Пермский Национальный Исследовательский  
Политехнический Университет

**Лабораторная работа № 4**

Информатика

за 2 семестр

Вариант № 1

Выполнил:

Студент группы РИС 20-1-бз

КургановН.В.

20-ЭТФ-631

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. Созданиеиерархииклассовсиспользованиемпростогонаследования.
4. Изучениепринципаподстановки.
5. **Задание.**

Базовыйкласс:

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 иДробная\_часть\_числа.

Определитьполныйнаборметодовсравнения.

1. **Laba4\_main.cpp**

#include<iostream>

#include"Pair.h"

#include"Fraction.h"

usingnamespace std;

voidmain()

{

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

#pragmaonce

#include"Pair.h";

classFraction:

publicPair

{

protected:

//атрибуты

intintPart;

intfractPart;

public:

//деконструктор

virtual ~Fraction();

//конструкторы

Fraction(void);

Fraction(int, int, int, int);

Fraction(int, int, double);

Fraction(constFraction&);

//селекторы

intGet\_intPart() { returnintPart; }

intGet\_fractPart() { returnfractPart; }

//модификаторы

voidSet\_intPart(int);

voidSet\_fractPart(int);

voidSet\_All(double);

//перегрузкаприсваивания

Fraction&operator=(constFraction&);

//перегрузкаоператоровсравнения

friendbooloperator> (constFraction&p1, constFraction&p2);

friendbooloperator<= (constFraction&p1, constFraction&p2);

friendbooloperator< (constFraction&p1, constFraction&p2);

friendbooloperator>= (constFraction&p1, constFraction&p2);

friendbooloperator== (constFraction&p1, constFraction&p2);

friendbooloperator!= (constFraction&p1, constFraction&p2);

//перегрузкавводавывода

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

friendostream&operator<< (ostream&out, constFraction&p);

};

1. **Fraction.cpp**

#include"Fraction.h"

#include<math.h>;

#include<tchar.h>

#include<iostream>

#include<iomanip>

#include<sstream>

intfractional\_part\_as\_int(doublenumber) {

intnumber\_of\_decimal\_places;

std::stringstream ss;

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

std::stringstrNum = 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);

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

}

//не параметризованный конструктор

Fraction::Fraction(void):Pair()

{

intPart = 0;

fractPart = 0;

}

//деструктор

Fraction::~Fraction()

{

}

//параметризованныйконструктор

Fraction::Fraction(intf, ints, intip, intfp):Pair(f,s)

{

intPart = ip;

fractPart = fp;

}Fraction::Fraction(intf, ints, doublenum) : Pair(f, s)

{

intPart = (int)num;

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

}

//клонирование

Fraction::Fraction(constFraction&Fraction)

{

first = Fraction.first;

second = Fraction.second;

intPart = Fraction.intPart;

fractPart = Fraction.fractPart;

}

//модификатор first

voidFraction::Set\_intPart(inti)

{

intPart = i;

}

//модификатор second

voidFraction::Set\_fractPart(intf)

{

fractPart = f;

}

//оператор присваивания

Fraction&Fraction::operator=(constFraction&Fraction)

{

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

first = Fraction.first;

second = Fraction.second;

intPart = Fraction.intPart;

fractPart = Fraction.fractPart;

return \*this;

}

//операторысравнения

booloperator==(constFraction&f1, constFraction&f2)

{

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

}

booloperator!=(constFraction&f1, constFraction&f2)

{

return !(f1==f2);

}

booloperator>(constFraction&f1, constFraction&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)) ;

}

booloperator>=(constFraction&f1, constFraction&f2)

{

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

}

booloperator<(constFraction&f1, constFraction&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));

}

booloperator<=(constFraction&f1, constFraction&f2)

{

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

}

//операторввода

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;

returnin;

}

//операторвывода

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

{

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

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

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

out<<"\n";

returnout;

}

1. Pair.h

#pragmaonce

#include<iostream>

usingnamespace std;

classPair

{

protected:

//атрибуты

int first;

int second;

public:

//деконструктор

virtual ~Pair();

//конструкторы

Pair(void);

Pair(int, int);

Pair(constPair&);

//селекторы

intGet\_first() { return first; }

intGet\_second() { return second; }

//модификаторы

voidSet\_first(int);

voidSet\_second(int);

//перегрузкаприсваивания

Pair&operator=(constPair&);

//перегрузка операторов сравнения

friendbooloperator> (constPair&p1, constPair&p2);

friendbooloperator<= (constPair&p1, constPair&p2);

friendbooloperator< (constPair&p1, constPair&p2);

friendbooloperator>= (constPair&p1, constPair&p2);

friendbooloperator== (constPair&p1, constPair&p2);

friendbooloperator!= (constPair&p1, constPair&p2);

//перегрузкавводавывода

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

friendostream&operator<< (ostream&out, constPair&p);

};

1. Pair.cpp

#include"Pair.h"

//не параметризованный конструктор

Pair::Pair(void)

{

first = 0;

second = 0;

}

//деструктор

Pair::~Pair()

{

}

//параметризованныйконструктор

Pair::Pair(intf,ints)

{

first = f;

second = s;

}

//клонирование

Pair::Pair(constPair&pair)

{

first = pair.first;

second = pair.second;

}

//модификатор first

voidPair::Set\_first(intf)

{

first = f;

}

//модификатор second

voidPair::Set\_second(ints)

{

second = s;

}

//операторприсваивания

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

{

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

first = pair.first;

second = pair.second;

return \*this;

}

//операторысравнения

booloperator==(constPair&p1, constPair&p2)

{

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

}

booloperator!=(constPair&p1, constPair&p2)

{

return !(p1==p2);

}

booloperator>(constPair&p1, constPair&p2)

{

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

}

booloperator>=(constPair&p1, constPair&p2)

{

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

}

booloperator<(constPair&p1, constPair&p2)

{

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

}

booloperator<=(constPair&p1, constPair&p2)

{

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

}

//операторввода

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

{

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

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

returnin;

}

//операторвывода

ostream&operator<< (ostream&out, constPair&pair)

{

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

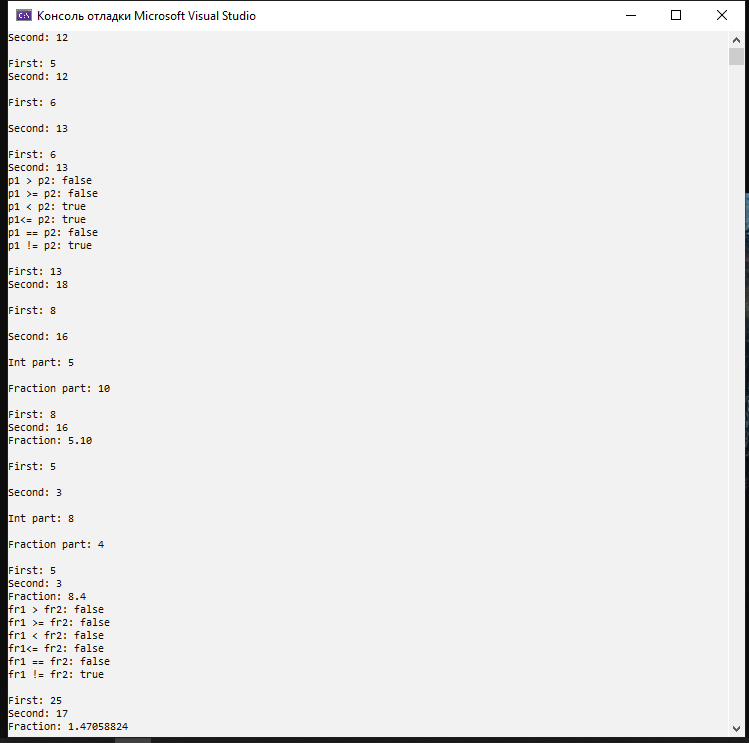
out<<"\nSecond: "<<pair.second;

out<<"\n";

returnout;

}

1. **Результат работы программы**

****

1. **Контрольные вопросы**
   1. Длячегоиспользуетсямеханизмнаследования?

*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. Какимобразомописывается производныйкласс?

К названию класса добавляются: после которого перечисляются наследуемые классы.

Описывать наследуемые переменные не нужно.

* 1. Наследуютсяликонструкторы?

Конструкторы наследуются

* 1. Наследуютсялидеструкторы?

Деструкторы наследуются

* 1. В какомпорядке конструируютсяобъектыпроизводныхклассов?

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

* 1. Вкакомпорядкеуничтожаютсяобъектыпроизводныхклассов?

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

* 1. Что представляютсобойвиртуальныефункцииимеханизмпозднегосвязывания?

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

* 1. Могутлибытьвиртуальнымиконструкторы?Деструкторы?

Конструкторы не могут быть виртуальными, деструкторы могут.

* 1. Наследуется ли спецификатор virtual?

Виртуальные спецификаторы наследуются

* 1. Какоеотношениеустанавливаетмеждуклассамиоткрытое наследование?

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

* 1. Какоеотношениеустанавливаетмеждуклассамизакрытоенаследование?

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

* 1. Вчем заключаетсяпринцип 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;

}