Министерство науки и высшего образования Российской Федерации  
Федеральное государственное автономное учреждение высшего образования  
**«Пермский национальный исследовательский политехнический университет»**

**(ПНИПУ)**

Электротехнический факультет  
Кафедра «Информационные технологии и автоматизированные системы»

**Отчет о лабораторной работе №4**

Вариант №4

Дисциплина: основы алгоритмизации и программирования

Тема: классы в С++

Выполнил студент ИВТ-22-2б:

Якушев Михаил Витальевич

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Проверил доцент кафедры ИТАС:

Полякова Ольга Андреевна

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата, подпись)

Пермь 2023

**1 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.

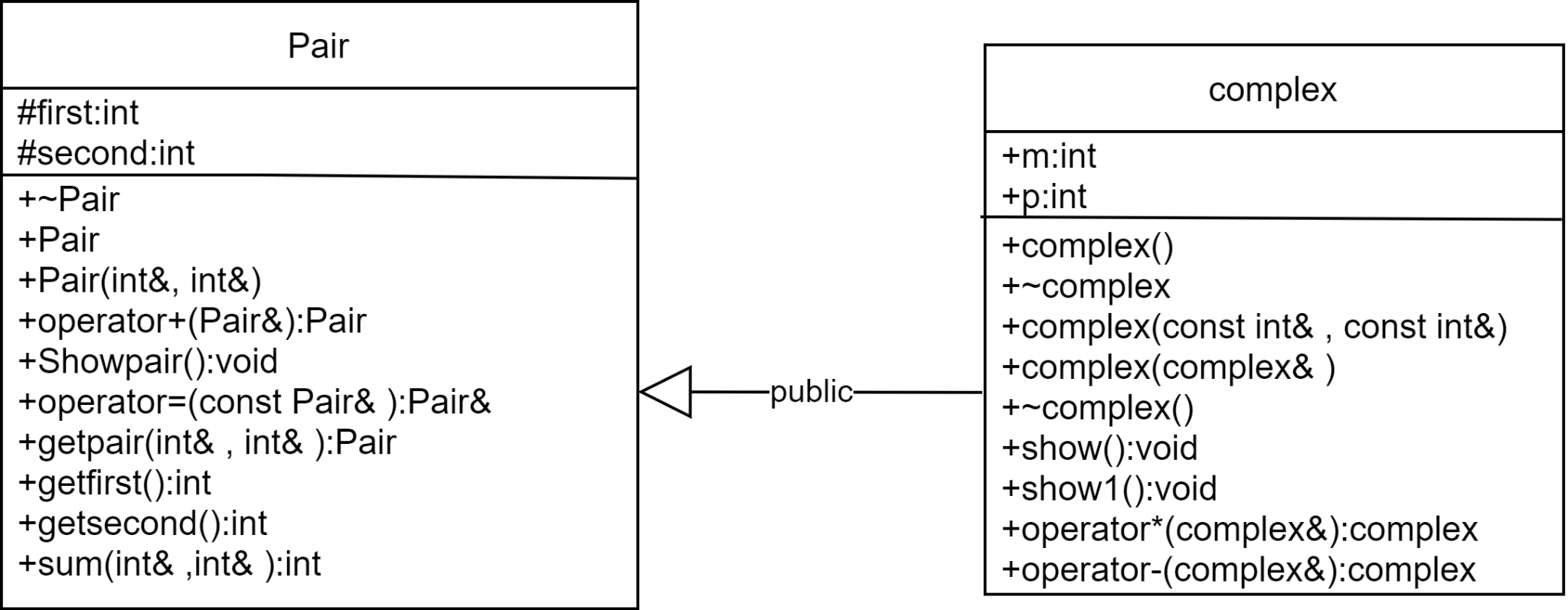
9. Реализовать функции, получающие и возвращающие объект базового класса.

Продемонстрировать принцип подстановки.

**2 Анализ задачи**

Класс complex наследуется от класса Pair, в методе void show1 класса complex вызывается методы getpair и sum из класса Pair, этот вызов демонстрирует метод подстановки.

**3 UML-диаграмма**

****

**4 Код программы**

#include <iostream>

using namespace std;

class Pair {

protected:

int first;

int second;

public:

~Pair();

Pair();

Pair(int&, int&);

Pair(Pair&);

Pair operator+(Pair&);

void Showpair() {

cout << this->first << this->second << endl;

}

friend ostream& operator<<(ostream& stream, const Pair& tm);

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

Pair& operator=(const Pair& a);

Pair getpair(int& t, int& r) {

Pair a(t,r);

return a;

}

int getfirst() {

return this->first;

}

int getsecond() {

return this->second;

}

int sum(int& t,int& p) {

Pair a(t, p);

return a.first + a.second;

}

};

class complex: public Pair {

public:

int m;

int p;

complex();

complex(const int& , const int&);

complex(complex& );

~complex();

void show(){

cout << '(' << this->m << ',' << this->p << "i)" << endl;

}

void show1() {

cout << getpair(this->m,this->p) << "sum: " << sum(this->m, this->p) << endl;

}

complex operator\*(complex& );

complex operator-(complex&);

friend ostream& operator<<(ostream& stream, const complex& tm);

friend istream& operator>>(istream&, complex&);

};

Pair::~Pair() {

}

Pair::Pair() {

first = 0;

second = 0;

}

Pair::Pair(int& first, int& second){

this->first = first;

this->second = second;

}

Pair::Pair(Pair& tmp) {

this->first = tmp.first;

this->second = tmp.second;

}

Pair Pair:: operator+(Pair& p) {

int tmp = this->first + p.first;

int tmp1 = this->second + p.second;

Pair a(tmp, tmp1);

return a;

}

ostream& operator<<(ostream& stream, const Pair& tm) {

stream << '(' << tm.first << ',' << tm.second << ')';

return stream;

};

istream& operator>>(istream& stream1, Pair& tm) {

stream1 >> tm.first;

stream1 >> tm.second;

return stream1;

};

Pair& Pair::operator=(const Pair& a) {

this->first = a.first;

this->second = a.second;

return \*this;

}

complex::~complex() {

}

complex::complex() {

}

complex::complex(const int& m, const int& p) {

this->m = m;

this->p = p;

}

complex::complex(complex& tmp) {

this->m = tmp.m;

this->p = tmp.p;

}

complex complex::operator\*(complex& t) {

int tmp = this->m \* t.m - (this->p \* t.p);

int tmp1 = this->m \* t.p + (this->p\*t.m);

complex a(tmp, tmp1);

return a;

}

complex complex ::operator-(complex& tp) {

int tmp = this->m-this->p;

int tmp1 =tp.m - tp.p;

complex a(tmp, tmp1);

return a;

}

ostream& operator<<(ostream& stream, const complex& tm) {

stream << '(' << tm.m << ',' << tm.p << "i)";

return stream;

};

istream& operator>>(istream& stream1, complex& tm) {

stream1 >> tm.m;

stream1 >> tm.p;

return stream1;

};

int main()

{

setlocale(LC\_ALL, "ru");

Pair a,b;

cout << "Введите первую пару чисел: ";

cin >> a;

cout << "Введите вторую пару чисел: ";

cin >> b;

cout << "Первая: " << a <<"\t" <<"Вторая: " << b <<endl;

cout << "a+b " << a + b << endl;

a = b;

cout << "a=b " << a <<endl;

complex d,r;

cout << "Введите комплексное число: ";

cin >> d;

cout << "Введите комплексное число: ";

cin >> r;

cout << r << d <<endl;

cout << "r \* d = " << r \* d << endl;

cout << "r - d = " << r - d << endl;

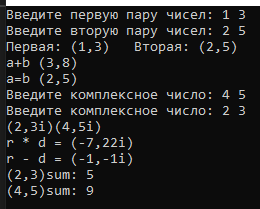
r.show1();

d.show1();

return 0;

}

**5 Результаты работы программы**

****

**6 Ответы на вопрос**

1. Для чего используется механизм наследования?

Наследование позволяет создавать новые классы, которые повторно используют, расширяют и изменяют функционал, определённый в других классах.

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

Член класса, описанные спецификатором public, могут 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?

Члены класса, описанные спецификатором private, могут использоваться только функциями – членами данного класса и функциями – “друзьями” своего класса. В производном классе они недоступны.

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

Члены класса, описанные спецификатором protected, как и описанные private, но дополнительно члены класса с данным атрибутом доступа могут 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};

class Pet

{

///

};

class Person :Pet

{

///

};

1. Наследуются ли конструкторы?

Поскольку конструкторы не наследуются, при создании производного класса наследуемые им данные-члены должны инициализироваться конструктором базового класса. Параметры конструктора базового класса указываются в определении конструктора производного класса. Таким образом, происходит передача аргументов от конструктора производного класса конструктору базового класса.

class Pet

{

protected:

int a, b;

public:

Pet(int x, int y) // Конструктор

{

a = x;

b = y;

}

};

class Person : Pet

{

protected:

int sum;

public:

Person(int x, int y, int s) :Pet(x, y)

{

sum = s;

}

};

1. Наследуются ли деструкторы?

Деструкторы не наследуются, но они автоматически вызываются, когда объект производного класса выходит из области видимости.

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.

Таким образом, порядок уничтожения объекта противоположен по отношению к порядку его конструирования.

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. Виртуальные функции обеспечивают вызов соответствующей функции для объекта независимо от выражения, используемого для вызова функции.

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

class Base

{

public:

virtual void print()

{

cout << "\nBase";

}

};

class Derive : public Base

{

public:

void print()

{

cout << "\nDerive";

}

};

void main()

{

Base B, \* bp;

Derive D, \* dp;

bp = &B;

dp = &D;

//указатель базового класса ставится на объект

//производного класса Base \*p = &D;

bp -> print(); // вызывается метод для Base

dp -> print(); // вызывается метод для Derive p –>print(); // вызывается метод для Derive

}

Таким образом, интерпретация каждого вызова виртуальной функции через указатель на базовый класс зависит от значения этого указателя, т.е. от типа объекта, для которого выполняется вызов.

1. Могут ли быть виртуальными конструкторы? Деструкторы?

Конструкторы не могут быть виртуальными, в отличие от деструкторов. Практически каждый класс, имеющий виртуальную функцию, должен иметь виртуальный деструктор.

12. Наследуется ли спецификатор virtual?

Виртуальность наследуется. После того как функция определена как виртуальная, её повторное определение в производном классе (с тем же самым прототипом) создаёт в этом классе новую виртуальную функцию, причём спецификатор virtual может не использоваться.

1. Какое отношение устанавливает между классами открытое наследование?

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.

1. Какое отношение устанавливает между классами закрытое наследование?

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

Программа, использующая наследующий класс, не может обратиться к методам, находящимся в базовом классе, для этого потребуется создать в наследующем классе функцию, вызывающую заприваченную функцию в Base.

class Base

{

public:

int number;

void func1();

};

class Derive : private Base // Закрытое наследование

{

public:

void f1() // метод выполняет то же, что и метод в Base

{

Base::func1();

}

};

1. В чем заключается принцип подстановки?

Принцип подстановки заключается в том, что везде, где может быть использован объект базового класса (при присваивании, при передаче параметров и возврате результата), вместо него разрешается использовать объект производного класса.

16. Имеется иерархия классов:

class Student

{

int age;

public:

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?

Он будет иметь:

- public: string name

- protected: string post

- protected: int stage

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

class Student

{

int age;

public:

string name;

Student() // Пустой конструктор

{

age = 0;

name = "";

}

};

class Employee : public Student

{

protected:

string post;

Employee() : Student()

{

post = "";

}

};

class Teacher : public Employee

{

protected:

int stage;

Teacher() : Employee()

{

stage = 0;

}

};

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

Student::Student(int age, string name)

{

this->age = age;

this->name = name;

}

Employee::Employee(string name, string post)

{

this->name = name;

this->post = post;

}

Teacher::Teacher(string name, string post, int stage)

{

this->name = name;

this->post = post;

this->stage = stage;

}

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

Student::Student(const Student&p)

{

age = p.age;

name = p.name;

}

Employee::Employee(const Employee&r)

{

name = r.name;

post = r.post;

}

Teacher::Teacher(const Teacher&s)

{

name = s.name;

post = s.post;

stage = s.stage;

}

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

Student& Student::operator=(const Student&p)

{

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

age = p.age;

return \*this;

}

Employee& Employee::operator=(const Employee& r)

{

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

name = r.name;

post = r.post;

return \*this;

}

Teacher& Teacher::operator=(const Teacher&s)

{

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

name = s.name;

post = s.post;

stage = s.stage;

return \*this;

}