c++学习笔记:

1. 引用：本质是常指针

引用和指针的区别：

1. 函数重载：

定义：用同一个函数名定义不同的函数，当函数名和不同的参数搭配时函数的含义 不同。

规则：1,函数名相同。

2,参数个数不同,参数的类型不同,参数顺序不同,均可构成重载。

3,返回值类型不同则不可以构成重载。

1. 内联函数：

特点：1）内联函数声明时inline关键字必须和函数定义结合在一起，否则编译器会直

接忽略内联请求。

1. C++编译器直接将函数体插入在函数调用的地方 。
2. 内联函数没有普通函数调用时的额外开销(压栈，跳转，返回)。

4）内联函数是一种特殊的函数，具有普通函数的特征（参数检查，返回类型等）。 5）内联函数由编译器处理，直接将编译后的函数体插入调用的地方，

宏代码片段 由预处理器处理， 进行简单的文本替换，没有任何编译过程。

1. C++中内联编译的限制：

不能存在任何形式的循环语句

不能存在过多的条件判断语句

函数体不能过于庞大

不能对函数进行取址操作

函数内联声明必须在调用语句之前

7）编译器对于内联函数的限制并不是绝对的，内联函数相对于普通函数的优势 只是省去了函数调用时压栈，跳转和返回的开销。因此，当函数体的执行开

销远大于压栈，跳转和返回所用的开销时，那么内联将无意义。

1. 面向对象

特点：封装、继承、多态

1. 封装（属性和方法）

本质：在于将数据和行为,绑定在一起然后能通过对象来完成操作。

访问属性：

Public、protected、private

Public修饰成员变量和成员函数可以在类的内部和类的外部被访问。

Private和protected修饰成员变量和成员函数只能在类的内部被访问

struct和class关键字区别

在用struct定义类时，所有成员的默认属性为public

在用class定义类时，所有成员的默认属性为private

1. 对象的构造函数

类中会有默认的无参构造函数，当没有任何的显示构造函数（有参、无参、拷贝）时， 默认无参构造就会没有

定义：与类名相同的特殊成员函数

调用：自动调用：一般情况下C++编译器会自动调用构造函数.

手动调用：在一些情况下则需要手工调用构造函数.

规则:

1 在对象创建时自动调用,完成初始化相关工作。

2 无返回值,与类名同,默认无参,可以重载,可默认参数。

3 一经实现,默认不复存在。

构造函数分类

无参构造函数 Test t；

带参构造函数 Test(10);

1. 析构函数

类中有默认析构函数，需要回收资源的时候在写也可以，当没有显示析构时，默认 析构就会出现

作用：并不是删除对象,而在对象销毁前完成的一些清理工作。

规则:

1 对象销毁时,自动调用。完成销毁的善后工作。

2 无返值 ,与类名同。无参。不可以重载与默认参数

1. 拷贝构造函数

类中默认的拷贝构造函数（单纯的将另一个对象的值赋值），当没有显示的拷贝构 造时，默认拷贝构造就会出现

由己存在的对象,创建新对象。也就是说新对象,不由构造器来构造,而是由拷贝构造 器来完成。

应用场景：

1. Test t1(10);

Test t2=t1; //⽤对象t1初始化对象 t2，调用t2的拷贝构造数

2. Test t1(10);

Test t2（t1）; //⽤对象t1 初始化 对象 t2

3. Location b=a; 对象b调⽤copy构造函数进⾏初始化

func(b); b实参取初始化形参p,会调⽤copy构造函数

1. Test func()

{

Test temp(10);

Return temp; //匿名的对象 = temp 调用匿名对象的拷贝构造

}

Void test()

{

Func(); //当一个函数返回一个匿名对象，函数外部没有任何变量 区接收它，这个匿名对象将不会再被使用，编译器会直接将 这个匿名对象回收掉，而不是等待函数执行完毕再回收，在 这里调用匿名对象的析构函数

Test t1=func();//并不会触发t1的拷贝构造，而是直接将匿名对象转 正，起名为t1，就不会被立即析构

Test t2；

t2=func(); //触发t2的等号操作符重载函数，匿名对象没有转正，直接析构

} //t1的析构

1. 深拷贝与浅拷贝

如果类中包含的数据元素全部在栈上,浅拷贝可以满足需求的。但类中成员有指针，就需要深拷贝。

Class Teacher

{

Teacher(int id,char \*name)

{

m\_id = id;

Int len= strlen(name);

m\_name = (char \*)malloc(len+1);

Strcpy(m\_name,name);

}

Teacher(const Teacher & another)

{

m\_id =another.m\_ id;

Int len= strlen(another.m\_name);

m\_name = (char \*)malloc(len+1);

Strcpy(m\_name,another.m\_name);

}

~Teacher()

{

If(m\_name != NULL)

{

Free(m\_name);

m\_name = NULL;

}

}

Void printT()

{

Cout<<”id=”<<m\_id <<”,name=”<<m\_name<<endl;

}

}

Public:

Int m\_id;

Char \*m\_name;

}

Int main()

{

Teacher t1(1,”zhangsan”);

t1.printT();

Teacher t1(t2); //若调用t1的默认拷贝构造函数，则在析构的时候，会析构两次 同一个堆区空间，程序出错//若调用t1的深拷贝构造函数，就会在堆上重新开辟 空间，析构的时候，会析构两次各自的堆区空间

}

1. 构造函数的初始化列表

A类成员的一个成员是B类的情况，，而且这个成员B它只有一个带参数的构造函数， 没有默认构造函数。这时要对这个A类成员进行初始化，就必须调用B类成员的带参 数的构造函数，利用初始化列表完成。

Class A{

Int a;

}

Class B{

Private:

A m\_a;

Const Int b;

Public:

B(A &a，int b):m\_a(a),b(100) //在这里调用m\_a的拷贝构造函数

B(int a，int b):m\_a(a),b(100) //在这里调用m\_ a的有参构造函数

}

Note：

当类成员中含有一个const对象时，或者是一个引用时，他们也必须要通过成员初 始化列表进行初始化，因为这两种对象要在声明后马上初始化，而在构造函数中， 做的是对他们的赋值，这样是不被允许的。

初始化列表中的初始化顺序,与成员对象在类中的定义顺序有关,与赋值顺序无关。

Test(10)； //会产生匿对象，先调用有参构造，然后立即调用匿名对象的析构函数

1. new和delete

使用格式：指针变量=new 类型(常量) //开辟一个单元 int \*p=new int(10);

指针变量=new 类型[表达式] //开辟数组 int \*p=new int[10];

Delete 指针变量

Delete[] 指针变量

区别：

new和delete是c++语言的运算符,不是函数，因此执行效率高,malloc和free是标 准库函数

malloc不会调用类的构造函数,而new会调用类的构造函数 Test \*tp=new Test(10);

Free不会调用类的析构函数，而delete会调用类的析构函数

1. static修饰的静态成员变量

类的静态成员,属于类,也属于对象,但终归属于类。在全局变量区分配一个空间、

声明：static int m\_a; //类的内部

初始化：int Test::m\_a=10; //类的外部

调用：类名::静态成员变量 Test::m\_a

类对象.静态成员变量 t1.m\_a

总结：

1,static 成员变量实现了同类对象间信息共享。

2,static 成员类外存储,求类大小,并不包含在内。

3,static 成员是命名空间属于类的全局变量,存储在 data 区。

4,static 成员只能类外初始化。

5,可以通过类名访问(无对象生成时亦可),也可以通过对象访问。

1. 静态成员函数

//声明 static 函数声明；

//调⽤ 类名::函数调⽤ 静态函数属于整个类

类对象.函数调⽤

总结：

1,静态成员函数的意义,不在于信息共享,数据沟通,而在于管理静态数据成员, 完

成对静态数据成员的封装。

2,静态成员函数只能访问静态数据成员。原因:非静态成员函数,在调用时this 指

针被当作参数传进。而静态成员函数属于类,而不属于对象,没有 this 指针

比如：private 的静态成员变量，若在类的外部访问，必须通过静态成员函数

1. 静态成员函数和变量的存储

C++类对象中的成员变量和成员函数是分开存储的。

普通成员变量：存储于对象中，与struct变量有相同的内存布局和字节对齐方式

静态成员变量：存储于全局数据区中

静态成员函数与普通成员函数的区别

静态成员函数不包含指向具体对象的指针

普通成员函数包含一个指向具体对象的this指针

1. this指针

this指针是常指针Test \*const， 只能指向对象本身

类的成员函数尾部出现const ，修饰的是this指针

1. 全局函数与成员函数

全局函数：

Test add(Test &t1,Test& t2)

{

Test t3=(t1.getA() + t2.getA(),t1.getB() + t2.getB())

Return t3;

}

成员函数

Test add(Test& another) 不可以实现连续操作

{

This->a +=this->a + another.a;

This->b += this->b + another.b;

Return this; //返回的不是对象本身，而是一个匿名对象

}

可以实现 (+=) +=

Test& add(Test& another)

{

This->a +=this->a + another.a;

This->b += this->b + another.b;

Return \*thids; //返回对象本身，如果想对一个对象连续调用成员方法，每次 调用都会改变对象本身，需要返回引用

}

1. 自定义数组类

MyArray.h

Class MyArray

{

Public:

MyArray();

MyArray(int len);

MyArray(const MyArray &another);

MyArray& operator=(const MyArray &another);

Int& operator[](int index)const;

Friend bool operator==(MyArray& array1,MyArray& array2);

Friend Ostream& operator<<(ostream& os,const MyArray& array);

Friend istream& operator<<(istream& is,MyArray& another);

Void setData(int index,int data);

Int getData(int index);

Int getLen()const;

~ MyArray();

Private:

Int len; //数组长度

Int \*space; //space在堆上开辟空间，space为地址

}

MyArray.c

MyArray::MyArray()

{

Cout<<”MyArray()...”<<endl;

This->len = 0;

This->space = NULL;

}

MyArray::MyArray(int len)

{

If(len <= 0)

{

This->len = 0;

Rerturn ;

}

This->len = len;

This->Space = new int[this->len];

Cout<<”MyArray(int len)...”<<endl;

}

MyArray::MyArray(const MyArray &another)

{

If(This->len >= 0)

This->len = another.len;

This->space = new int[this->len];

For(int i = 0;i<this->len;++i)

This->space[i]= another.space[i];

Cout<<”:MyArray(const MyArray &another)...”<<endl;

}

MyArray& MyArray::operator=(const MyArray &another)

{

If(this == &another)

Return ;

If(this->space != NULL)

{

Delete[] this->space;

This->space = NULL;

This->len = 0;

}

//深拷贝

If(This->len >= 0)

This->len = another.len;

This->space = new int[this->len];

For(int i = 0;i<this->len;++i)

This->space[i]= another.space[i];

}

Int& MyArray::operator[](int index)const

{

Rerurn this->space[index];

}

Ostream& operator<<(ostream& os,const MyArray& array) //实现cout<<array1

{

Os<<”遍历整个数组”<<endl;

For(int i =0; i < array.getLen(); ++i)

Os<<array[i]<<” ”l;

Os<<endl;

Return os;

}

istream& operator<<(istream& is,MyArray& array)

{

Cout<<”请输入”<<array.getLen()<<”个数”<<endl;

For(int i = 0; i < array,getLen();++i)

Is

>>array[i];

Return is;

}

bool operator==(MyArray& array1,MyArray& array2)

{

If(array1.len != array2.len)

Return false;

For(int i=0;i<array1.len;++i)

{

If(array1.lspace[i] != array[2].space[i])

Return false;

}

Return true;

}

MyArray::~MyArray()

{

If(this->space != NULL)

{

Delete[] this->space;

This->space = NULL;

This->len = 0;

}

Cout<<”~MyArray()...”<<endl;

}

Void MyArray::setData(int index,int data)

{

If(this->space != NULL)

This->space[index] = data;

}

Void MyArray::getData(int index)

{

Retuen This->space[index] ;

}

Int MyArray::geLen()const

{

Return this->len;

}

Int main()

{

MyArray array1(10); //开辟10个元素的数组

for(int i = 0; i < 10; ++i)

Array1.setData(i,i+10); //array1[i] = i+10;

For(int i = 0;i< 10; i++)

Cout<<array1.getData(i)<<” ”; //array1[i]

Cout<<endl;

MyArray array2=array1; //调用array2的深拷贝构造函数

For(int i = 0;i< 10; i++)

Cout<<array2.getData(i)<<” ”; //array2[i]

Cout<<endl;

Cout<<array2; //调用了<<的重载

MyArray array3(5);

Cin>>array3;

If(array1 == array3)

Cout<<”not equal”<<endl;

}

1. 友元函数

同类对象之间无私处，异类对象之间有友元

友元函数

优点：是可以直接访问类私有成员的非成员函数。是定义在类外的普通函数,它不 属于任何类,但需要在类的定义中加以声明,声明时只需在友元的名称前加上 关键字friend

缺点：破坏类的封装性和隐藏性，使非成员函数可以访问类的私有成员

全局函数可以做友元函数 ：直接在类中声明

类成员函数作友元函数：前向声明类+函数实现和声明分开写

1. 友元类

友元类的所有成员函数都是另一个类的友元函数,都可以访问另一个类中的隐藏信息(包括私有成员和保护成员)。

格式：friend class 类名;

其中:friend 和 class 是关键字,类名必须是程序中的⼀个已定义过的类。

注意事项:

(1) 友元关系不能被继承。

(2) 友元关系是单向的,不具有交换性。若类 B 是类 A 的友元,类 A 不一定是类

B的友元,要看在类中是否有相应的声明。

(3) 友元关系不具有传递性。若类 B 是类 A 的友元,类 C 是 B 的友元,类 C 不一

定是类 A 的友元,同样要看类中是否有相应的声明。

1. 操作符重载

操作符重载+函数作为全局函数(友元函数)

Complex operator+(Complex &c1,Complex &c2) //Complex 类的友元函数

{

Complex temp = (c1.a+c2.a,c1.b+c2.b);

Return temp;

}

main()

{ //全局的调用方式

Complex c3= c1+c2;

//Complex c3 = operator+(c1+c2);

//Complex c3 = c1.operator+(c2);

}

Note:如果要连减，不能返回引用

操作符重载+作为类成员函数(成员函数)

Complex operator+(Complex &another) //Complex类的成员函数

{

Complex temp = (this->a+another.a,this->b+another.b);

Return temp;

}

main()

{

Complex c3= c1+c2;

//Complex c3 = c1.operator+(c2);

}

+=操作符重载

友元重载

Complex& operator+=(Complex &c1,Complex &c2) //Complex 类的友元函数

{

c1.a += c2.a；

c1.b += c2.b;

Return c1;

}

成员函数

Complex& operator+=(Complex &another)

{

This->a += another.a;

This->b += another.b;

Return \*this;

}

前++：++c1

Complex& operator++(Complex &c1) //Complex 类的友元函数

{

c1.a++;

C1.b++;

Return c1;

}

Complex& operator++() //Complex 类的友元函数

{

This->a++;

This->b++;

Return \*this;

}

后++：c1++

Const Complex operator++(Complex &c1，int) //Complex 类的友元函数，占位符

{

Complex temp(c1.a,c1.b);

c1.a++;

C1.b++;

Return temp;

}

Const Complex operator++(int) //Complex 类的友元函数

{

Complex temp=\*this;

This->a++;

This->b++;

Return temp;

}

左移、右移操作符：只能写在全局，不能写在成员方法中，否则调用顺序会变反

等号操作符=

默认等号操作符函数实现浅拷贝，当类的成员中有指针是，不合适，需要重载

Student& Operator=(const Student &another) //返回引用，支持连等

{

//防止自身赋值

If(&another == this)

Return \*this;

//先将自身的垃圾回收掉

If(this->name != NULL)

{

Delete[] this->name;

This->name =NULL;

This->id = 0;

}

//深拷贝

this->id= another.id;

Int len= strlen(another.name);

This->name = new char[len+1];

Strcpy(this->name,another.name);

Return \*this;

}

Main()

{

Student s1(1,”zhangsan”);

Student s2=s1; //执行s2的拷贝构造函数

Student s3;

S3=s1; //执行s3的等号操作符重载

}

重载[]

1. 继承：

子类全盘接收父类,除了构造器与析构器。

1、子类对象在创建时会首先调用父类的构造函数

2、父类构造函数执行结束后，执行子类的构造函数

3、当父类的构造函数有参数时，需要在子类的初始化列表中显示调用

4、析构函数调用的先后顺序与构造函数相反

Class child

{

Public:  
 child(int a,int b):Parent(a) //显式调用子类的有参构造

{

This->a = a; //默认调用子类的无参构造

This->b = b;

}

}

语法：

class 派⽣类名:[继承⽅式] 基类名

继承方式：

Publicl:公有继承，当类的继承方式为公有继承时,基类的公有和保护成员的访问属 性在派生类中不变,而基类的私有成员不可访问。

Protected：保护继承，保护继承中,基类的公有成员和私有成员都以保护成员的身 份出现在派生类中,而基类的私有成员不可访问

Private:私有继承，当类的继承方式为私有继承时,基类中的公有成员和保护成员都 以私有成员身 份出现在派生类中,而基类的私有成员在派生类中不可访问。

1、需要被外界访问的成员直接设置为public

2、只能在当前类中访问的成员设置为private

3、只能在当前类和子类中访问的成员设置为protected，protected成员的访问权限 介于public和private之间。

兼容性原则：

子类对象可以当作父类对象使用

子类对象可以直接赋值给父类对象，子类对象所占内存空间大于等于父类

子类对象可以直接初始化父类对象

父类指针可以直接指向子类对象，子类指针不能直接指向父类对象，因为父类对象 满足不了子类指针的全部需求 parent \*pp,child \*cp; parent p;child c; pp=&c;

父类引用可以直接引用子类对象

父类和子类出现同名变量

1、当子类成员变量与父类成员变量同名时

2、子类依然从父类继承同名成员

3、在子类中通过作用域分辨符::进行同名成员区分（在派生类中使用基类的同名成 员，显式地使用类名限定符）

4、同名成员存储在内存中的不同位置

Class child

{

Public:

Child(int p\_a,int c\_a):parent(p\_a)

This->a = c\_a;

Void printT()

{

Cout<<this->a<<endl; //child 的a

Cout<<parent::a<<endl; //parent 的a

}

Int a;

}

子类中的static关键字

基类定义的静态成员，将被所有派生类共享。静态成员只有一份

多继承：

Public sofabed：public sofa,public bed

虚继承：

如果一个派生类从多个基类派生，而这些基类又有一个共同的基类，则在对该 基类中声明的名字进行访问时，可能产生二义性

如果出现菱形继承，父类继承爷爷类的时候要进行虚继承virtual，防止子类在 多继承时爷爷中的变量拷贝多份，出现二义性。

1. 多态

多态的三个必要条件

1.父类指针（引用）指向子类对象。则调用子类重写的虚函数，而不是父类的函数

2.要有继承，子类继承父类

3.要有虚函数重写子类重写父类的虚函数

虚析构函数

构造函数不能是虚函数。

析构函数可以是虚的。执行子类的虚构函数，会默认触发父类的虚构函数（多态）

重载、重写、重定义

重载：

在同一个作用域的两个函数、函数名相同、函数参数列表不同

重定义：（普通函数重定义）

发生在两个不同类中，一个父类、一个子类，父类普通函数被子类重写，父类 和子类的函数名、参数列表都一样

重写：（虚函数重写）

发生在两个不同类，一个父类、一个子类，父类虚函数被子类重写，发生多态

多态的原理

虚函数表和vptr指针

当类中声明虚函数时，编译器会在类中生成一个虚函数表；

虚函数表是一个存储类成员函数指针的数据结构；

虚函数表是由编译器自动生成与维护的；

virtual成员函数会被编译器放入虚函数表中；

存在虚函数时，每个对象中都有一个指向虚函数表的指针(vptr指针)。

Parent \*pp = &p;

Parent \*pp = &c;

vptr指针初始化

vptr指针初始化是分步进行的，

对象在创建的时,由编译器对VPTR指针进行初始化只有当对象的构造完全结 束后VPTR的指向才最终确定

Class parent

{

Public :

Parent(int a)

{

This->a =a;

Print(); //这里调用的是父类的print函数

}

Virtual void print()

{

}

}

Class child:public parent

{

Public:

Child(int a,int b):Parent(a) //在调用父类的构造函数时，会将 vptr指针当做父类来处理，此时会临时指向父类的虚函数表

{

//将子类对象空间变成子类对象处理，vptr从此就只向子类 的虚函数表

}

Virtual void print()

{

}

}

Main

{

Parent \*pp = new child(10,20);

Sizeof(parent) ; //8

Sizeof(child); //12 int a,int b,vptr

}

父类指针和子类指针的步长

当通过父类指针遍历子类数组时，会出错

Main

{

Child array[] = { child(1),child(2),child(3)};

Parent \*pp = &array[0];

Child \*cp = &array[0];

For(i=0;i<3;++i,++pp) //pp+sizeof(parent),

Pp->print(); //出错，父类指针的步长和子类指针步长不同

}

纯虚函数

定义：纯虚函数是一个在基类中说明的虚函数，在基类中没有定义，要求任何 派生类都要重写这个纯虚函数，不重写的话，这个派生类还是抽象类

语法：virtual 类型 函数名（参数表）=0;

一个具有纯虚函数的基类称为抽象类。

抽象类不能够实例化 //Figure f;

1. 智能指针(类型auto ptr)

#include <memory>

Main

{

auto ptr<A> ptr(new A) //数据类型，定义一个指向int类型的智能指针ptr

//智能指针是一个类，ptr是对象，指向new int的地址,智能指针自动回收

}

自定义智能指针：

Class MyAutoptr

{

Public:

MyAutoptr(A \*ptr)

{

This->ptr = ptr; //此时ptr = new A(10)

}

~MyAutoptr()

{

If(this->ptr != null )

{

Delete this->ptr;

This->ptr = null;

}

}

A\* Operator->()

{

Return this->ptr;

}

A& operator\*()

{

Return \*(this->ptr);

}

Private：

A \*ptr

}

Void test()

{

MyAutoptr myp(new A(10));

myp->func(); //myp.ptr->func()

(\*myp).func(); //(\*(myp->ptr)).func()

}