Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение

высшего образования

**«Пермский национальный исследовательский**

**политехнический университет»**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

направление подготовки: 09.03.01 - «Информатика и вычислительная техника»

**Отчет по лабораторной работе “Классы” №4**

**по дисциплине**

**«Теория алгоритмов и структуры данных»**

Выполнил студент гр. ИВТ-21-1б

Ахунов Руслан Булатович

Проверил:

Ст. Преподаватель кафедры ИТАС

Яруллин Денис Владимирович

(оценка) (подпись)

г. Пермь - 2022

**Постановка задачи:**

1.  Определить пользовательский класс.

2.  Определить в классе следующие конструкторы: без параметров, с параметрами, копирования.

3.  Определить в классе деструктор.

4.  Определить в классе компоненты-функции для просмотра и установки полей данных (селекторы и модификаторы).

5.  Перегрузить операцию присваивания.

6.  Перегрузить операции ввода и вывода объектов с помощью потоков.

7.  Определить производный класс.

8.  Написать программу, в которой продемонстрировать создание объектов и работу всех перегруженных операций.

9.      Реализовать функции, получающие и возвращающие объект базового класса. Продемонстрировать принцип подстановки.

Вариант 2:

Базовый класс:

ПАРА\_ЧИСЕЛ (PAIR)

Первое\_число (first) - int Второе\_число (second) – int

Определить методы изменения полей и вычисления произведения чисел. Создать производный класс ПРЯМОУГОЛЬНИК (RECTANGLE), с полями- сторонами. Определить методы для вычисления площади и периметра прямоугольника.

**Анализ задачи:**

class Pair

{

protected:

    int first;

    int second;

public:

    Pair()

    {

        first = 0;

        second = 0;

    }

    Pair(int f, int s)

    {

        first = f;

        second = s;

    }

    Pair(const Pair& tmp)

    {

        first = tmp.first;

        second = tmp.second;

    }

    void setFirst(int f)

    {

        first = f;

    }

    void setSecond(int s)

    {

        second = s;

    }

    int getFirst() { return first; }

    int getSecond() { return second; }

    Pair& operator=(const Pair& p)

    {

        if (&p == this) return \*this;

        first = p.first;

        second = p.second;

        return \*this;

    }

    Pair operator\*(const Pair tmp)

    {

        first = first \* tmp.first;

        second = second \* tmp.second;

        return \*this;

    }

    friend istream& operator>>(istream& in, Pair& tmp);

    friend ostream& operator<<(ostream & out, const Pair& tmp);

    virtual ~Pair()

    {

        // cout << "Удалили Pair..." << endl << endl;

    }

};

istream& operator>>(istream& in, Pair& tmp)

{

    cout << "Введите первое число: "; in >> tmp.first;

    cout << "Введите второе число: "; in >> tmp.second;

    return in;

}

ostream& operator<<(ostream& out, const Pair& tmp)

{

    return (out << "(" << tmp.first << "," << tmp.second << ")");

}

Класс Pair, в спецификаторе доступа protected имеется два поля first и second. В спецификаторе доступа Public, определены конструкторы без параметров, с параметрами и копирования; «Сеторы» и «Гетеры» для полей; Перегружены операторы присваивания и умножения; Так же перегружены операторы потокового ввода и вывода.

class Rect : public Pair

{

public:

    int P()

    {

        int p;

        p = (first + second) \* 2;

        return p;

    }

    int S()

    {

        int s;

        s = first \* second;

        return s;

    }

    Rect()

    {

        first = 0;

        second = 0;

    }

    Rect(int f, int s)

    {

        first = f;

        second = s;

    }

    Rect(const Rect& tmp)

    {

        first = tmp.first;

        second = tmp.second;

    }

    void setFirst(int f)

    {

        first = f;

    }

    void setSecond(int s)

    {

        second = s;

    }

    int getFirst() { return first; }

    int getSecond() { return second; }

    Rect& operator=(const Rect& p)

    {

        if (&p == this) return \*this;

        first = p.first;

        second = p.second;

        return \*this;

    }

    friend istream& operator>>(istream& in, Rect& tmp);

    friend ostream& operator<<(ostream& out, const Rect& tmp);

    ~Rect()

    {

        // cout << "Удалили Rect..." << endl << endl;

    }

};

istream& operator>>(istream& in, Rect& tmp)

{

    cout << "Введите одну сторону прямоугольника: "; in >> tmp.first;

    cout << "Введите другую сторону прямоугольника: "; in >> tmp.second;

    return in;

}

ostream& operator<<(ostream& out, const Rect& tmp)

{

    return (out << "Одна сторона равна: " << tmp.first << " / " << "Другая стороная равна: " << tmp.second);

}

Наследуемы класс Rect, к котором так же есть свои конструкторы без параметров, с параметрами и копирования. Свои «Сетеры» и «Гетеры», а так же перегруженные операторы присваивания и потокового ввода и вывода. Так же есть дополнительные методы, такие как P – находит периметр фигуры, и S – находит площадь фигуры.

void f1(Pair& c)

{

    c.setFirst(20);

    c.setSecond(22);

    cout << c;

}

Pair f2()

{

    Rect r(50, 20);

    return r;

}

int main()

{

    setlocale(LC\_ALL, "Russian");

    Pair a;

    cin >> a;

    cout << a << endl;

    Pair b(34, 76);

    cout << b << endl;

    cout << "Произведение a и b = " << a \* b << endl;

    a = b;

    cout << a << endl;

    Rect c;

    cin >> c;

    cout << c << endl;

    cout << "Периметр прямоугольника равен: " << c.P() << endl;

    cout << "Площадь прямоугольника равна: " << c.S() << endl << endl;

    f1(c);

    a = f2();

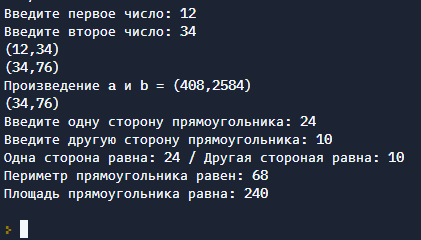
    cout << a;

}

В функции f1 мы изменяем переменную с помощью «сетеров». В функции f2 мы используем конструктор с параметрами и возвращаем данную переменну.

main() – объявляем переменную типа Pair а, b. Используем оператор произведения и выводим полученный результат. Теперь переходим к типу Rect, находим периметр и площадь данной переменной. Используем функции f1 и f2.

**Результат работы программы**

****

Вводим значения для переменной, смотрим что получилось. Находим произведение двух переменных. Затем переходим к другому типу данных, вводим новые значения, получаем периметр и площадь.

**Код программы**

#include <iostream>

using namespace std;

class Pair

{

protected:

    int first;

    int second;

public:

    Pair()

    {

        first = 0;

        second = 0;

    }

    Pair(int f, int s)

    {

        first = f;

        second = s;

    }

    Pair(const Pair& tmp)

    {

        first = tmp.first;

        second = tmp.second;

    }

    void setFirst(int f)

    {

        first = f;

    }

    void setSecond(int s)

    {

        second = s;

    }

    int getFirst() { return first; }

    int getSecond() { return second; }

    Pair& operator=(const Pair& p)

    {

        if (&p == this) return \*this;

        first = p.first;

        second = p.second;

        return \*this;

    }

    Pair operator\*(const Pair tmp)

    {

        first = first \* tmp.first;

        second = second \* tmp.second;

        return \*this;

    }

    friend istream& operator>>(istream& in, Pair& tmp);

    friend ostream& operator<<(ostream & out, const Pair& tmp);

    virtual ~Pair()

    {

        // cout << "Удалили Pair..." << endl << endl;

    }

};

istream& operator>>(istream& in, Pair& tmp)

{

    cout << "Введите первое число: "; in >> tmp.first;

    cout << "Введите второе число: "; in >> tmp.second;

    return in;

}

ostream& operator<<(ostream& out, const Pair& tmp)

{

    return (out << "(" << tmp.first << "," << tmp.second << ")");

}

class Rect : public Pair

{

public:

    int P()

    {

        int p;

        p = (first + second) \* 2;

        return p;

    }

    int S()

    {

        int s;

        s = first \* second;

        return s;

    }

    Rect()

    {

        first = 0;

        second = 0;

    }

    Rect(int f, int s)

    {

        first = f;

        second = s;

    }

    Rect(const Rect& tmp)

    {

        first = tmp.first;

        second = tmp.second;

    }

    void setFirst(int f)

    {

        first = f;

    }

    void setSecond(int s)

    {

        second = s;

    }

    int getFirst() { return first; }

    int getSecond() { return second; }

    Rect& operator=(const Rect& p)

    {

        if (&p == this) return \*this;

        first = p.first;

        second = p.second;

        return \*this;

    }

    friend istream& operator>>(istream& in, Rect& tmp);

    friend ostream& operator<<(ostream& out, const Rect& tmp);

    ~Rect()

    {

        // cout << "Удалили Rect..." << endl << endl;

    }

};

istream& operator>>(istream& in, Rect& tmp)

{

    cout << "Введите одну сторону прямоугольника: "; in >> tmp.first;

    cout << "Введите другую сторону прямоугольника: "; in >> tmp.second;

    return in;

}

ostream& operator<<(ostream& out, const Rect& tmp)

{

    return (out << "Одна сторона равна: " << tmp.first << " / " << "Другая стороная равна: " << tmp.second);

}

void f1(Pair& c)

{

    c.setFirst(20);

    c.setSecond(22);

    cout << c;

}

Pair f2()

{

    Rect r(50, 20);

    return r;

}

int main()

{

    setlocale(LC\_ALL, "Russian");

    Pair a;

    cin >> a;

    cout << a << endl;

    Pair b(34, 76);

    cout << b << endl;

    cout << "Произведение a и b = " << a \* b << endl;

    a = b;

    cout << a << endl;

    Rect c;

    cin >> c;

    cout << c << endl;

    cout << "Периметр прямоугольника равен: " << c.P() << endl;

    cout << "Площадь прямоугольника равна: " << c.S() << endl << endl;

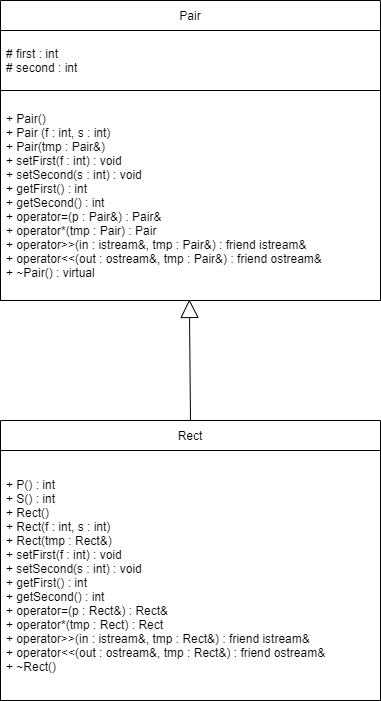
    f1(c);

    a = f2();

    cout << a;

}

**Диаграмма класса**



**Вопросы**

* 1. **Для чего используется механизм наследования?**

Наследование - это механизм получения нового класса на основе уже существующего. Существующий класс может быть дополнен или изменен для создания нового класса.

* 1. **Каким образом наследуются компоненты класса, описанные со спецификатором public?**

public – член класса может использоваться любой функцией, которая является членом данного или производного класса, а также к public - членам возможен доступ извне через имя объекта.

* 1. **Каким образом наследуются компоненты класса, описанные со спецификатором private?**

private – член класса может использоваться только функциями – членами данного класса и функциями – “друзьями” своего класса. В производном классе он недоступен.

* 1. **Каким образом наследуются компоненты класса, описанные со спецификатором protected?**

protected – то же, что и private, но дополнительно член класса с данным атрибутом доступа может использоваться функциями-членами и функциями – “друзьями” классов, производных от данного.

* 1. **Каким образом описывается производный класс?**

Синтаксис определения производного класса:

class имя\_класса : список\_базовых\_классов

{список\_компонентов\_класса};

* 1. **Наследуются ли конструкторы?**

Нет.

* 1. **Наследуются ли деструкторы?**

Нет.

* 1. **В каком порядке конструируются объекты производных классов?**

Объекты класса конструируются снизу вверх: сначала базовый, потом компоненты- объекты (если они имеются), а потом сам производный класс. Таким образом, объект производного класса содержит в качестве подобъекта объект базового класса.

* 1. **В каком порядке уничтожаются объекты производных классов?**

Уничтожаются объекты в обратном порядке: сначала производный, потом его компоненты-объекты, а потом базовый объект.

* 1. **Что представляют собой виртуальные функции и механизм позднего связывания?**

К механизму виртуальных функций обращаются в тех случаях, когда в каждом производном классе требуется свой вариант некоторой компонентной функции. Классы, включающие такие функции, называются полиморфными и играют особую роль в ООП.

Виртуальные функции предоставляют механизм позднего (отложенного) или динамического связывания. Любая нестатическая функция базового класса может быть сделана виртуальной, для чего используется ключевое слово virtual.

* 1. **Могут ли быть виртуальными конструкторы? Деструкторы?**

Конструкторы не могут быть виртуальными, в отличие от деструкторов.

* 1. **Наследуется ли спецификатор virtual?**

Да.

* 1. **Какое отношение устанавливает между классами открытое наследование?**

Private – не наследуется, protected – protected, public – public.

* 1. **Какое отношение устанавливает между классами закрытое наследование?**

Private – не наследуется, protected – private, public – private.

* 1. **В чем заключается принцип подстановки?**

Открытое наследование устанавливает между классами отношение «является»: класс-наследник является частью класса-родителя. Это означает, что везде, где может быть использован объект базового класса (при присваивании, при передаче параметров и возврате результата), вместо него разрешается использовать объект производного класса.

Закрытое наследование – это наследование реализации, в этом случае принцип подстановки не соблюдается.

* 1. **Имеется иерархия классов: class Student**

**{**

**public:**

**};**

**int age;**

**string name;**

**...**

**class Employee : public Student**

**{**

**protected:**

**string post;**

**...**

**};**

**class Teacher : public Employee**

**{**

**protected: int stage;**

**...**

**};**

**Teacher x;**

**Какие компонентные данные будет иметь объект х?**

Protected: int stage, string post;

public: string name;

* 1. **Для классов Student, Employee и Teacher написать конструкторы без параметров.**

Student() {age = 0; name = “”;}

Employee() {name = “”, job = “”;}

Teacher() {stage = 0; name = “”, job = “”;}

* 1. **Для классов Student, Employee и Teacher написать конструкторы с параметрами.**

Student(int a, string n) {age = a; name = n;}

Employee(string j, int zp) {job = n, int = zp;}

Teacher(int s, string n, string j) {stage = s; name = n, job = j;}

* 1. **Для классов Student, Employee и Teacher написать конструкторы копирования.**

Student(const Student& t) {age = t.age; name = t.name;}

Employee(const Employee& t) {name = t.name, job = t.job;}

Teacher(const Teacher& t) {stage = t.stage; name = t.name, job = p.job;}

* 1. **Для классов Student, Employee и Teacher определить операцию присваивания.**

Student& operator =(const Student&c){

Age = c.age;

Name = c.name;

Return \*this;

}

Employee& operator =(const Employee&c){

post = c.job;

Name = c.name;

Return \*this;

}

Teacher& operator =(const Teacher&c){

Stage = c.stage;

Post = c.job;

Name = c.name;

Return \*this;