Пермский Национальный Исследовательский  
Политехнический Университет

**Лабораторная работа № 4**

Информатика

за 2 семестр

Вариант № 1

Выполнил:

Студент группы РИС 20-1-бз

Акиев В.В.

20-ЭТФ-641

Проверила:

Доцент кафедры ИТАС

Полякова О.А.

2021

**Простое наследование. Принцип подстановки.**

1. **Цель задания**.
2. Создание консольного приложения, состоящего из нескольких файлов в системе программирования VisualStudio.
3. Создание иерархии классов с использованием простого наследования.
4. Изучение принципа подстановки.
5. **Задание.**

Базовый класс:

ПАРА\_ЧИСЕЛ(PAIR)

Первое\_число (first) - int

Второе\_число(second)- int

Определить методы изменения полей и сравнения пар (пара p1 больше пары р2, если(p1.first>p2.first)||(p1.first==p2.first&&p1.second>p2.second).

Создать производный класс ДРОБЬ (FRACTION), с полями

Целая\_часть\_числа и Дробная\_часть\_числа.

Определит ьполный набор методов сравнения.

1. **Лабораторная\_4.cpp**

#include <iostream>

#include "Pair.h"

#include "Fraction.h"

using namespace std;

void main()

{

Pair p1, p2;

cin >> p1;

cout << p1;

cin >> p2;

cout << p2;

string str = p1 > p2 ? "true" : "false";

cout << "p1 > p2: " << str << "\n";

str = p1 >= p2 ? "true" : "false";

cout << "p1 >= p2: " << str << "\n";

str = p1 < p2 ? "true" : "false";

cout << "p1 < p2: " << str << "\n";

str = p1<= p2 ? "true" : "false";

cout << "p1<= p2: " << str << "\n";

str = p1 == p2 ? "true" : "false";

cout << "p1 == p2: " << str << "\n";

str = p1 != p2 ? "true" : "false";

cout << "p1 != p2: " << str << "\n";

int f = p1.Get\_first();

int s = p1.Get\_second();

f = f + ++s;

p1.Set\_first(s);

p1.Set\_second(f);

cout << p1;

Fraction fr1,fr2;

cin >> fr1;

cout << fr1;

cin >> fr2;

cout << fr2;

str = fr1 > fr2 ? "true" : "false";

cout << "fr1 > fr2: " << str << "\n";

str = fr1 >= fr2 ? "true" : "false";

cout << "fr1 >= fr2: " << str << "\n";

str = fr1 < fr2 ? "true" : "false";

cout << "fr1 < fr2: " << str << "\n";

str = fr1 <= fr2 ? "true" : "false";

cout << "fr1<= fr2: " << str << "\n";

str = fr1 == fr2 ? "true" : "false";

cout << "fr1 == fr2: " << str << "\n";

str = fr1 != fr2 ? "true" : "false";

cout << "fr1 != fr2: " << str << "\n";

f = fr1.Get\_first();

s = fr1.Get\_second();

p1.Set\_first(s);

p1.Set\_second(f);

f = f + ++s;

Fraction fr3;

fr3 = Fraction(f,s,f/(double)s);

cout << fr3;

}

1. **Fraction.h**

#pragma once

#include "Pair.h";

class Fraction:

public Pair

{

protected:

//атрибуты

int intPart;

int fractPart;

public:

//деконструктор

virtual ~Fraction();

//конструкторы

Fraction(void);

Fraction(int, int, int, int);

Fraction(int, int, double);

Fraction(const Fraction&);

//селекторы

int Get\_intPart() { return intPart; }

int Get\_fractPart() { return fractPart; }

//модификаторы

void Set\_intPart(int);

void Set\_fractPart(int);

void Set\_All(double);

//перегрузка присваивания

Fraction& operator=(const Fraction&);

//перегрузка операторов сравнения

friend bool operator> (const Fraction& p1, const Fraction& p2);

friend bool operator<= (const Fraction& p1, const Fraction& p2);

friend bool operator< (const Fraction& p1, const Fraction& p2);

friend bool operator>= (const Fraction& p1, const Fraction& p2);

friend bool operator== (const Fraction& p1, const Fraction& p2);

friend bool operator!= (const Fraction& p1, const Fraction& p2);

//перегрузка ввода вывода

friend istream& operator>> (istream& in, Fraction& p);

friend ostream& operator<< (ostream& out, const Fraction& p);

};

1. **Fraction.cpp**

#include "Fraction.h"

#include <math.h>;

#include <tchar.h>

#include <iostream>

#include <iomanip>

#include <sstream>

int fractional\_part\_as\_int(double number) {

int number\_of\_decimal\_places;

std::stringstream ss;

ss << std::setprecision(9) << number;

std::string strNum = ss.str();

size\_t pos = strNum.find('.');

if (pos != strNum.npos)

number\_of\_decimal\_places = strNum.size() - 1 - pos;

else

number\_of\_decimal\_places = 0;

double dummy;

double frac = modf(number, &dummy);

return round(frac \* pow(10, number\_of\_decimal\_places));

}

//не параметризованный конструктор

Fraction::Fraction(void):Pair()

{

intPart = 0;

fractPart = 0;

}

//деструктор

Fraction::~Fraction()

{

}

//параметризованный конструктор

Fraction::Fraction(int f, int s, int ip, int fp):Pair(f,s)

{

intPart = ip;

fractPart = fp;

}Fraction::Fraction(int f, int s, double num) : Pair(f, s)

{

intPart = (int)num;

fractPart = fractional\_part\_as\_int(num);

}

//клонирование

Fraction::Fraction(const Fraction& Fraction)

{

first = Fraction.first;

second = Fraction.second;

intPart = Fraction.intPart;

fractPart = Fraction.fractPart;

}

//модификатор first

void Fraction::Set\_intPart(int i)

{

intPart = i;

}

//модификатор second

void Fraction::Set\_fractPart(int f)

{

fractPart = f;

}

//оператор присваивания

Fraction& Fraction::operator=(const Fraction& Fraction)

{

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

first = Fraction.first;

second = Fraction.second;

intPart = Fraction.intPart;

fractPart = Fraction.fractPart;

return \*this;

}

//операторы сравнения

bool operator==(const Fraction& f1, const Fraction& f2)

{

return ((f1.first == f2.first && f1.second == f2.second) && (f1.intPart == f2.intPart && f1.fractPart == f2.fractPart));

}

bool operator!=(const Fraction& f1, const Fraction& f2)

{

return !(f1 == f2);

}

bool operator>(const Fraction& f1, const Fraction& f2)

{

return ((f1.first > f2.first) || (f1.first == f2.first && f1.second > f2.second)) && ((f1.intPart > f2.intPart) || (f1.intPart == f2.intPart && f1.fractPart > f2.fractPart)) ;

}

bool operator>=(const Fraction& f1, const Fraction& f2)

{

return (f1 > f2) || (f1 == f2);

}

bool operator<(const Fraction& f1, const Fraction& f2)

{

return ((f1.first < f2.first) || (f1.first == f2.first && f1.second < f2.second)) && ((f1.intPart < f2.intPart) || (f1.intPart == f2.intPart && f1.fractPart < f2.fractPart));

}

bool operator<=(const Fraction& f1, const Fraction& f2)

{

return (f1 < f2) || (f1 == f2);

}

//оператор ввода

istream& operator>> (istream& in, Fraction& Fraction)

{

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

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

cout << "\nInt part: "; in >> Fraction.intPart ;

cout << "\nFraction part: "; in >> Fraction.fractPart;

return in;

}

//оператор вывода

ostream& operator<< (ostream& out, const Fraction& Fraction)

{

out << "\nFirst: " << Fraction.first;

out << "\nSecond: " << Fraction.second;

out << "\nFraction: " << Fraction.intPart << "."<< Fraction.fractPart;

out << "\n";

return out;

}

1. **Pair.h**

#pragma once

#include <iostream>

using namespace std;

class Pair

{

protected:

//атрибуты

int first;

int second;

public:

//деконструктор

virtual ~Pair();

//конструкторы

Pair(void);

Pair(int, int);

Pair(const Pair&);

//селекторы

int Get\_first() { return first; }

int Get\_second() { return second; }

//модификаторы

void Set\_first(int);

void Set\_second(int);

//перегрузка присваивания

Pair& operator=(const Pair&);

//перегрузка операторов сравнения

friend bool operator> (const Pair& p1, const Pair& p2);

friend bool operator<= (const Pair& p1, const Pair& p2);

friend bool operator< (const Pair& p1, const Pair& p2);

friend bool operator>= (const Pair& p1, const Pair& p2);

friend bool operator== (const Pair& p1, const Pair& p2);

friend bool operator!= (const Pair& p1, const Pair& p2);

//перегрузка ввода вывода

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

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

};

1. **Pair.cpp**

#include "Pair.h"

//не параметризованный конструктор

Pair::Pair(void)

{

first = 0;

second = 0;

}

//деструктор

Pair::~Pair()

{

}

//параметризованный конструктор

Pair::Pair(int f,int s)

{

first = f;

second = s;

}

//клонирование

Pair::Pair(const Pair& pair)

{

first = pair.first;

second = pair.second;

}

//модификатор first

void Pair::Set\_first(int f)

{

first = f;

}

//модификатор second

void Pair::Set\_second(int s)

{

second = s;

}

//оператор присваивания

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

{

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

first = pair.first;

second = pair.second;

return \*this;

}

//операторы сравнения

bool operator==(const Pair&p1, const Pair&p2)

{

return (p1.first == p2.first && p1.second == p2.second);

}

bool operator!=(const Pair& p1, const Pair& p2)

{

return !(p1 == p2);

}

bool operator>(const Pair& p1, const Pair& p2)

{

return (p1.first > p2.first) || (p1.first == p2.first && p1.second > p2.second);

}

bool operator>=(const Pair& p1, const Pair& p2)

{

return (p1 > p2) || (p1 == p2);

}

bool operator<(const Pair& p1, const Pair& p2)

{

return (p1.first < p2.first) || (p1.first == p2.first && p1.second < p2.second);

}

bool operator<=(const Pair& p1, const Pair& p2)

{

return (p1 < p2) || (p1 == p2);

}

//оператор ввода

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

{

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

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

return in;

}

//оператор вывода

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

{

out << "\nFirst: " << pair.first;

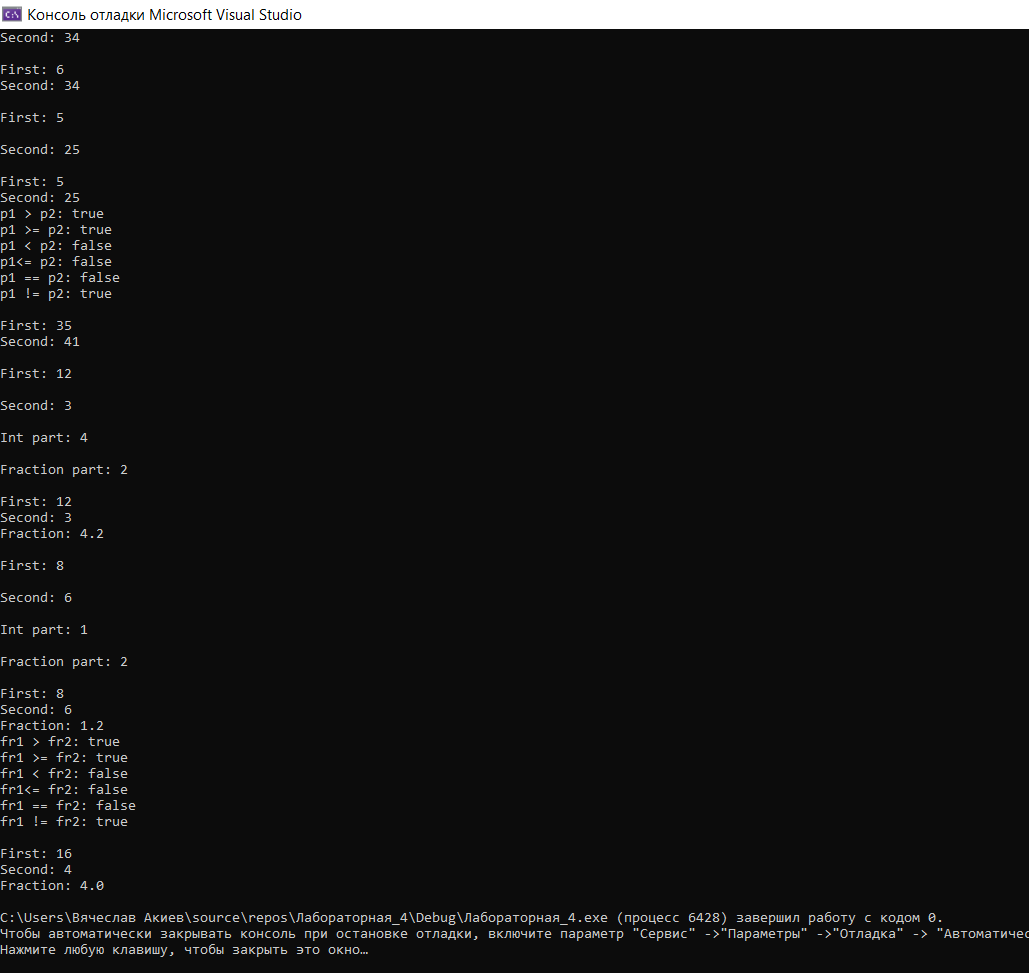
out << "\nSecond: " << pair.second;

out << "\n";

return out;

}

1. **Результат работы программы**

****

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

Для структурирования и повторно использования уже написанного кода, что,

в свою очередь, может значительно ускорить процесс разработки программ

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

Данные наследуются без изменения уровня доступа к ним

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

Все унаследованные данные становятся private

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

Все унаследованные данные становятся protected

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

К названию класса добавляются: после которого перечисляются наследуемые классы.

Описывать наследуемые переменные не нужно.

* 1. **Наследуютсяликонструкторы?**

Конструкторы наследуются

* 1. **Наследуютсялидеструкторы?**

Деструкторы наследуются

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

Сначала вызывается конструктор базового класса, а затем конструктор производного класса.

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

Сначала вызывается деструктор производного класса, а затем деструктор базового класса.

* 1. **Что представляют собой виртуальные функциии механизма позднего связывания?**

Виртуальные функции – функции связывание вызова которой с ее определениемпроисходит во время выполнения программы (то есть динамически).

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

Конструкторы не могут быть виртуальными, деструкторы могут.

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

Виртуальные спецификаторы наследуются

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

При открытом наследование между классами устанавливается отношение "Класс-наследник является частью класса-родителя"

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

Закрытое наследование - наследование реализации (при наследовании заново реализуются все методы)

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

При соблюдении этого принципа разрешается производным классам использовать объекты базового класса без реализации

* 1. **Имеется иерархия классов:**

stringname

* 1. **Для классов Student, Employee и Teacher написать конструкторы без параметров.**
  2. **Для классов Student, Employee и Teacher написать конструкторы с параметрами.**
  3. **Для классов Student, Employee и Teacher написать конструкторы копирования.**
  4. **Для классов Student, Employee и Teacher определить операцию присваивания.**

classStudent

{

public:

Student();

Student(int, string );

Student(constStudent&);

string name;

Student&operator= (constStudent&);

protected:

int age;

};

Student::Student(void)

{

age = 0;

name ="";

}

Student::Student(inta, stringn)

{

age = a;

name =n;

}

Student::Student(constStudent&s)

{

age = s.age;

name =s.name;

}

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

{

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

age = s.age;

name =s.name;

return \*this;

}

classEmployee: publicStudent

{

public:

Employee();

Employee(int, string, string);

Employee(constEmployee&, constStudent&);

Employee&operator= (constEmployee&);

protected:

string post;

};

Employee::Employee()

{

age = 0;

name ="";

post ="";

}

Employee::Employee(inta, stringn, stringp) : Student(a, n)

{

post =p;

}

Employee::Employee(constEmployee&e)

{

age = e.age;

name =e.name;

post =e.post;

}

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

{

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

age = e.age;

name =e.name;

post =e.post;

return \*this;

}

classTeacher:publicEmployee

{

public:

Teacher();

Teacher(int, string, string, int);

Teacher(constTeacher&);

Employee&operator= (constTeacher&);

protected:

int stage;

};

Teacher::Teacher()

{

age = 0;

name ="";

post ="";

stage = 0;

}

Teacher::Teacher(inta, stringn, stringp, ints) : Employee(a, n, p)

{

age = a;

name =n;

post =p;

stage = s;

}

Teacher::Teacher(constTeacher&t)

{

age = t.age;

name =t.name;

post =t.post;

stage = t.stage;

}

Employee&Teacher::operator=(constTeacher&t)

{

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

age = t.age;

name =t.name;

post =t.post;

stage = t.stage;

return \*this;

}