Министерство образования и науки Российской Федерации

Федеральное государственное бюджетное образовательное учреждение высшего образования

«**Пермский национальный исследовательский политехнический университет»**

Кафедра «Информационные технологии и автоматизированные системы»

**ОТЧЕТ**

Дисциплина: «Информатика»

Тема: Простое наследование. Принцип подстановки

Семестр 2

Выполнил работу

Студент группы РИС-22-1Б

Тарасов C.В.

Проверил

Доцент кафедры ИТАС

Полякова О.А.

Г. Пермь-2023

**Постановка задачи(общая и конкретного варианта)**

Определить пользовательский класс.

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

Определить в классе деструктор.

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

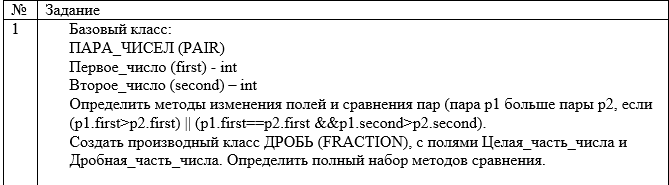
Перегрузить операцию присваивания.

Перегрузить операции ввода и вывода объектов с помощью потоков.

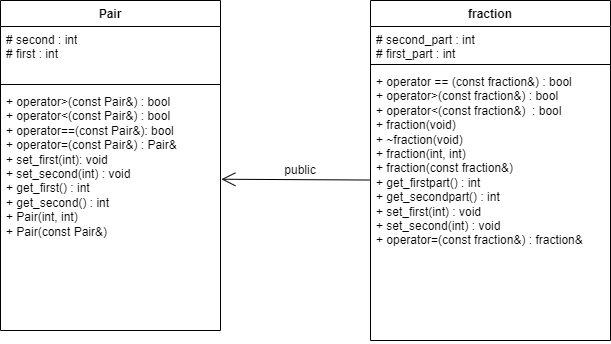
Определить производный класс.

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

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



**Описание класса**

****

**Определение компонентных функций**

Pair::Pair(void)

{

first = 0;

second = 0;

}

//деструктор

Pair::~Pair(void)

{

}

//конструктор с параметрами

Pair::Pair(int ffirst, int ssecond)

{

first = ffirst;

second = ssecond;

}

//конструктор копировани¤

Pair::Pair(const Pair& t )

{

first = t.first;

second = t.second;

}

//селекторы

int Pair::get\_first()

{

return first;

}

int Pair::get\_second()

{

return second;

}

//модификаторы

void Pair::set\_first(int ffirst)

{

first = ffirst;

}

void Pair::set\_second(int ssecond)

{

second = ssecond;

}

//перегрузка приваивани¤

Pair& Pair::operator=(const Pair& t)

{

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

first = t.first;

second = t.second;

return \*this;

}

//перегрузка ввода-вывода

//методы сравнени¤ пар из варианта

bool Pair::operator>(const Pair& t)

{

return (first > t.first) || (first == t.first) && second > t.second;

}

bool Pair::operator<(const Pair& t)

{

return (first < t.first) || (first == t.first) && second < t.second;

}

bool Pair::operator==(const Pair& t)

{

return (first == t.first) && (second == t.second);

}

fraction::fraction(void)

{

first\_part = 0;

second\_part = 0;

}

//деструктор

fraction::~fraction(void)

{

}

//конструктор с параметрами

fraction::fraction(int first\_part, int second\_part) :Pair(first\_part, second\_part)

{

this->first\_part = first\_part;

this->second\_part = second\_part;

}

//конструктор копировани¤

fraction::fraction(const fraction& t)

{

first\_part = t.first\_part;

second\_part = t.second\_part;

}

//селекторы

int fraction::get\_firstpart()

{

return first\_part;

}

int fraction::get\_secondpart()

{

return second\_part;

}

//модификаторы

void fraction::set\_first(int ffirst)

{

first\_part = ffirst;

}

void fraction::set\_second(int ssecond)

{

second\_part = ssecond;

}

//перегрузка присваивани¤

fraction& fraction::operator=(const fraction& t)

{

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

first\_part = t.first\_part;

second\_part = t.second\_part;

return \*this;

}

//перегрузка ввода-вывода

// полный набор методов сравнени¤

bool fraction::operator == (const fraction& t)

{

double value1 = first\_part / second\_part;

double value2= t.first\_part / t.second\_part;

return value1 == value2;

}

bool fraction::operator>(const fraction& t)

{

double value1 = first\_part / second\_part;

double value2 = t.first\_part / t.second\_part;

return value1 > value2;

}

bool fraction::operator<(const fraction& t)

{

double value1 = first\_part / second\_part;

double value2 = t.first\_part / t.second\_part;

return value1 < value2;

}

**Определение глобальных функций**

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

{

out << "\nПервое число в паре = " << t.first;

out << "\nВторое число в паре = " << t.second;

out << "\n";

return out;

}

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

{

cout << " Введите первое число в паре "; in >> t.first;

cout << " Введите второе число в паре "; in >> t.second;

return in;

}

ostream& operator<<(ostream& out, const fraction& t)

{

out << "\nцелая часть числа = " << t.first\_part;

out << "\nдробная часть числа = " << t.second\_part;

out << "\n";

return out;

}

istream& operator>>(istream& in, fraction& t)

{

cout << " введите целую часть числа "; in >> t.first\_part;

cout << " введите дробную часть числа "; in >> t.second\_part;

return in;

}

**Функция main()**

void f1(Pair& c)

{

c.set\_first(993);

}

Pair f2()

{

fraction l(123,321);

return l;

}

int main()

{

setlocale(LC\_ALL, "Russian");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

//работа с классом Pair

Pair a;

cin >> a;

cout <<" a "<< a;

Pair b(93, 12);

cout <<" b " <<b;

//методы сравнения пар

cout << "============ Проверка методов сравнения ============ " << endl;

if (a > b) cout << " a больше чем b " << endl;

else if (a < b) cout << " a меньше чем b " << endl;

else if (a==b) cout << " a равен b " << endl;

a = b;

cout <<"============ Проверка работы присваивания ============ "<<endl<<" a "<< a;

//Работа с классом fraction

cout << "============ Работа с fraction ============ "<<endl;

fraction c;

cin >> c;

cout << " c " << c;

fraction d(90, 9);

cout << " d " << d;

//методы сравнения

cout << "============ Проверка методов сравнения ============ " << endl;

if (c > d) cout << " Дробь c больше чем d " << endl;

else if (c < d) cout << " Дробь c меньше чем d " << endl;

else if (c == d) cout << " Дробь c равна d" << endl;

//принцип подстановки

cout << "============ Функции получающие и возвращающие объект базового класса, принцип подстановки ============ "<<endl;

f1(b);

cout << " b " << b;

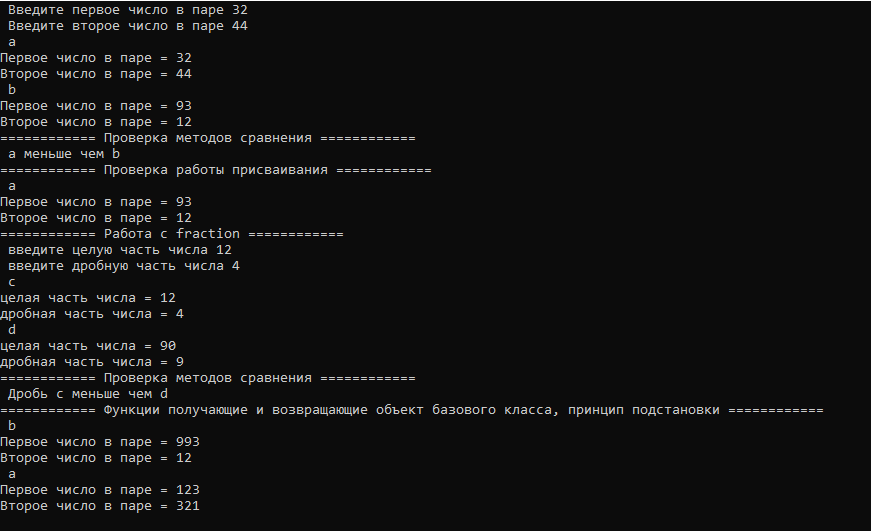
// a - Pair, f2() return - fraction,but a still Pair

a = f2();

cout << " a " << a;

}

**Объяснение результатов работы программы**



**Ответы на контрольные вопросы**

1. Для чего используется механизм наследования?

Наследование позволяет структурировать и повторно использовать код и ускорить процесс разработки. Оно позволяет выделить общее для нескольких классов поведение и вынести его в отдельную сущность - базовый класс.

2. 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.

3. 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.

4. 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.

5. Каким образом описывается производный класс?

Синтаксис определения производного класса:

class имя\_класса : список\_базовых\_классов

{список\_компонентов\_класса};

class Base{ //базовый класс

…

};

class Derive: Base { //производный класс

…

};

6. Наследуются ли конструкторы?

Конструкторы не наследуются, при создании производного класса

наследуемые им данные-члены должны инициализироваться конструктором базового класса.

7. Наследуются ли деструкторы?

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

8. 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.

9. В каком порядке уничтожаются объекты производных классов?

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

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

class Base {

public:

virtual void print(){cout<<”\nBase”;}

. . .

};

class Derive : public Base {

public:

void print(){cout<<”\n Derive”;}

};

int main() {

Base B,\*bp;

Derive D,\*dp;

bp=&B; //указатель базового класса ставится на объект

dp=&D; //производного класса

Base \*p = &D;

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

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

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

return 0;

}

11. Могут ли быть виртуальными конструкторы? Деструкторы?

Виртуальными могут быть только нестатические функции-члены, то есть конструкторы и деструкторы не могут быть виртуальными, так как они ненаследуемы.

12. Наследуется ли спецификатор virtual?

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

13. 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.Таким образом, порожденный класс представляет собой модификацию базового класса.

class A { protected: int i; };

class B: public A {...};// i остается protected членом B

class C: public B {public:void f();};

//поскольку i является protected для B, оно может наследоваться вC

void C::f(){ i = 2;} //данная функция работает т.к. i доступна в C

14. Какое отношение устанавливает между классами закрытое наследование?

Закрытое наследование означает, что от базового класса необходимо взять какую-то функциональность, базовый класс и потомок не имеют какой-либо концептуальной связи .Закрытое наследование не носит характера отношения подтипов. Закрытое (также как и защищенное) наследование не создает иерархии типов.

class A { protected: int i; };

class B: private A {...};// i преобразовано к private члену B

class C: public B {public:void f();};

//поскольку i является private для B, оно не может наследоваться вC (не создается иерархии объектов)

void C::f(){ i = 2;}// данная функция не работает т.к. i не доступна

15. В чем заключается принцип подстановки?

Принцип подстановки: класс S может считаться подклассом T, если замена объектов T на объекты S не приведёт к изменению работы программы.

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?

В классе Employee открыто наследуется public name (age не наследуется, поскольку поле private). В Teacher открыто наследуется public name, и

открыто наследуется protected post. Таким образом, у класса Teacher доступны следующие поля: public name, protected post, protected stage.

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

Student::Student(){

age = 0;

name = “NULL”; }

Employee::Employee() {

name = “NULL”;

post = “NULL”; }

Teacher::Teacher(){

name = “NULL”;

post = “NULL”;

stage = 0; }

18. 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 spost) {

this->name = name;

this->post = post; }

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

this->name = name;

this->post = post;

this->stage = stage; }

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

Student::Student(const Student& s){

age = s.age;

name = s.name; }

Employee::Employee(const Employee& e) {

name = e.name;

post = e.post; }

Teacher::Teacher(const Teacher& t){

name = t.name;

post = t.post;

stage = t.stage; }

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

Student& Student::operator = (const& Student s) {

if (&s==this) return\*this; //проверка на самоприсваивание

name = s.name;

age = s.age;

return \*this; }

Employee& Employee::operator = (const& Employee e) {

if (&e==this) return\*this; //проверка на самоприсваивание

name = e.name;

age = e.age;

return \*this; }

Teacher& Teacher::operator = (const& Teacher t) {

if (&t==this) return\*this; //проверка на самоприсваивание

name = t.name;

age = t.age;

return \*this; }