Лабораторная работа 5.

Программная реализация отношений простого наследования

Задача:

1. Создайте производный класс для АТД, реализованного по заданию лабораторной работы 3, используя одиночное наследование.

2. Проверьте работоспособность АТД и производного класса на тестовом наборе данных.

#include <iostream>

#include <string>

#include <clocale>

using namespace std;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ВТОРОЙ КЛАСС\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

template <class T, class F>

class Voice

{

private:

T modelNumber;

F power;

char\* name;

bool unit;

public:

Voice();

Voice(T a, F b, char\* c, bool d);

Voice(const Voice<T, F>& a);

~Voice();

void Info();

Voice <T, F> sum(const Voice<T, F>& a);

bool Cmp(const Voice<T, F>& a);

void Copy(const Voice<T, F>& a);

Voice<T, F>operator =(const Voice<T, F>& a);

Voice<T, F>operator +(const Voice<T, F>& a);

bool operator >(const Voice<T, F>& a);

bool operator <(const Voice<T, F>& a);

bool operator ==(const Voice<T, F>& a);

bool operator !=(const Voice<T, F>& a);

bool operator >=(const Voice<T, F>& a);

bool operator <=(const Voice<T, F>& a);

};

template <class T, class F>

void Voice<T, F> ::Info()

{

cout << "Название: " << name;

cout << "Номер модели звукового устройства: " << modelNumber << endl;

cout << "Мощность: " << power << "у.е." << endl;

if (unit)

{

cout << "Улучшенное устройство";

}

else

{

cout << "Стоковое устройство";

}

cout << endl;

}

template <class T, class F>

Voice<T, F> ::Voice()

{

modelNumber = (T)0;

power = (F)0;

unit = false;

name = new char[1];

strcpy(name, "");

}

template <class T, class F>

Voice<T, F> ::Voice(T t, F f, char\* n, bool u)

{

modelNumber = (T)t;

power = (F)f;

unit = u;

name = new char[strlen(n) + 1];

strcpy(name, n);

}

template <class T, class F>

Voice<T, F> ::Voice(const Voice<T, F>& v)

{

if (&v != this)

{

modelNumber = v.modelNumber;

power = v.power;

name = new char[strlen(v.name) + 1];

strcpy(name, v.name);

unit = v.unit;

}

}

template <class T, class F>

Voice<T, F> ::~Voice()

{

delete[] name;

}

template <class T, class F>

bool Voice<T, F> ::Cmp(const Voice<T, F>& a)

{

if (power == a.power && modelNumber == a.modelNumber && unit == a.unit && name == a.name)

return true;

else

return false;

}

template <class T, class F>

void Voice<T, F> ::Copy(const Voice<T, F>& a)

{

modelNumber = a.modelNumber;

power = a.power;

name = new char[strlen(a.name) + 1];

strcpy(name, a.name);

unit = a.unit;

}

template <class T, class F>

Voice<T, F> Voice<T, F> ::sum(const Voice<T, F>& a)

{

Voice<T, F> f;

char\* tmp = new char[strlen(name) + 1];

strcpy(tmp, name);

f.modelNumber = modelNumber + a.modelNumber;

f.power = power + a.power;

f.unit = unit + a.unit;

strcat(tmp, "-");

strcat(tmp, a.name);

f.name = new char[strlen(tmp) + 1];

strcpy(f.name, tmp);

return f;

}

template <class T, class F>

Voice<T, F> Voice<T, F> :: operator=(const Voice<T, F>& a)

{

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

else

{

this->Copy(a);

return \*this;

}

}

template <class T, class F>

Voice<T, F> Voice<T, F> :: operator+(const Voice<T, F>& a)

{

Voice f = this->sum(a);

return f;

}

template <class T, class F>

bool Voice<T, F> :: operator<(const Voice<T, F>& a)

{

return power < a.power && modelNumber < a.modelNumber;

}

template <class T, class F>

bool Voice<T, F> :: operator>(const Voice<T, F>& a)

{

return power > a.power&& modelNumber > a.modelNumber;

}

template <class T, class F>

bool Voice<T, F> :: operator==(const Voice<T, F>& a)

{

return power == a.power && modelNumber == a.modelNumber && name == a.name && unit == a.unit;

}

template <class T, class F>

bool Voice<T, F> :: operator!=(const Voice<T, F>& a)

{

return power != a.power || modelNumber != a.modelNumber || name != a.name || unit == a.unit;

}

template <class T, class F>

bool Voice<T, F> :: operator<=(const Voice<T, F>& a)

{

return (power < a.power && modelNumber < a.modelNumber) || (power != a.power || modelNumber != a.modelNumber || name != a.name || unit == a.unit);

}

template <class T, class F>

bool Voice<T, F> :: operator>=(const Voice<T, F>& a)

{

return (power > a.power&& modelNumber > a.modelNumber) || (power != a.power || modelNumber != a.modelNumber || name != a.name || unit == a.unit);

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_наследование\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

template <class T>

class pillow

{

protected:

T soft;

char\* color;

public:

pillow();

pillow(T s, char\* c);

~pillow();

void virtual print();

void refill();

void use();

};

template <class T>

pillow<T>::pillow()

{

soft = (T)0;

color = new char[7];

strcpy(color, "white");

}

template <class T>

pillow<T>::pillow(T s, char\* c)

{

soft = (T)s;

color = new char[strlen(c) + 1];

strcpy(color, c);

}

template <class T>

pillow<T> ::~pillow()

{

soft = (T)0;

//color = (char\*)"";

delete[] color;

}

template <class T>

void pillow<T>::print()

{

cout << "наполнение: " << soft << endl;

cout << "название: " << color << endl << endl;

}

template <class T>

void pillow<T> ::refill()

{

soft += (T)1;

}

template <class T>

void pillow<T>::use()

{

if (soft != (T)0)

{

soft -= (T)1;

}

}

template <class T>

class fan : pillow<T>

{

private:

double rounds;

public:

void virtual print();

fan();

void refill();

void use();

fan(T s, char\* c, double r);

void speedUp();

};

template <class T>

void fan<T>::refill()

{

pillow<T>::refill();

}

template <class T>

void fan<T>::use()

{

pillow::use();

}

template <class T>

fan<T>::fan(T s, char\* c, double r) : pillow<T>::pillow(s, c)

{

rounds = r;

}

template <class T>

fan<T>::fan() : pillow<T>::pillow()

{

rounds = 1;

}

template <class T>

void fan<T>::print()

{

pillow<T>::print();

/\*cout << "наполнение: " << soft << endl;

cout << "название: " << color << endl;\*/

cout << "оборотов в минуту: " << rounds << endl << endl;

}

template <class T>

void fan<T>::speedUp()

{

rounds += 10;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_МЭЙН\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMM

int main()

{

setlocale(LC\_ALL, "rus");

pillow<int> c1(), c2(10, (char\*)"white");

fan<int> f1(), f2(30, (char\*)"another", 22.5);

c2.print();

f2.print();

f2.refill();

f2.speedUp();

f2.print();

system("pause");

return 0;

}

1. **Дайте определение наследования.**

Наследование (inheritance) – это механизм получения нового класса на основе существующего класса. Существующий класс может быть дополнен или изменен для создания производного класса. Наследование осуществляется с помощью конструкции:

class имя\_класса:

public|protected|private)opt имя\_базового\_класса

{

объявления членов

};

1. **Какие модификаторы прав доступа к производному классу вы знаете и в чем их особенности?**

Модификаторы прав доступа к членам класса (public, protected и private) могут применяться в объявлении класса в любом порядке и сколько угодно раз. Открытый член доступен во всей области видимости, где виден класс. Закрытый член доступен другим функциям-членам своего класса. Защищенный член доступен не только другим функциям-членам своего класса, но и функциям-членам класса, унаследованного непосредственно от данного класса. Если все члены открыты, то ключевое слово class можно заменить ключевым словом struct.

1. **Как выполняется конструктор при наследовании?**

При наследовании и инициализации членов класса конструкторы выполняются в следующем порядке:

1. Базовые классы инициализируются в порядке объявления.
2. Члены инициализируются в порядке объявления.

Виртуальные базовые классы создаются до того, как создан любой из производных классов, и до того, как созданы невиртуальные базовые классы. Порядок их создания – «из глубины, слева направо». Деструкторы вызываются в обратном выполнению конструкторов порядке.

1. **Дайте определение чисто виртуальной функции.**

Виртуальная функция может замещаться в производном классе. Выбор того, какое определение функции вызвать для виртуальной функции, происходит динамически на этапе выполнения. Самый распространенный случай – это когда базовый класс содержит виртуальную функцию, а производные классы имеют свои версии этой функции.

1. **Какие операторы используются для разыменования указателя на член класса?**

Указатель на член класса имеет тип T::\*, где T – имя класса. Для разыменования указателя на член класса используются два оператора: .\* и →\*.

Рассмотрим, например, выражение z.\*fet. Здесь сначала разыменовывается указатель для получения переменной-члена, а затем происходит доступ к члену объекта z.

**Лабораторная работа 6**

1. Используя предыдущую программу, создайте новый производный класс с применением множественного наследования.

2. Проверьте работоспособность АТД и производных классов на тестовом наборе данных.

#include <iostream>

#include <string>

#include <clocale>

using namespace std;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_КЛАСС\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

МММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММ

template <class T, class F>

class Voice

{

protected:

T modelNumber;

F power;

int mn, p;

char\* name;

bool unit, errorT, errorF;

public:

Voice();

Voice(T a, F b, char\* c, bool d);

Voice(const Voice<T, F>& a);

~Voice();

///<summary>

///ffff

///</summary>

void Info();

Voice <T, F> sum(const Voice<T, F>& a);

bool Compare(const Voice<T, F>& a);

void Copy(const Voice<T, F>& a);

Voice<T, F>operator =(const Voice<T, F>& a);

Voice<T, F>operator +(const Voice<T, F>& a);

bool operator >(const Voice<T, F>& a);

bool operator <(const Voice<T, F>& a);

bool operator ==(const Voice<T, F>& a);

bool operator !=(const Voice<T, F>& a);

bool operator >=(const Voice<T, F>& a);

bool operator <=(const Voice<T, F>& a);

};

template <class T, class F>

void Voice<T, F> ::Info()

{

cout << "Название: " << name << endl;

cout << "Номер модели звукового устройства: ";

if (errorT)

{

cout << mn;

}

else

{

cout << modelNumber;

}

cout << endl;

cout << "Мощность: ";

if (errorF) {

cout << p;

}

else {

cout << power;

}

cout << "у.е." << endl;

if (unit)

{

cout << "Улучшенное устройство";

}

else

{

cout << "Стоковое устройство";

}

cout << endl;

}

template <class T, class F>

Voice<T, F> ::Voice()

{

try {

if (typeid(T) != typeid(double) && typeid(T) != typeid(int) && typeid(T) != typeid(char) && typeid(T) != typeid(float)) {

throw 1;

}

if (typeid(F) != typeid(double) && typeid(F) != typeid(int) && typeid(F) != typeid(char) && typeid(F) != typeid(float)) {

throw 2;

}

power = (F)0;

modelNumber = (T)10;

errorT = false;

errorF = false;

}

catch (int e){

if (e == 1) {

mn = 10;

cout << "1й указанный вами тип был преобразован в int тк был неверно указан" << endl;

p = 10;

cout << "2й указанный вами тип был преобразован в int тк был неверно указан" << endl;

errorT = true;

}

if (e == 2) {

p = 10;

cout << "2й указанный вами тип был преобразован в int тк был неверно указан" << endl;

}

errorF = true;

}

unit = false;

char \* n = (char\*)"white";

name = new char[strlen(n) + 1];

strcpy(name, n);

}

template <class T, class F>

Voice<T, F> ::Voice(T t, F f, char\* n, bool u)

{

try {

if (typeid(T) != typeid(double) && typeid(T) != typeid(int) && typeid(T) != typeid(char) && typeid(T) != typeid(float)) {

throw 1;

}

if (typeid(F) != typeid(double) && typeid(F) != typeid(int) && typeid(F) != typeid(char) && typeid(F) != typeid(float)) {

throw 2;

}

power = (F)f;

modelNumber = (T)t;

errorF = false;

errorT = false;

}

catch (int e) {

if (e == 1) {

mn = 10;

cout << "1й и, возможно, 2й указанный вами тип был преобразован в int тк были неверно указаны, теперь значение modelNumber = 10, power = 10" << endl;

p = 10;

errorT = true;

errorF = true;

}

if (e == 2) {

p = 10;

cout << "2й указанный вами тип был преобразован в int тк был неверно указан, теперь значение power = 10" << endl;

}

errorF = true;

}

power = (F)f;

unit = u;

name = new char[strlen(n) + 1];

strcpy(name, n);

}

template <class T, class F>

Voice<T, F> ::Voice(const Voice<T, F>& v)

{

if (&v != this)

{

if (errorT) {

mn = v.mn;

}

else

modelNumber = v.modelNumber;

if (errorF) {

p = v.p;

}

else

power = v.power;

name = new char[strlen(v.name) + 1];

strcpy(name, v.name);

unit = v.unit;

}

}

template <class T, class F>

Voice<T, F> ::~Voice()

{

delete[] name;

}

template <class T, class F>

bool Voice<T, F> ::Compare(const Voice<T, F>& a)

{

if (power == a.power && modelNumber == a.modelNumber && unit == a.unit && name == a.name)

return true;

else

return false;

}

template <class T, class F>

void Voice<T, F> ::Copy(const Voice<T, F>& a)

{

if (!errorT) {

if (!a.errorT)

modelNumber = a.modelNumber;

else

modelNumber = a.mn;

}

else {

if (!a.errorT)

mn = a.modelNumber;

else

mn = a.mn;

}

if (!errorF) {

if (!a.errorF)

power = a.power;

else

power = a.p;

}

else {

if (!a.errorF)

p = a.power;

else

p = a.p;

}

name = new char[strlen(a.name) + 1];

strcpy(name, a.name);

unit = a.unit;

}

template <class T, class F>

Voice<T, F> Voice<T, F> ::sum(const Voice<T, F>& a)

{

Voice<T, F> f;

char\* tmp = new char[strlen(name) + 1];

strcpy(tmp, name);

if (!errorT) {

if (!a.errorT)

f.modelNumber = modelNumber + a.modelNumber;

else

f.modelNumber = modelNumber + a.mn;

}

else {

if (!a.errorT)

f.mn = mn + a.modelNumber;

else

f.mn = mn + a.mn;

}

if (!errorF) {

if (!a.errorF)

f.power = power + a.power;

else

f.power = power + a.p;

}

else {

if (!a.errorF)

f.p = p + a.power;

else

f.p = p + a.p;

}

f.unit = true;

strcat(tmp, "-");

strcat(tmp, a.name);

f.name = new char[strlen(tmp) + 1];

strcpy(f.name, tmp);

return f;

}

template <class T, class F>

Voice<T, F> Voice<T, F> :: operator=(const Voice<T, F>& a)

{

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

else

{

this->Copy(a);

return \*this;

}

}

template <class T, class F>

Voice<T, F> Voice<T, F> :: operator+(const Voice<T, F>& a)

{

Voice f = this->sum(a);

return f;

}

template <class T, class F>

bool Voice<T, F> :: operator<(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power < a.power;

else

return power < a.p;

}

else {

if (!a.errorF)

return p < a.power;

else

return p < a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator>(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power > a.power;

else

return power > a.p;

}

else {

if (!a.errorF)

return p > a.power;

else

return p > a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator==(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power == a.power;

else

return power == a.p;

}

else {

if (!a.errorF)

return p == a.power;

else

return p == a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator!=(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power != a.power;

else

return power != a.p;

}

else {

if (!a.errorF)

return p != a.power;

else

return p != a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator<=(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power <= a.power;

else

return power <= a.p;

}

else {

if (!a.errorF)

return p <= a.power;

else

return p <= a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator>=(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power >= a.power;

else

return power >= a.p;

}

else {

if (!a.errorF)

return p >= a.power;

else

return p >= a.p;

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_наследование\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

ММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММ

class Pillow

{

protected:

int soft;

char\* color;

public:

Pillow();

Pillow(int s, char\* c);

~Pillow();

void virtual print();

void refill();

void use();

};

Pillow::Pillow()

{

soft = 0;

color = new char[7];

strcpy(color, "white");

}

Pillow::Pillow(int s, char\* c)

{

soft = s;

color = new char[strlen(c) + 1];

strcpy(color, c);

}

Pillow ::~Pillow()

{

soft = 0;

//color = (char\*)"";

delete[] color;

}

void Pillow::print()

{

cout << "Мягкость: " << soft << endl;

cout << "Цвет: " << color << endl;

}

void Pillow ::refill()

{

soft += 1;

}

void Pillow::use()

{

if (soft != 0)

{

soft -= 1;

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_наследуем\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMМММММ

class InsulationPillow : Pillow

{

private:

double noiseDown;

public:

void print();

InsulationPillow();

void refill();

void use();

InsulationPillow(int s, char\* c, double r);

void speedUp();

};

void InsulationPillow::refill()

{

Pillow::refill();

}

void InsulationPillow::use()

{

Pillow::use();

}

InsulationPillow::InsulationPillow(int s, char\* c, double r) : Pillow::Pillow(s, c)

{

noiseDown = r;

}

InsulationPillow::InsulationPillow() : Pillow::Pillow()

{

noiseDown = 1;

}

void InsulationPillow::print()

{

Pillow::print();

cout << "Шумоизоляция: " << noiseDown << endl;

}

void InsulationPillow::speedUp()

{

noiseDown += 10;

}

template <class T, class F>

class InsulationVoice : public InsulationPillow, public Voice<T, F>

{

protected:

int hertz;

public:

void print();

InsulationVoice();

//s - soft(магкость), c - цвет, r - шумоподавление,

//h - , T - , F - ,

//n - , u -

InsulationVoice(int s, char\* c, double r, int h, T t, F f, char\* n, bool u);

void impruveHertz(int h);

};

template <class T, class F>

InsulationVoice<T, F>::InsulationVoice(int s, char\* c, double r, int h, T t, F f, char\* n, bool u)

: InsulationPillow::InsulationPillow(s, c, r), Voice<T, F>::Voice(t, f, n, u)

{

hertz = h;

}

template <class T, class F>

InsulationVoice<T, F>::InsulationVoice() : InsulationPillow::InsulationPillow(), Voice<T, F>::Voice()

{

hertz = 120;

}

template <class T, class F>

void InsulationVoice<T, F>::print()

{

InsulationPillow::print();

Voice<T, F>::Info();

cout << "Герцовка: " << hertz << endl;

}

template <class T, class F>

void InsulationVoice<T, F>::impruveHertz(int h)

{

hertz += h;

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DO\_IT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMМ

int main()

{

setlocale(LC\_ALL, "rus");

InsulationVoice<int, int> c(20, (char\*)"blue", 2.2, 222, 32, 45, (char\*)"wwwww", false), d;

InsulationPillow a, b;

Voice<int, int> h, g;

a.refill();

if (c == d) {

cout << "первый объект равен 2му\n";

}

else {

cout << "первый объект не равен 2му\n";

}

c.print();

cout << endl;

d.print();

cout << endl;

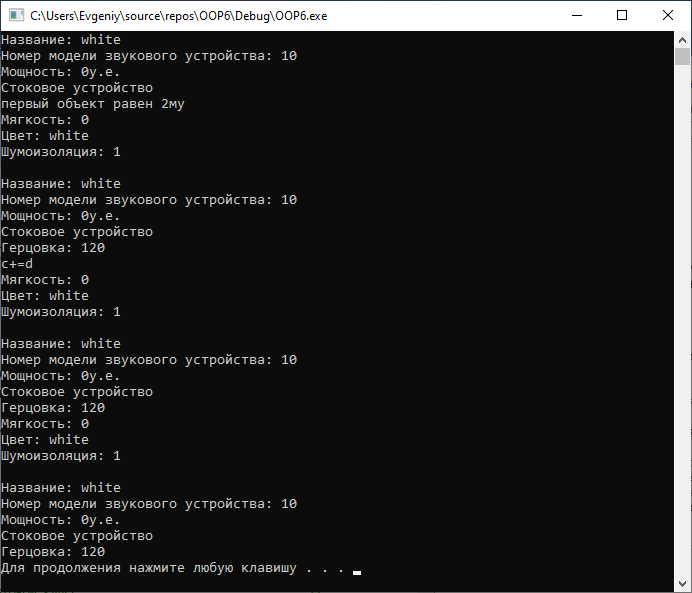
h.Info();

cout << endl;

system("pause");

return 0;

}



1. **С какой целью и в каких случаях используется множественное наследование?**

Множественное наследование (multiple inheritance) делает возможным получение производного класса от нескольких базовых классов.

1. **Опишите синтаксис заголовка производного класса при множественном наследовании.**

Синтаксис заголовка класса расширяется, чтобы можно было использовать список базовых классов с атрибутами доступа.

class shape {

// класс для интерфейса фигуры

};

class tview {

// класс, реализующий просмотр текста

};

class tshape : public shape, private tview {

// адаптер текстового просмотра,

// позволяющий просматривать фигуры

};

В этом примере производный класс tshape открыто наследует базовый класс shape и закрыто наследует класс tview. Подобная схема создания классов называется *адаптерной схемой*. Она использует множественное наследование для объединения интерфейса с реализацией. Поэтому такой прием создания классов еще называют *классом-примесью*.

1. **Дайте определение ориентированного ациклического графа.**

– это граф, узлы которого являются классами, а ориентированные ребра направлены от производных классов к базовым.

1. **Где инициируются виртуальные базовые классы?**

Виртуальные базовые классы инициализируются (вызывается void-конструктор) перед любыми невиртуальными базовыми классами и в том порядке, в котором они появляются в ПАГе наследования при просмотре его снизу-вверх и слева направо.

1. **Что будет, если из объявлений классов Cow и Buffalo убрать ключевое слово virtual (см. последний пример в теоретических положениях)?**

Ключевое слово virtual в классе Cow и классе Buffalo предотвращает многократное копирование полей данных weight, price, color из предков класса Beefalo.

Лаба 7

#include <iostream>

#include <string>

#include <clocale>

using namespace std;

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_КЛАСС\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

МММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММ

template <class T, class F>

class Voice

{

protected:

T modelNumber;

F power;

int mn, p;

char\* name;

bool unit, errorT, errorF;

public:

Voice();

Voice(T a, F b, char\* c, bool d);

Voice(const Voice<T, F>& a);

~Voice();

///<summary>

///ffff

///</summary>

void Info();

Voice <T, F> sum(const Voice<T, F>& a);

bool Compare(const Voice<T, F>& a);

void Copy(const Voice<T, F>& a);

Voice<T, F>operator =(const Voice<T, F>& a);

Voice<T, F>operator +(const Voice<T, F>& a);

bool operator >(const Voice<T, F>& a);

bool operator <(const Voice<T, F>& a);

bool operator ==(const Voice<T, F>& a);

bool operator !=(const Voice<T, F>& a);

bool operator >=(const Voice<T, F>& a);

bool operator <=(const Voice<T, F>& a);

};

template <class T, class F>

void Voice<T, F> ::Info()

{

cout << "Название: " << name << endl;

cout << "Номер модели звукового устройства: ";

if (errorT)

{

cout << mn;

}

else

{

cout << modelNumber;

}

cout << endl;

cout << "Мощность: ";

if (errorF) {

cout << p;

}

else {

cout << power;

}

cout << "у.е." << endl;

if (unit)

{

cout << "Улучшенное устройство";

}

else

{

cout << "Стоковое устройство";

}

cout << endl;

}

template <class T, class F>

Voice<T, F> ::Voice()

{

try {

if (typeid(T) != typeid(double) && typeid(T) != typeid(int) && typeid(T) != typeid(char) && typeid(T) != typeid(float)) {

throw 1;

}

if (typeid(F) != typeid(double) && typeid(F) != typeid(int) && typeid(F) != typeid(char) && typeid(F) != typeid(float)) {

throw 2;

}

power = (F)0;

modelNumber = (T)10;

errorT = false;

errorF = false;

}

catch (int e){

if (e == 1) {

mn = 10;

cout << "1й указанный вами тип был преобразован в int тк был неверно указан" << endl;

p = 10;

cout << "2й указанный вами тип был преобразован в int тк был неверно указан" << endl;

errorT = true;

}

if (e == 2) {

p = 10;

cout << "2й указанный вами тип был преобразован в int тк был неверно указан" << endl;

}

errorF = true;

}

unit = false;

char \* n = (char\*)"HF2";

name = new char[strlen(n) + 1];

strcpy(name, n);

}

template <class T, class F>

Voice<T, F> ::Voice(T t, F f, char\* n, bool u)

{

try {

if (typeid(T) != typeid(double) && typeid(T) != typeid(int) && typeid(T) != typeid(char) && typeid(T) != typeid(float)) {

throw 1;

}

if (typeid(F) != typeid(double) && typeid(F) != typeid(int) && typeid(F) != typeid(char) && typeid(F) != typeid(float)) {

throw 2;

}

power = (F)f;

modelNumber = (T)t;

errorF = false;

errorT = false;

}

catch (int e) {

if (e == 1) {

mn = 10;

cout << "1й и, возможно, 2й указанный вами тип был преобразован в int тк были неверно указаны, теперь значение modelNumber = 10, power = 10" << endl;

p = 10;

errorT = true;

errorF = true;

}

if (e == 2) {

p = 10;

cout << "2й указанный вами тип был преобразован в int тк был неверно указан, теперь значение power = 10" << endl;

}

errorF = true;

}

power = (F)f;

unit = u;

if (n != "") {

name = new char[strlen(n) + 1];

strcpy(name, n);

}

else {

char \* nn = (char\*)"HF2";

name = new char[strlen(nn) + 1];

throw "переменная имени не может быть пустой";

}

}

template <class T, class F>

Voice<T, F> ::Voice(const Voice<T, F>& v)

{

if (&v != this)

{

if (errorT) {

mn = v.mn;

}

else

modelNumber = v.modelNumber;

if (errorF) {

p = v.p;

}

else

power = v.power;

name = new char[strlen(v.name) + 1];

strcpy(name, v.name);

unit = v.unit;

}

}

template <class T, class F>

Voice<T, F> ::~Voice()

{

delete[] name;

}

template <class T, class F>

bool Voice<T, F> ::Compare(const Voice<T, F>& a)

{

if (power == a.power && modelNumber == a.modelNumber && unit == a.unit && name == a.name)

return true;

else

return false;

}

template <class T, class F>

void Voice<T, F> ::Copy(const Voice<T, F>& a)

{

if (!errorT) {

if (!a.errorT)

modelNumber = a.modelNumber;

else

modelNumber = a.mn;

}

else {

if (!a.errorT)

mn = a.modelNumber;

else

mn = a.mn;

}

if (!errorF) {

if (!a.errorF)

power = a.power;

else

power = a.p;

}

else {

if (!a.errorF)

p = a.power;

else

p = a.p;

}

name = new char[strlen(a.name) + 1];

strcpy(name, a.name);

unit = a.unit;

}

template <class T, class F>

Voice<T, F> Voice<T, F> ::sum(const Voice<T, F>& a)

{

Voice<T, F> f;

char\* tmp = new char[strlen(name) + 1];

strcpy(tmp, name);

if (!errorT) {

if (!a.errorT)

f.modelNumber = modelNumber + a.modelNumber;

else

f.modelNumber = modelNumber + a.mn;

}

else {

if (!a.errorT)

f.mn = mn + a.modelNumber;

else

f.mn = mn + a.mn;

}

if (!errorF) {

if (!a.errorF)

f.power = power + a.power;

else

f.power = power + a.p;

}

else {

if (!a.errorF)

f.p = p + a.power;

else

f.p = p + a.p;

}

f.unit = true;

strcat(tmp, "-");

strcat(tmp, a.name);

f.name = new char[strlen(tmp) + 1];

strcpy(f.name, tmp);

return f;

}

template <class T, class F>

Voice<T, F> Voice<T, F> :: operator=(const Voice<T, F>& a)

{

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

else

{

this->Copy(a);

return \*this;

}

}

template <class T, class F>

Voice<T, F> Voice<T, F> :: operator+(const Voice<T, F>& a)

{

Voice f = this->sum(a);

return f;

}

template <class T, class F>

bool Voice<T, F> :: operator<(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power < a.power;

else

return power < a.p;

}

else {

if (!a.errorF)

return p < a.power;

else

return p < a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator>(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power > a.power;

else

return power > a.p;

}

else {

if (!a.errorF)

return p > a.power;

else

return p > a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator==(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power == a.power;

else

return power == a.p;

}

else {

if (!a.errorF)

return p == a.power;

else

return p == a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator!=(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power != a.power;

else

return power != a.p;

}

else {

if (!a.errorF)

return p != a.power;

else

return p != a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator<=(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power <= a.power;

else

return power <= a.p;

}

else {

if (!a.errorF)

return p <= a.power;

else

return p <= a.p;

}

}

template <class T, class F>

bool Voice<T, F> :: operator>=(const Voice<T, F>& a)

{

if (!errorF) {

if (!a.errorF)

return power >= a.power;

else

return power >= a.p;

}

else {

if (!a.errorF)

return p >= a.power;

else

return p >= a.p;

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_наследование\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

ММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММММ

class Pillow

{

protected:

int soft;

char\* color;

public:

Pillow();

Pillow(int s, char\* c);

~Pillow();

void virtual print();

void refill();

void use();

};

Pillow::Pillow()

{

soft = 0;

color = new char[7];

strcpy(color, "white");

}

Pillow::Pillow(int s, char\* c)

{

soft = s;

color = new char[strlen(c) + 1];

strcpy(color, c);

}

Pillow ::~Pillow()

{

soft = 0;

//color = (char\*)"";

delete[] color;

}

void Pillow::print()

{

cout << "Мягкость: " << soft << endl;

cout << "Цвет: " << color << endl;

}

void Pillow ::refill()

{

soft += 1;

}

void Pillow::use()

{

if (soft != 0)

{

soft -= 1;

}

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_наследуем\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMМММММ

class InsulationPillow : Pillow

{

private:

double noiseDown;

public:

void print();

InsulationPillow();

void refill();

void use();

InsulationPillow(int s, char\* c, double r);

void speedUp();

};

void InsulationPillow::refill()

{

Pillow::refill();

}

void InsulationPillow::use()

{

Pillow::use();

}

InsulationPillow::InsulationPillow(int s, char\* c, double r) : Pillow::Pillow(s, c)

{

noiseDown = r;

}

InsulationPillow::InsulationPillow() : Pillow::Pillow()

{

noiseDown = 1;

}

void InsulationPillow::print()

{

Pillow::print();

cout << "Шумоизоляция: " << noiseDown << endl;

}

void InsulationPillow::speedUp()

{

noiseDown += 10;

}

template <class T, class F>

class InsulationVoice : public InsulationPillow, public Voice<T, F>

{

protected:

int hertz;

public:

void print();

InsulationVoice();

//s - soft(магкость), c - цвет, r - шумоподавление,

//h - герцовка, T - номер модели, F - мощность,

//n - , u -

InsulationVoice(int s, char\* c, double r, int h, T t, F f, char\* n, bool u);

void impruveHertz(int h);

};

template <class T, class F>

InsulationVoice<T, F>::InsulationVoice(int s, char\* c, double r, int h, T t, F f, char\* n, bool u)

: InsulationPillow::InsulationPillow(s, c, r), Voice<T, F>::Voice(t, f, n, u)

{

hertz = h;

}

template <class T, class F>

InsulationVoice<T, F>::InsulationVoice() : InsulationPillow::InsulationPillow(), Voice<T, F>::Voice()

{

hertz = 120;

}

template <class T, class F>

void InsulationVoice<T, F>::print()

{

InsulationPillow::print();

Voice<T, F>::Info();

cout << "Герцовка: " << hertz << endl;

}

template <class T, class F>

void InsulationVoice<T, F>::impruveHertz(int h)

{

if (h > 0)

hertz += h;

else

throw "меньше нуля";

}

//\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_DO\_IT\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\\

MMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMMМ

int main()

{

setlocale(LC\_ALL, "rus");

InsulationVoice<int, int> c(20, (char\*)"blue", 2.2, 222, 32, 45, (char\*)"wwwww", false), d;

InsulationPillow a, b;

try {

Voice<int, int> h(12, 22, (char\*)"", false), g;

}

catch (const char\* e) {

cout << e << " переменная заменена на стоковую\n";

}

InsulationVoice<bool, int> ff;//bool использовать нельзя

ff.print();

cout << endl;

a.refill();

if (c == d) {

cout << "первый объект равен 2му\n\n";

}

else {

cout << "первый объект не равен 2му\n\n";

}

try {

d.impruveHertz(-200);

}

catch (const char\* e) {

cout << "переменная е " << e << endl << endl;

}

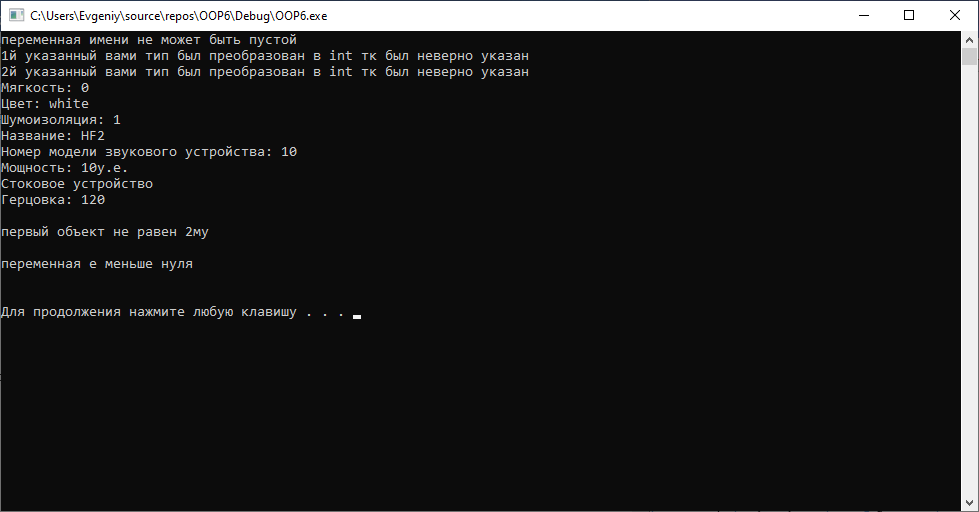
cout << endl;

system("pause");

return 0;

}

Тест



**1. Дайте определение исключения.**

Обычно исключение – это возникающая в программе нештатная ситуация, с которой программа не может справиться. Например, при делении на ноль выполнение программы аварийно завершается системой. С++ дает возможность восстанавливать программу из ошибочных ситуаций и продолжать ее выполнение.

**2. В каком блоке описывается список обработчиков для возбужденного исключения?**

В блоке, который следует сразу за пробным блоком.

**3. Опишите синтаксис обработчика исключения.**

catch (формальный\_аргумент) {

составная\_конструкция}

**4. Что такое спецификация исключения?**

Синтаксически спецификация исключения является частью объявления и определения функции. Если спецификация исключения опущена, то предполагается, что такой функцией может быть возбуждено произвольное исключение.

**5. Для каких целей используется функция unexpected?**

Предоставляемая системой функция unexpected() вызывается, когда она возбудила исключение, которое отсутствует в ее списке спецификации исключений. По умолчанию вызывается функция terminate(). Или же можно задать обработчик, воспользовавшись функцией set\_terminate().