaks, Size);
}
Point Triangle::GeomCenter() {
    return PolygonCenter(Peaks, Size);
}

std::istream& operator >>(std::istream& is, Triangle& t) {
    t.ReadFigure(is);
    return is;
}
std::ostream& operator <<(std::ostream& os, Triangle& t) {
    t.WriteFigure(os);
    return os;
}
```

main.hpp:

```
#ifndef MAIN_HPP
#define MAIN_HPP

#include <iostream>
#include <fstream>
#include <vector>

#include "Figure.hpp"
#include "Hexagon.hpp"
#include "Octagon.hpp"
#include "Triangle.hpp"

#endif
```

main.cpp:

```
#include "../include/main.hpp"

int main(int argc, char* argv[]) {
    Hexagon hex;
    Octagon oct;
    Triangle tri;
    Point p;

    Figure* fig;
    std::vector<Figure*> v;
```

```
int key = 0;
char buff[50];

while(key < 5 && key >= 0) {
   std::cout << "============================\n";
   std::cout << "1)Add\n2)Functions\n3)Remove\n4)Test\n5)Exit\n";
   std::cout << "----------------------------\n";
   std::cin >> key;
   if(key == 1) {
      std::cout << "----------------------------\n";
      std::cout << "1)Hexagon\n2)Octagon\n3)Triangle\n";
      std::cout << "----------------------------\n";
      std::cin >> key;
      if(key == 1) {
         fig = new Hexagon;
      } else if(key == 2) {
         fig = new Octagon;
      } else if(key == 3) {
         fig = new Triangle;
      } else {
         continue;
      }
      std::cin >> *fig;
      v.push_back(fig);
   } else if(key == 2) {
      if(v.size() == 0) {
         std::cout << "----------------------------\n";
         std::cout << "Empty vector.\n";
      }
      for(int i = 0; i < v.size(); ++i) {
         p = v[i]->GeomCenter();
         std::cout << "----------------------------\n";
         std::cout << "Фигура: " << *(v[i]) << "\n";
         std::cout << "Геометрический центр: " << p << "\n";
         std::cout << "Площадь: " << v[i]->FigureArea() << "\n";
      }
   } else if(key == 3) {
      std::cout << "----------------------------\n";
      std::cout << "Index: ";
      std::cin >> key;
      if(key >= 0 && key < v.size()) {
         delete v[key];
         v.erase(v.begin() + key);
      }
   } else if(key == 4) {
```

```
            std::cin >> buff;
            std::ifstream fin(buff);
            if(!fin.is_open()) {
                continue;
            }
            fin >>  hex;
            fin >> oct;
            fin >> tri;
            std::cout << "----------------------------\n";
            std::cout << "Фигура: " << hex << "\n";
            p = hex.GeomCenter();
            std::cout << "Геометрический центр: " << p << "\n";
            std::cout << "Площать: " << hex.FigureArea() << "\n";
            std::cout << "----------------------------\n";
            std::cout << "Фигура: " << oct << "\n";
            p = oct.GeomCenter();
            std::cout << "Геометрический центр: " << p << "\n";
            std::cout << "Площать: " << oct.FigureArea() << "\n";
            std::cout << "----------------------------\n";
            std::cout << "Фигура: " << tri << "\n";
            p = tri.GeomCenter();
            std::cout << "Геометрический центр: " << p << "\n";
            std::cout << "Площать: " << tri.FigureArea() << "\n";

            fin.close();
        } else if(key == 5) {
            break;
        }
    }
    for(int i = 0; i < v.size(); ++i) {
        delete v[i];
    }
    return 0;
}
```

main.cpp:

```
#include <iostream>
#include <fstream>
#include "bigString.hpp"

void separator()
{
    std::cout <<"==============";
    for(int i = 0; i < 128; ++i)
    {
```

```
        std::cout << "=";
    }
    std::cout << std::endl;
}

unsigned long long get_numfile(std::ifstream& fin)
{
    unsigned long long num = 0;
    char ch;
    while(fin.get(ch))
    {
        if (ch != ' ' && ch != '\n' && ch != EOF)
        {
            num = num * 10 + (ch - '0');
        }
        else
        {
            break;
        }
    }
    return num;
}

int main(int argc, char* argv[])
{
    unsigned long long n1;
    unsigned long long n2;

    if (argc < 2)
    {
        return 1;
    }

    std::ifstream fin(argv[1], std::ios_base::in);

    if (!fin.is_open())
    {
        std::cout << "Нельзя отрыть файл!\n";
        return 2;
    }

    n1 = get_numfile(fin);
    n2 = get_numfile(fin);
    BigString str1(n1, n2);
```

```
n1 = get_numfile(fin);
n2 = get_numfile(fin);
BigString str2(n1, n2);

separator();
std::cout << "str1          ";
str1.print_bits();
std::cout << "str2          ";
str2.print_bits();
std::cout << "str1 and str2 ";
str1.BS_and(str2).print_bits();
separator();
std::cout << "str1          ";
str1.print_bits();
std::cout << "str2          ";
str2.print_bits();
std::cout << "str1 or str2  ";
str1.BS_or(str2).print_bits();
separator();
std::cout << "str1          ";
str1.print_bits();
std::cout << "str2          ";
str2.print_bits();
std::cout << "str1 xor str2 ";
str1.BS_xor(str2).print_bits();
separator();
std::cout << "str1          ";
str1.print_bits();
std::cout << "not str1      ";
str1.BS_not().print_bits();
separator();
std::cout << "str2          ";
str2.print_bits();
std::cout << "not str2      ";
str2.BS_not().print_bits();
separator();
std::cout << "Num of units in str1: " << str1.numof_units() << std::endl;
std::cout << "Num of units in str2: " << str2.numof_units() << std::endl;
separator();
BigString* p = str1.comparison(str2);
std::cout << "Max in str1 and str2:\n";
p->print_bits();

fin.close();
return 0;
```

}

CmakeLists.txt:
```
cmake_minimum_required(VERSION 3.1)
set(MAIN_CPP source/main.cpp)
set(FIGURE_CPP source/Figure.cpp)
set(HEXAGON_CPP source/Hexagon.cpp)
set(OCTAGON_CPP source/Octagon.cpp)
set(TRIANGLE_CPP source/Triangle.cpp)

project(oop_exercise_03)

add_executable(oop_exercise_03 ${MAIN_CPP} ${FIGURE_CPP} $
{HEXAGON_CPP} ${OCTAGON_CPP} ${TRIANGLE_CPP})

set_property(TARGET oop_exercise_03 PROPERTY CXX_STANDART 11)
```

2. Ссылка на репозиторий на GitHub.
https://github.com/ArtemKD/oop_exercise_03
3. Набор testcases.
test_01.txt:
```
1 1 1 2 3 3 5 2 5 1 3 0
0 0 0 1 0 2 1 2 2 2 2 1 2 0 1 0
1 1 1 2 2 1
```

test_02.txt:
```
1 1 1 2 3 3  5 2 5  1 3 0
1 6 3 9 6 11 9 9 11 6 9 3 6 1 3 3
1 1 3 3 5 1
```

4. Результаты выполнения тестов.
test_01.txt:
```
Фигура: (1, 1)(1, 2)(3, 3)(5, 2)(5, 1)(3, 0)
Геометрический центр: (3, 1.5)
Площать: 8
---------------------------
Фигура: (0, 0)(0, 1)(0, 2)(1, 2)(2, 2)(2, 1)(2, 0)(1, 0)
Геометрический центр: (1, 1)
Площать: 4
---------------------------
Фигура: (1, 1)(1, 2)(2, 1)
Геометрический центр: (1.33333, 1.33333)
Площать: 0.5
```

test_02.txt:
Фигура: (1, 1)(1, 2)(3, 3)(5, 2)(5, 1)(3, 0)
Геометрический центр: (3, 1.5)
Площать: 8

Фигура: (1, 6)(3, 9)(6, 11)(9, 9)(11, 6)(9, 3)(6, 1)(3, 3)
Геометрический центр: (6, 6)
Площать: 60

Фигура: (1, 1)(3, 3)(5, 1)
Геометрический центр: (3, 1.66667)
Площать: 4

5. Объяснение результатов работы программы.

В программе build/oop_exercise_01 реализовано меню с пунктами:

1)Add (Добавление элемента в вектор фигур)

2)Functions(Вычисление площади и геометрического центра для всех фигур в векторе)

3)Remove(Удаление фигру по заданному индексу.

4)Test(Выполнение теста (путь указывается от текущей папки))

5)Exit(Завершение выполнения программы)

6. Вывод.

Выполняя данную лабораторную я получил опыт работы с наследованием, а также изучил основы работы с классами в C++. Создал абстрактный класс Figure и классы, наследующиеся от него классы hexagon, octagon и triangle. Реализовал нахождение площади и геометрического центра каждой фигуры.