Пример 06.12. Абстрактный класс. Чисто виртуальные методы.

class A // abstract

{

public:

virtual void f() = 0;

};

class B : public A

{

public:

virtual void f() override { cout<<"Executing f from B;"<<endl; }

};

class C

{

public:

static void g(A& obj) { obj.f(); }

};

void main()

{

B obj;

C::g(obj);

}

Пример 06.15. Виртуальный деструктор.

class A

{

public:

virtual ~A() = 0;

virtual void f() {}

virtual void f(int i) {}

};

A::~A() {}

class B : public A

{

public:

virtual ~B() override { cout<<"Class B destructor called;"<<endl; }

virtual void f() override {}

};

void main()

{

A\* pobj = new B();

p->f(1);

delete pobj;

}

Пример 06.13. Виртуальные методы и конструкторы и деструкторы.

class A

{

public:

virtual ~A() { cout<<"Class A destructor called;"<<endl; }

virtual void f() { cout<<"Executing f from A;"<<endl; }

};

class B : public A

{

public:

B() { this->f(); }

virtual ~B()

{

cout<<"Class B destructor called;"<<endl;

this->f();

}

void g() { this->f(); }

};

class C : public B

{

public:

virtual ~C() { cout<<"Class C destructor called;"<<endl; }

virtual void f() override { cout<<"Executing f from C;"<<endl; }

};

void main()

{

C obj;

obj.g();

}

Пример 06.14. Дружба и наследование.

class C; // forward объявление

class A

{

private:

void f1() { cout<<"Executing f1;"<<endl; }

friend C;

};

class B : public A

{

private:

void f2() { cout<<"Executing f2;"<<endl; }

};

class C

{

public:

static void g1(A& obj) { obj.f1(); }

static void g2(B& obj)

{

obj.f1();

// obj.f2(); // Error!!! Имеет доступ только к членам A

}

};

class D : public C

{

public:

// static void g2(A& obj) ( obj.f1(); } // Error!!! Дружба не наследуется

};

void main()

{

A aobj;

C::g1(aobj);

B bobj;

C::g1(bobj);

C::g2(bobj);

}

Пример 06.16. Дружба и виртуальные методы.

class C; // forward объявление

class A

{

protected:

virtual void f() { cout<<"Executing f from A;"<<endl; }

friend C;

};

class B : public A

{

protected:

virtual void f() override { cout<<"Executing f from B;"<<endl; }

};

class C

{

public:

static void g(A& obj) { obj.f(); }

};

void main()

{

B bobj;

C::g(bobj);

}

Пример 06.17. Множественное наследование и виртуальные методы.

class A

{

public:

virtual ~A() = 0;

virtual void f() { cout<<"Executing f from A;"<<endl; }

};

A::~A() {}

class B

{

public:

virtual ~B() = 0;

virtual void f() { cout<<"Executing f from B;"<<endl; }

};

B::~B() {}

class C : private A, public B

{

public:

virtual ~C() {}

virtual void f() { cout<<"Executing f from C;"<<endl; }

};

class D

{

public:

void g1(A& obj)

{

obj.f();

}

void g2(B& obj)

{

obj.f();

}

};

void main()

{

C obj;

D d;

d.g2(obj);

d.g2(obj);

}

Пример 06.18. Проблемы массива объектов.

class A

{

public:

virtual void f() = 0;

};

class B : public A

{

int b;

public:

virtual void f() override { cout<<"Executing f from B;"<<endl; }

};

class C

{

public:

static A& index(A\* p, int i) { return p[i]; }

};

void main()

{

const int N = 10;

B vect[N];

A& alias = C::index(vect, 5);

alias.f(); // Error!!!

}

Пример 07.1. Обработка исключительных ситуаций.

class ExceptionArray : public std::exception

{

protected:

char\* errormsg;

public:

ExceptionArray(const char\* msg)

{

int Len = strlen(msg) + 1;

this->errormsg = new char[Len];

strcpy\_s(this->errormsg, Len, msg);

}

virtual ~ExceptionArray() { delete[] errormsg; }

virtual const char\* what() const noexcept override { return this->errormsg; }

};

class ErrorIndex : public ExceptionArray

{

private:

const char\* errIndexMsg = "Error Index";

int ind;

public:

ErrorIndex(const char\* msg, int index) : ExceptionArray(msg), ind(index) {}

virtual ~ErrorIndex() {}

virtual const char\* what() const noexcept override

{

int Len = strlen(errormsg) + strlen(errIndexMsg) + 8;

char\* buff = new char[Len + 1];

sprintf\_s(buff, Len, "%s %s: %4d", errormsg, errIndexMsg, ind);

char\* temp = errormsg;

delete[]temp;

const\_cast<ErrorIndex\*>(this)->errormsg = buff;

return errormsg;

}

};

int main()

{

try

{

throw(ErrorIndex("Index!!", -1));

}

catch (ExceptionArray& error)

{

cout << error.what() << endl;

}

catch (std::exception& error)

{

cout << error.what() << endl;

}

catch (...)

{

}

return 0;

}

Пример 07.2. Блок try для раздела инициализации конструктора.

class Array

{

private:

double\* mas;

int cnt;

public:

Array(int q);

~Array() { delete[] mas; }

};

Array::Array(int q) try: mas(new double[q]), cnt(q)

{}

catch(const std::bad\_alloc& exc)

{

cout<<exc.what()<<endl;

}

void main()

{

Array a(-1);

}