Московский Авиационный Институт
(Национальный Исследовательский Университет)

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

Лабораторная работа №1.

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

Москва
2017

ЦЕЛЬ РАБОТЫ

Целью лабораторной работы является:
• Программирование классов на языке C++
• Управление памятью в языке C++
• Изучение базовых понятий ООП.
• Знакомство с классами в C++.
• Знакомство с перегрузкой операторов.
• Знакомство с дружественными функциями.
• Знакомство с операциями ввода-вывода из стандартных библиотек.

ЗАДАНИЕ

Необходимо спроектировать и запрограммировать на языке C++ классы фигур, согласно вариантов задания.
Классы должны удовлетворять следующим правилам:
• Должны иметь общий родительский класс Figure.
• Должны иметь общий виртуальный метод Print, печатающий параметры фигуры и ее тип в стандартный поток вывода cout.
• Должный иметь общий виртуальный метод расчета площади фигуры – Square.
• Должны иметь конструктор, считывающий значения основных параметров фигуры из стандартного потока cin.
• Должны быть расположенны в раздельных файлах: отдельно заголовки (.h), отдельно описание методов (.cpp).

Программа должна позволять вводить фигуру каждого типа с клавиатуры, выводить параметры фигур на экран и их площадь.

Теория

Классы и объекты в C++ являются основными концепциями объектно-ориентированного программирования — ООП. Объектно-ориентированное программирование — расширение структурного программирования, в котором основными концепциями являются понятия классов и объектов. Основное отличие языка программирования C++ от C состоит в том, что в C нет классов, а следовательно язык C не поддерживает ООП, в отличие от C++.
Классы в C++ — это абстракция описывающая методы, свойства, ещё не существующих объектов. **Объекты** — конкретное представление абстракции, имеющее свои свойства и методы. Созданные объекты на основе одного класса называются экземплярами этого класса. Эти объекты могут иметь различное поведение, свойства, но все равно будут являться объектами одного класса. В ООП существует три основных принципа построения классов:

1. **Инкапсуляция** — это свойство, позволяющее объединить в классе и данные, и методы, работающие с ними и скрыть детали реализации от пользователя.
2. **Наследование** — это свойство, позволяющее создать новый класс-потомок на основе уже существующего, при этом все характеристики класса родителя присваиваются классу-потомку.
3. **Полиморфизм** — свойство классов, позволяющее использовать объекты классов с одинаковым интерфейсом без информации о типе и внутренней структуре объекта.

Перегрузка - это возможность поддерживать несколько функций с одним названием, но разными сигнатурами вызова.
Дружественная функция — это функция, которая не является членом класса, но имеет доступ к членам класса, объявленным в полях `private` или `protected`

Листинг:

figure.h

```
#ifndef FIGURE_H
#define FIGURE_H

class Figure
{
public:
      virtual ~Figure() {}
      virtual void print() const = 0;
      virtual double area() const = 0;
};
#endif
```

square.h

```
#ifndef SQUARE_H
#define SQUARE_H
#include <iostream>
#include "figure.h"

class Square : public Figure
{
      public:
      Square();
      Square(std::istream& is);
      Square(size_t i);

      void print() const override;
      double area() const override;
private:
      double m_side;
};
#endif
```

square.cpp

```
#include "square.h"
Square::Square()
{
      m_side = 0.0;
}

Square::Square(std::istream& is)
{
      std::cout << "================" << std::endl;
      std::cout << "Enter side: ";
      is >> m_side;
      if (!is){
      m_side=0;
      is.clear();
      is.ignore();
      }
}

Square::Square(size_t i) : m_side(i)
{
    std::cout << "Square passed to function. Side: " << m_side <<
std::endl;
}
```

```
void Square::print() const
{
	std::cout << "================" << std::endl;
	std::cout << "Figure type: square" << std::endl;
	std::cout << "Side size: " << m_side << std::endl;
}

double Square::area() const
{
	return m_side * m_side;
}
```

rectangle.h

```
#ifndef RECTANGLE_H
#define RECTANGLE_H
#include <iostream>
#include "figure.h"

class Rectangle : public Figure
{
public:
	Rectangle();
	Rectangle(std::istream& is);
	Rectangle(size_t i, size_t j);

	void print() const override;
	double area() const override;

private:
	double m_sideA;
	double m_sideB;
};

#endif
```

rectangle.cpp

```
#include "rectangle.h"
Rectangle::Rectangle()
{
	m_sideA = 0.0;
	m_sideB = 0.0;
}

Rectangle::Rectangle(std::istream& is)
{
	std::cout << "================" << std::endl;
	std::cout << "Enter side A: ";
	is >> m_sideA;
	if (!is){
	m_sideA=0;
	is.clear();
	is.ignore();}
	std::cout << "Enter side B: ";
	is >> m_sideB;
}

Rectangle::Rectangle(size_t i,size_t j) : m_sideA(i), m_sideB(j)
{
    std::cout << "Rectangle passed to function. Sides: " << m_sideA    <<
", " << m_sideB << std::endl;
}
```

```
void Rectangle::print() const
{
	std::cout << "================" << std::endl;
	std::cout << "Figure type: rectangle" << std::endl;
	std::cout << "Side A size: " << m_sideA << std::endl;
	std::cout << "Side B size: " << m_sideB << std::endl;
}

double Rectangle::area() const
{
	return m_sideA * m_sideB;
}
```

trapezoid.h

```
#ifndef TRAPEZOID_H
#define TRAPEZOID_H

#include <iostream>
#include "figure.h"

class Trapezoid : public Figure
{
public:
	Trapezoid();
	Trapezoid(std::istream& is);
	Trapezoid(size_t i,size_t j,size_t k);

	void print() const override;
	double area() const override;

private:
	double m_sideA;
	double m_sideB;
	double m_height;
};

#endif
```

trapezoid.cpp

```
#include "trapezoid.h"
Trapezoid::Trapezoid()
{
	m_sideA = 0.0;
	m_sideB = 0.0;
	m_height = 0.0;
}

Trapezoid::Trapezoid(std::istream& is)
{
	std::cout << "================" << std::endl;
	std::cout << "Enter side A: ";
	is >> m_sideA;
	if (!is){
	m_sideA=0;
	is.clear();
	is.ignore();}
	std::cout << "Enter side B: ";
	is >> m_sideB;
```

```
            std::cout << "Enter height: ";
            is >> m_height;
      }

      Trapezoid::Trapezoid(size_t i, size_t j, size_t k) : m_sideA(i),
      m_sideB(j), m_height(k) {
          std::cout << "Trapezoid passed to function. Sides: " << m_sideA    <<
", " << m_sideB << ", " << m_height << std::endl;
      }

      void Trapezoid::print() const
      {
            std::cout << "================" << std::endl;
            std::cout << "Figure type: trapezoid" << std::endl;
            std::cout << "Side A size: " << m_sideA << std::endl;
            std::cout << "Side B size: " << m_sideB << std::endl;
            std::cout << "Height: " << m_height << std::endl;
      }

      double Trapezoid::area() const
      {
            return m_height * (m_sideA + m_sideB) / 2.0;
      }
```

main.cpp

```
      #include "square.h"
      #include "rectangle.h"
      #include "trapezoid.h"
      void testFigure(Figure* figure);
      int main()
      {
      char action;
            while (action)
            {
                  std::cout << "================" << std::endl;
                  std::cout << "Menu:" << std::endl;
                  std::cout << "1)Square" << std::endl;
                  std::cout << "2)Rectangle" << std::endl;
                  std::cout << "3)Trapezoid" << std::endl;
                  std::cout << "0) Quit" << std::endl;
                  std::cin >> action;

                  if (action == '0'){break;}

                  if (action == '1'){
                        testFigure(new Square(std::cin));
                        break;
                  }
                  if (action == '2'){
                  testFigure(new Rectangle(std::cin));
                  break;
                  }
                  if (action == '3'){
                  testFigure(new Trapezoid(std::cin));
                  break;
                  }

            }
      }
      void testFigure(Figure* figure){
            figure->print();

            std::cout << "Area: " << figure->area() << std::endl;
```

```
        delete figure;
    }
```

Выводы: в данной лабораторной работе я получил навыки пограммирования классов на языке C++, познакомился с перегрузкой операторов и дружественными функциями.

https://github.com/israelcode/oop/tree/master/sem2/lab1