Министерство высшего образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

**«Пермский национальный исследовательский политехнический университет» (ПНИПУ)**

Электротехнический факультет

Кафедра «Информационные технологии и автоматизированные системы»

ОТЧЁТ

Лабораторная работа №4

Простое наследование. Принцип подстановки.

Выполнил

Студент группы РИС-22-2б

Вековшинин Д. А.

Проверила

Доцент кафедры ИТАС

Полякова О. А.

Пермь 2023

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 10: Базовый класс:

ТРОЙКА\_ЧИСЕЛ (TRIAD)

Первое\_число (first) - int

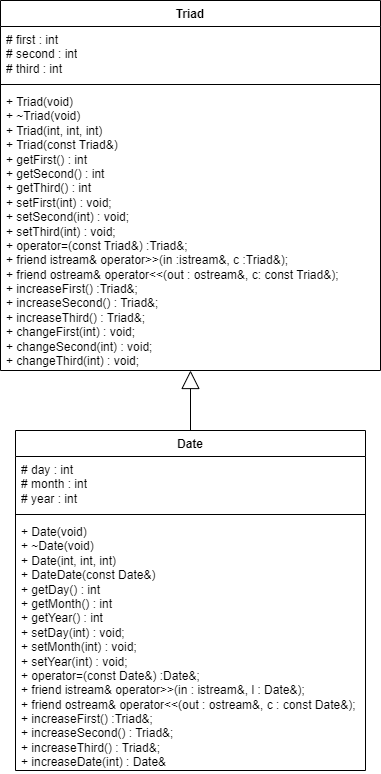
Второе\_число (second) – int

Третье\_число (third) - int

Определить методы изменения полей и увеличения полей на 1. Создать производный класс DATE с полями год, месяц и число. Переопределить

методы увеличения полей на 1 и определить метод увеличения даты на n дней.

UML-диаграмма



Определение компонентных функций класса Triad

//конструктор без параметров

Triad::Triad(void)

{

first = 0;

second = 0;

third = 0;

}

//деструктор

Triad::~Triad(void)

{

}

//конструктор с параметрами

Triad::Triad(int F, int S, int T)

{

first = F;

second = S;

third = T;

}

//конструктор копирования

Triad::Triad(const Triad& car)

{

first = car.first;

second = car.second;

third = car.third;

}

//модификаторы

void Triad::setFirst(int F)

{

first = F;

}

void Triad::setSecond(int S)

{

second = S;

}

void Triad::setThird(int T)

{

third = T;

}

//перегрузка операции присваивания

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

{

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

first = t.first;

second = t.second;

third = t.third;

return \*this;

}

//методы увеличения полей на 1

Triad& Triad::increaseFirst()

{

first++;

return \*this;

}

Triad& Triad::increaseSecond()

{

second++;

return \*this;

}

Triad& Triad::increaseThird()

{

third++;

return \*this;

}

//методы изменения полей

void Triad::changeFirst(int F)

{

first=F;

}

void Triad::changeSecond(int S)

{

second=S;

}

void Triad::changeThird(int T)

{

third=T;

}

Определение компонентных функций класса Date

//конструктор без параметров

Date::Date(void) :Triad()

{

day = 0;

year = 0;

month = 0;

}

//дестрктор

Date::~Date(void)

{

}

//конструктор с параметрами

Date::Date(int D, int M, int Y) :Triad(D, M, Y)

{

day = D;

month = M;

year = Y;

}

//конструктор копирования

Date::Date(const Date& D)

{

day = D.day;

month = D.month;

year = D.year;

}

//модификаторы

void Date::setDay(int D)

{

day = D;

}

void Date::setMonth(int M)

{

day = M;

}

void Date::setYear(int Y)

{

day = Y;

}

//оперция присваивания

Date& Date::operator=(const Date& d)

{

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

day = d.day;

month = d.month;

year = d.year;

return \*this;

}

//методы увеличения полей на 1

Date& Date::increaseFirst()

{

day++;

return \*this;

}

Date& Date::increaseSecond()

{

month++;

return \*this;

}

Date& Date::increaseThird()

{

year++;

return \*this;

}

//метод увеличения даты на n дней

Date& Date::increaseDate(int n)

{

day+=n;

return \*this;

}

Определение глобальных функций

//глобальная функция для ввода

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

{

cout << "\nFirst: "; in >> t.first;

cout << "\nSecond: "; in >> t.second;

cout << "\nThird: "; in >> t.third;

return in;

}

//глобальная функция для вывода

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

{

out << "\nFIRST : " << t.first;

out << "\nSECOND : " << t.second;

out << "\nTHIRD : " << t.third;

out << "\n";

return out;

}

//операция ввода

istream& operator>>(istream& in, Date& d)

{

cout << "\nDay: "; in >> d.day;

cout << "\nMonth: "; in >> d.month;

cout << "\nYear: "; in >> d.year;

return in;

}

//операция вывода

ostream& operator<<(ostream& out, const Date& d)

{

out << "\nDAY : " << d.day;

out << "\nMONTH : " << d.month;

out << "\nYEAR : " << d.year;

out << "\n";

return out;

}

void f1(Triad& c)

{

c.setFirst(99999);

cout << c;

}

Triad f2()

{

Date d(10, 5, 2001);

return d;

}

Функция main()

void main()

{

//работа с классом Triad

Triad a;

cin >> a;

cout << a;

Triad b(1234, 567, 890);

cout << b;

a = b;

cout << a;

//работа с классом Date

Date c;

cin >> c;

cout << c;

//принцип подстановки

f1(c);//передаем объект класса Date

a = f2();//создаем в функции объект класса Date

cout << a;

a.changeSecond(12);

cout << a;

a.increaseThird();

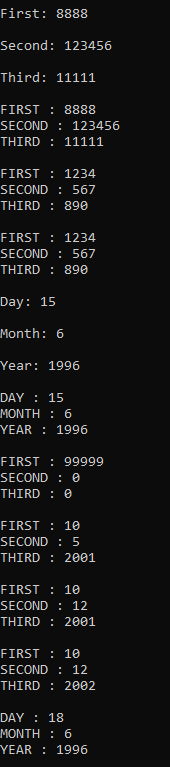
cout << a;

c.increaseDate(3);

cout << c;

}

Результаты работы программы



Ответы на контрольные вопросы

1. Для чего используется механизм наследования?

Наследование - это механизм получения нового класса на основе уже

существующего.

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?

Производный класс наследует описание базового класса; затем он может быть изменен добавлением новых членов, изменением существующих функций-членов и изменением прав доступа.

1. Наследуются ли конструкторы?

Конструкторы не наследуются.

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.

1. Могут ли быть виртуальными конструкторы? Деструкторы?

Конструкторы не могут быть виртуальными, в отличие от деструкторов.

1. 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.

1. Какое отношение устанавливает между классами открытое наследование?

Открытое наследование устанавливает между классами отношение «является»: класс-наследник является частью класса-родителя.

1. Какое отношение устанавливает между классами закрытое наследование?

Закрытое наследование – это наследование реализации.

1. В чем заключается принцип подстановки?

Везде, где может быть использован объект базового класса (при присваивании, при передаче параметров и возврате результата), вместо него разрешается использовать объект производного класса.

1. 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?

age, name, post, stage

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

Student::Student()

{

age = 0;

name = “”;

}

Employee::Employee() : Student()

{

age = 0;

name = “”;

post = “”;

}

Teacher::Teacher() : Employee()

{

age = 0;

name = “”;

post = “”;

stage = 0;

}

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

Student::Student(int A, string N)

{

age = A;

name = N;

}

Employee(int A, string N, string P)

{

age = A;

name = N;

post = P;

}

Teacher(int A, string N, string P, int S)

{

age = A;

name = N;

post = P;

stage = S;

}

1. 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)

{

age = E.age;

name = E.name;

post = E.post;

}

Teacher::Teacher(const Teacher& T);

{

age = T.age;

name = T.name;

post = T.post;

stage = T.stage;

}

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

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

{

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

age = s.age;

name = s.name;

return \*this;

}

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

{

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

age = e.age;

name = e.name;

post = e.post;

return \*this;

}

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

{

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

age = t.age;

name = t.name;

post = t.post;

stage = t.stage;

return \*this;

}