Лекция 1 03.01.25

ЮВСЕЕВА ОЛЬГА ВИКТОРОВНА

Структура, Класс— это описание пользовательского типа данных.

Class Animal {

char [50] name;

int height;

int weight;

};

В классе может описываться функция. Структуры используются для хранения данных чаще всего. Структуры не обладают свойствами объекта.

Метод = функция;

Абстрагирование создание объекта типа класс;

Принципы: инкапсуляции / полиморфизм / наследование.

Инкапсуляция — принцип, согласно которому от пользователя скрываются данные.

Спецификаторы доступа — private, public, protected;

В классе по умолчанию спецификатор доступа — private;

Class Animal {

char [50] name;

int height;

int weight;

public:

void voice(){

cout << /// << name;

}

};

Для соответствия принципу инкапсуляции необходимо использовать геттеры и сеттеры.

Конструктор — функция, вызываемая авто при создании объекта типа class.

Деструктор — функция, вызываемая при удаления объекта типа class.

Class Animal {

char [50] name;

int height;

int weight;

public:

Animal(){

/// конструктор

}

-Animal (){

// деструктор

}

};

Инициализация списком: Animal () : name (..), height (..), weight(..);

Kонструктор с параметрами:

Animal (char\* \_name, int \_height, int \_weight){

name = ///

///

}

Конструктор копирования :

Animal(Animal \_animal){

height = animal.height;

///

name = new name[50];

for (/// ) {

name[i] = \_animal.name[i];  
 }

}

Деструктор выполняется автоматически при удалении объекта. В основном используется для избежания утечки памяти, для закрытия потоков и т. д.

Animal cat;

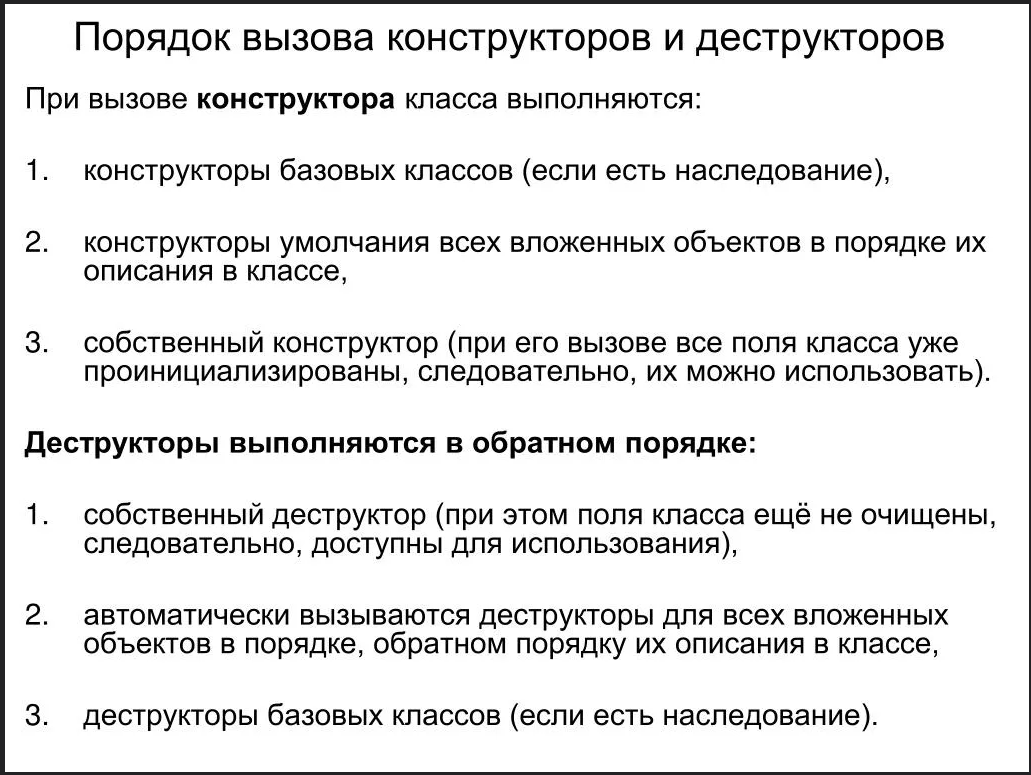
Animal cat01();

Animal cat02( «name », 1, 2);

Animal cat03 = cat02; // конструктор копирования

Animal cat04 (cat03); // конструктор копирования

Порядок вызова конструкторов и деструкторов: …



Указатель на член класс this:

this.name = new name [50];

Статические члены класса.

Public:

static int count;

static void get\_count () {///}

Animal () {

count++;

}

-Animal () {

count--;

}

Статические переменные одни на весь объект.

Возможно обращения: Animal::count;

Статическим может быть и метод: Animal::get\_Count();

Lessons 2; 17/02/25

Class B;

Class X{

int ia1;

public:

X() {

ia = 0;  
 }

};

class B {

int b1;

public:

frind int X::Func1(B& b1) // общедоступная функция из пространства имен

}

int X::func(B& bb){

ia1 + =bb.b1

return ia1;

}

void main() {

int b1;

B b2;

X a2;

i1 = a2.func1(b2);

}

////

Class B;

class D {

int x ;

friend void func(B& b , D& d);  
};

void func(B& b, D& d){

cout << b.x+d.y;  
}

Дружественность позволяет указать несколько классов, связанных между собой.

Class Auto;

class Motorcycle {

friend class Auto;

bool isStillWorking = true;

};

class Auto {

public:

void crash DTP (Motorcycle & moto){

moto.isStillWorking = false;

}

};

//

Статический полиморфизм реализуется путем функций перегрузки;

Перегружать можно дружественной функции, метода.

Перегружать нельзя . :: ?: .\* # sizeof typeid

Число парматеров функции меньше на один.

При перегрузке дружественной функцией, то парматры функции = фактическим передаваемым переменных.

Class Complex {

double re, in;

public:

Complex(double r = 0, double i = 0): re(i), in (i){}

Complex perator + (const Complex& y);

}

Complex Comples::operator + (const Complex & y) {

Complex t (re + y.re, im + y.im);

return t;

}

int main () {

Complex t1(2, 5);

Complex t2(4, 6);

cout << t1 + t2;

}

Операция присваивания может быть перегружено только методом класса.

Class Vector {

int \* p;

int size;

public:

Vector& opeerator = (const Vector & v1);

friend Vector & += (const vector & v1, const Vector & v2);

}

Vector & Vector::operator = (const Vector& v1) {  
 if (size != v1.size) {

delete p;

size = v1.size;

p = new int [size];

}

for (int i = 0; i < size; i++) {

p[i] = v1.p[i];  
}

return \*this;

}

Оператор += пергрузка дружественной функцией:

...

Vector & operator += (Vector & v1, const Vector& v2){

int j = v1.size();

if (j > v2.size()) {  
 j = v2.size;

for (int i = 0; i <j; i ++){  
 v1.p[i] + = v2.p[i];

}

}

Перегрузка унарных операций:

Complex & operator ++ () {  
 ++ re;

return \* this;

}

Complex operator ++ (int re) {

Complex tmp(\*this);

re++;

return tmp;

}

int main () {

Complex c1(1, 1);

Complex c2 = c1++; // c2.re = 1

Complex c3 = c1++; //c3.re = 2  
}

Перегрузка индексирования:

Class Vector {

int \* p;

int size;

public:

int operator [] (int i) {  
 return p[i];

}

}

Перегрузка ввода и вывода:

#include <iostream>

class Complex {

double re, im;

public:

friend ostrema & operator operator << (ostream & outm const Complex& par);

};

ostream & operator << (ostream & out, const Complex & par){  
 out << par.re << “” < par. im << “i”;

return out;

}

Лекция 5 17.03.25

Виртуальность методов

Динамический полиморфизм

Виртуальные деструкторы

Интерйефс указателя

Закрытое наследование

Таблица вертуальных методов

Ромбовидное наследование

Невиртуальный деструктор приведет к утечке памяти

Класс реализации

Виртуальный деструктор нужен для нединамических

Конструктор нельзя описывать как виртуальный.

Полиморфный тип

class A {

int \*pa;

int ia;

public:

A(int par = 10){

ia = par1;

pa = new int [par1];

cout << “A()”;

}

virtual ~A () {

delete [] pa;

cout << “~A”;

}

virtual int sa() {  
 return ia;

}

};

class C: public A {

double \*pc;

int ic;

public:

C(int par2 = 10, int par\_1 = 10): A (par1){

ic = par2;

pc = new double[par2];

cout << “ C()“;

}

~C() {

delete [] pc;

cout << “ ~C()”

}

};

void main(){

A\* ppc = new C(5);

cout << “\n” << “size int = ” << pp1-> sa();

cout << ((C\*)ppa1)-> sc()<< “\n”;

delete pp1;

}

Console log:

A() C() size int = 10 size double =5

~C() ~A()

Необходнима память на храненение вертуальных методов

АБСТРАКТНЫЕ КЛАССЫ ЧИСТЫЕ ВИРТУАЛЬНЫЕ ФУНКЦИИ

virtual func() = 0;

Абстрактный класс - класс, содержащий 1 и более виртуальный метод

Абстрактные классы не могут быть заданы в виде объектов

Указатели на абстрактные классы можно задать

Класс наследника абастратного класса тоже может быть наследником

МНОЖЕСТВТЕННОЕ НАСЛЕДОВАНИЕ ИНТЕРФЕСОВ

Class X

{

int i1;

int jx;

}

Class Y {

int i1;

itn ij;

}

Class Z: Public X, public Y {

int jz;

}

void main () {  
 Z z1;

z1.X::i1 = 5;

}

ВИРТУАЛЬНЫЕ БАЗОВЫЕ КЛАССЫ

class W {

}

class X: public virtual W {

}

class Y: public virtual W {

}

class Z: public X, pulic Y{

}

выше пример ромбовидного наследования

2 картинки

ТЕМА

RUN Time Type Identification

dynamic\_cast

typeid

type\_info

class A {

vitual vodi fx (){  
 cout << “A::fx”<< “ \n”;

}

};

class B: public A {

public:

vodi fx() {

cout << “B::fx” << “\n”;

}

};

void f2(A\* ptr ) {  
 B\* dptr = dynamic\_cast<B\*> (ptr);

cout << “type of pointer dptr: ” << typeid(dptr).name() << “\n”;

}

void main () {  
 A \* p1 = new(A);

f2(p1);

}

РЕЗУЛЬТАТ: type pointer dptr : class B\*