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: 09.03.04 - «Программная инженерия»

Лабораторная работа №4

По теме **«Простое наследование. Принцип подстановки»**

Вариант №10

Выполнял:

студент группы РИС-24-1б

Морозова Н.С.

Проверял:

Доцент кафедры ИТАС

Полякова О.А.

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.

Задание:

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.

Анализ задачи:

1. Описать в файле traid.h класс Traid, а в файле traid.cpp определить указанные методы.
2. Определить внутри класса функции изменения полей change() и прибавление 1 plus().
3. Создать производный класс Date в файле date.h и так же определить необходимые метеды для изменения полей, в файле date.cpp описать их.
4. Переопределить внутри класса функции изменения полей change() и прибавление 1 plus().
5. В файле Lab-4.cpp находится главная функция, где реализованы все методы.

Решение

Код

Файл traid.h

#pragma once

#include <iostream>

using namespace std;

class Traid

{

protected:

int first;

int second;

int third;

public:

//конструктор без параметров

Traid(void);

//констрктор с параметрами

Traid(int, int, int);

//конструктор копирования

Traid(const Traid&);

//деструктор

virtual ~Traid(void);

//селекторы

int Get\_first() { return first; }

int Get\_second() { return second; }

int Get\_third() { return third; }

//модификаторы

void Set\_first(int F) { first = F; }

void Set\_second(int S) { second = S; }

void Set\_third(int T) { third = T; }

//перегрузка операции присваивания

Traid& operator=(const Traid&);

//глобальные операторы-функции ввода-вывода

friend istream& operator>>(istream& in, Traid& c);

friend ostream& operator<<(ostream& out, const Traid& c);

//фунцкия изменения полей

void change() {

int n, dat;

cout << "Какое поле поменять? \n(first - 1, second - 2, third - 3) ";

cin >> n;

cout << "Введите значение: ";

cin >> dat;

switch (n)

{

case 1: first = dat; break;

case 2: second = dat; break;

case 3: third = dat; break;

default: cout << "Ошибка!"; break;

}

}

// прибавление 1

void plus() {

cout << "Прибавление 1";

first += 1;

second += 1;

third += 1;

}

};

Файл traid.cpp

#include "traid.h"

#include <iostream>

//конструктор без параметров

Traid::Traid(void){

first = 0;

second = 0;

third = 0;

}

//конструктор с параметрами

Traid::Traid(int F,int S,int T){

first = F;

second = S;

third = T;

}

//конструктор копирования

Traid::Traid(const Traid& traid){

first = traid.first;

second = traid.second;

third = traid.third;

}

//деструктор

Traid::~Traid(void) { cout << "Деструктор" << endl; }

//перегрузка операции присваивания

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

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

first = t.first;

third = t.third;

second = t.second;

return \*this;

}

//глобальная функция для ввода

istream& operator>>(istream& in,Traid& t){

cout << "first: ";

in >> t.first;

cout << "second: ";

in >> t.second;

cout << "third: ";

in >> t.third;

return in;

}

//глобальная функция для вывода

ostream& operator<<(ostream& out, const Traid& t){

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

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

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

out << endl;

return out;

}

Файл date.h

#pragma once

#include "traid.h"

#include <iostream>

using namespace std;

class Date: public Traid

{

protected:

int year;

int month;

int day;

public:

Date(void);

Date(int, int, int);

Date(const Date&);

~Date(void);

//селекторы

int Get\_year() { return year; }

int Get\_month() { return month; }

int Get\_day() { return day; }

//модификаторы

void Set\_year(int Y) { year = Y; }

void Set\_month(int M) { month = M; }

void Set\_day(int D) { day = D; }

Date& operator=(const Date&);

friend istream& operator>>(istream& in, Date& c);

friend ostream& operator<<(ostream& out, const Date& c);

//фунцкия изменения полей

void change(){

int n;

do

{

cout << "Сколько дней прибавить к дате? ";

cin >> n;

} while (n > 30);

day += n;

}

// прибавление 1

void plus() {

cout << "Прибавление 1";

if (day + 1 == 32)

{

month += 1;

day = 1;

}

else day += 1;

if (month + 1 == 13)

{

month = 1;

year += 1;

}

else month += 1;

year += 1;

}

};

Файл date.cpp

#include "traid.h"

#include "date.h"

#include <iostream>

Date::Date(void){

year = 0;

month = 0;

day = 0;

}

//конструктор с параметрами

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

year = Y;

month = M;

month = D;

}

//конструктор копирования

Date::Date(const Date& date){

year = date.year;

month = date.month;

day = date.day;

}

Date::~Date(void) { cout << "Деструктор" << endl; }

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

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

year = d.year;

month = d.month;

day = d.day;

return \*this;

}

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

cout << "Год: ";

in >> d.year;

cout << "Месяц: ";

in >> d.month;

cout << "День: ";

in >> d.day;

return in;

}

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

out << "\nГод: " << d.year;

out << "\nМесяц: " << d.month;

out << "\nДень: " << d.day;

out << endl;

return out;

}

Файл Lab-4.cpp

#include <iostream>

#include <Windows.h>

#include "traid.h"

#include "date.h"

using namespace std;

int main(){

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

Traid tr1;

cin >> tr1;

cout << tr1;

tr1.change();

cout << tr1 << endl;

tr1.plus();

cout << tr1;

Traid tr2(11, 22, 33);

cout << tr2;

tr1 = tr2;

cout << tr2;

Date d1;

cout << d1 << endl;

cin >> d1;

cout << d1;

d1.change();

cout << d1 << endl;

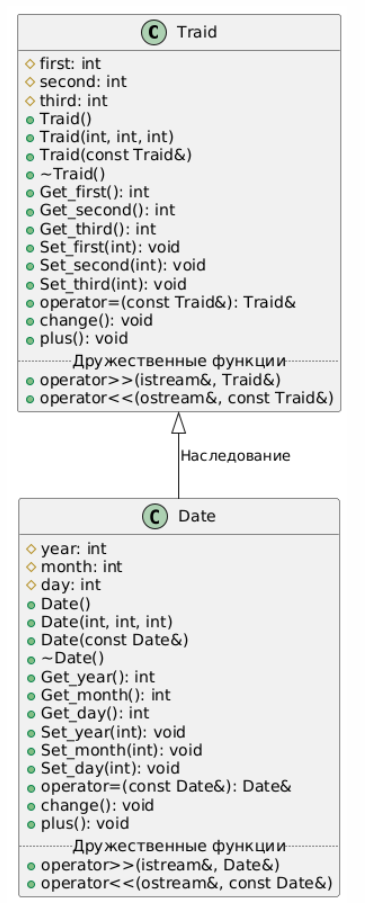
d1.plus();

cout << d1 << endl;

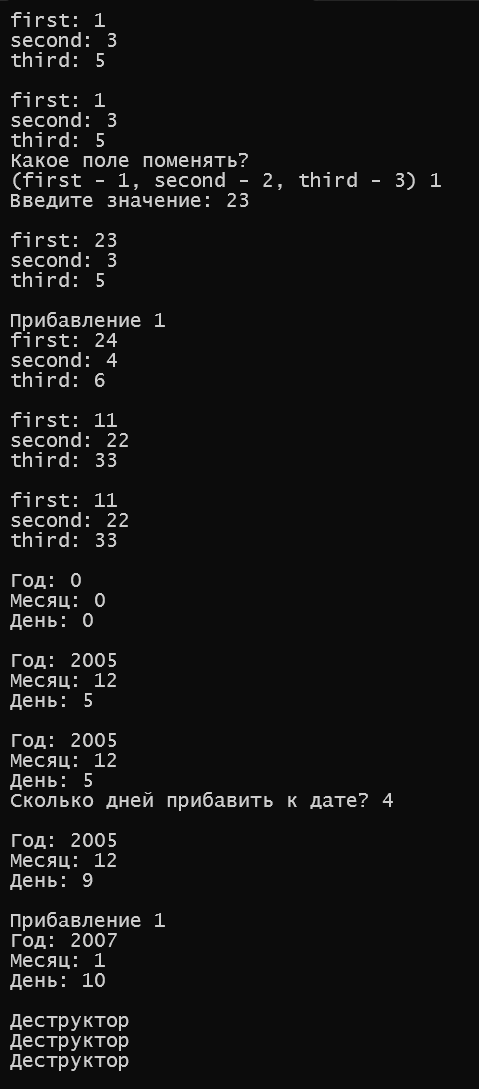
return 0;

}

UML-диаграмма



Результат работы программы:



Контрольные вопросы:

1. Для чего используется механизм наследования?

Наследование позволяет создавать новый класс (производный) на основе существующего (базового), повторно используя его код и расширяя функциональность. Это способствует уменьшению дублирования кода и организации иерархии классов.

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

Компоненты с public доступны из любого места программы: внутри класса, в производных классах и извне (через объекты).

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

Компоненты с private доступны только внутри самого класса. Они скрыты от внешнего кода и производных классов.

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

Компоненты с protected доступны внутри класса и в производных классах, но не доступны извне (через объекты).

1. Каким образом описывается производный класс?

Пример:

class Base { ... };

class Derived : public Base { ... }; // public - наследование

1. Наследуются ли конструкторы?

Конструкторы не наследуются. Производный класс должен определять свои конструкторы, но может вызывать конструктор базового класса.

1. Наследуются ли деструкторы?

Деструкторы не наследуются в явном виде, но при уничтожении объекта производного класса автоматически вызывается деструктор базового класса (после деструктора производного).

1. В каком порядке конструируются объекты производных классов?

Сначала вызываются конструкторы базовых классов (в порядке их объявления), затем конструкторы членов класса, и только потом конструктор производного класса.

1. В каком порядке уничтожаются объекты производных классов?

В обратном порядке: сначала деструктор производного класса, затем деструкторы членов класса, и наконец деструкторы базовых классов.

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

Виртуальные функции позволяют переопределять методы в производных классах. Позднее связывание означает, что выбор версии функции происходит во время выполнения на основе типа объекта.

1. Могут ли быть виртуальными конструкторы? Деструкторы?

Конструкторы не могут быть виртуальными.

Деструкторы могут (и часто должны) быть виртуальными, особенно в базовых классах, чтобы корректно вызывать деструкторы производных классов.

1. Наследуется ли спецификатор virtual?

Да, если функция объявлена виртуальной в базовом классе, она остаётся виртуальной во всех производных классах, даже без явного указания virtual.

1. Какое отношение устанавливает между классами открытое наследование?

Открытое наследование моделирует отношение "является" (is-a). Публичные и защищённые члены базового класса сохраняют свою доступность в производном классе.

1. Какое отношение устанавливает между классами закрытое наследование?

Закрытое наследование моделирует отношение "реализовано посредством" (has-a). Все члены базового класса становятся private в производном классе и недоступны извне.

1. В чем заключается принцип подстановки?

Объекты производного класса должны быть способны заменить объекты базового класса без изменения корректности программы. То есть, производный класс должен соблюдать контракт базового класса, не нарушая его логики.

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?

Объект x (типа Teacher) будет содержать следующие данные:

Унаследованные от Student:

* int age (но они private, поэтому недоступны напрямую в Teacher)
* string name (public)

Унаследованные от Employee:

* string post (protected)

Собственные:

* int stage (protected)

В коде age и name объявлены после public:, но если они находятся вне секций private/protected/public, то в C++ их видимость зависит от типа класса (class по умолчанию — private, struct — public).

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

class Student {

int age;

public:

string name;

Student() : age(0), name("") {} // Конструктор без параметров

};

class Employee : public Student {

protected:

string post;

public:

Employee() : Student(), post("") {} // Вызов конструктора Student

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher() : Employee(), stage(0) {} // Вызов конструктора Employee

};

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

class Student {

int age;

public:

string name;

Student(int a, const string& n) : age(a), name(n) {}

};

class Employee : public Student {

protected:

string post;

public:

Employee(int a, const string& n, const string& p) : Student(a, n), post(p) {}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher(int a, const string& n, const string& p, int s) : Employee(a, n, p), stage(s) {}

};

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

class Student {

int age;

public:

string name;

Student(const Student& other) : age(other.age), name(other.name) {}

};

class Employee : public Student {

protected:

string post;

public:

Employee(const Employee& other) : Student(other), post(other.post) {}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher(const Teacher& other) : Employee(other), stage(other.stage) {}

};

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

class Student {

int age;

public:

string name;

Student& operator=(const Student& other)

{

if (this != &other)

{

age = other.age;

name = other.name;

}

return \*this;

}

};

class Employee : public Student {

protected:

string post;

public:

Employee& operator=(const Employee& other)

{

if (this != &other)

{

Student::operator=(other); // Вызов присваивания для Student

post = other.post;

}

return \*this;

}

};

class Teacher : public Employee {

protected:

int stage;

public:

Teacher& operator=(const Teacher& other) {

if (this != &other) {

Employee::operator=(other); // Вызов присваивания для Employee

stage = other.stage;

}

return \*this;

}

};