c++复习纲要

本文按题型整理分析了c++的考点

改错

函数

函数传递参数方式

注意，如果函数的参数列表为引用的时候，调用它的时候不用加&

```
#include <iostream>
void swap1(int &a,int &b)
{
    int temp = b;
    b = a;
    a = temp;
}
void swap2(int *a,int *b)
{
    int temp = *b;
    *b = *a;
    *a = temp
}

int main()
{
    int a=5,b=6;
    swap1(a,b);
    swap1(&a,&b); //error!
    swap2(&a,&b);
}
```

默认参数

参数是从后往前绑定的，如果要使用默认参数，则使用默认参数的变量的后面的每个变量都要被赋予一个默认值

```
class B{
public:
    B(int c,int a=10, float y)//error
    {
    i=a;
    b=c;
    z=y;}
private:
    int i,b;
    float z;
};
```

```
class B{
public:
    B(int c,int a=10, float y=0)//right
    {
    i=a;
    b=c;
    z=y;}
private:
    int i,b;
    float z;
};
```

指针

---

const关键字

const修饰指针和变量

被const关键字修饰的变量只能在初始化时赋值，赋值后不能改变它的值

const关键字修饰时采用就近原则

```
int main()
{
    int a=5,b=6;
    cout << a << b << endl;
    int *pa = &a;
    const int *pb = &b; // 这里int离const最近，修饰的是int，因此不能改变数值
    *pb=7;// error!
    pb = &a;
    int *const pc = &b; // 这里离pc最近，修饰的是指针，因此不能改变地址
    pc = &a;// error!
    *pc = a;
    cout << a << *pb << *pc << endl;
    return 0;
}
```

const修饰成员变量

如果const修饰类中的参数，那么初始化该类的时候，赋值只能通过列表赋值，顺带一提，引用也要

```
#include<iostream>
using namespace std;

class A
{
private:
    const int a;
    int &b;
public:
    A(int b){
        a = b; // error!
```

```
    };
    A(int b,int c):a(b),b(c){
        // right
    };
    void print(){

        cout << a <<endl;
    }
};

int main()
{
    A a(2);
    a.print();
    return 0;
}
```

const修饰成员函数

const修饰的成员函数只能调用const修饰的成员函数，并且不能修改任何成员变量

```
#include<iostream>
using namespace std;

class A
{
private:
    const int a;
public:
    A(int b):a(b){

    };
    void b(){

    }
    void print() const {
        cout << a <<endl;
        a = 10; // ERROR!
        b(); // ERROR!
    }
};
int main()
{
    A a(2);
    a.print();
    return 0;
}
```

static关键字

修饰类的成员变量

静态成员变量不属于任何一个对象，对它的初始化需要在类外进行，访问也只能用`类名::标识符`

```
class A
{
private:
    int a;
    static int c;
public:
    A(int b){
        c = b; //wrong
    };
};

int A :: c = 1; // right

int main()
{
    return 0;
}
```

修饰类的成员函数

静态成员函数主要用来处理静态成员变量，关于静态成员变量请看读程序题型中的阐述

静态成员函数可以直接使用静态成员变量，而对非静态成员变量则只能通过对象访问，静态成员函数可以通过`对象.函数`访问，也可以通过`类名::函数`访问

```
class A
{
private:
    int a;
    static int c;
public:
    A(int b){
        a = b;
    }
    void b(){

    }
    static void print(A p) {
        cout << a << endl; // wrong
        cout << p.a << endl; // right
        cout << c << endl;
    }
};

int A :: c = 1;

int main()
{
    A a(2);
```

```
    a.print(a);
    A::print(a);
    A b(3);
    b.print(b);
    return 0;
}
```

类和对象

在没有声明的时候，类里的成员变量和成员函数都默认为private类型，私有类型不允许在类外访问，protected类能够在子类中访问

```
class A
{
    int a;
};


int main()
{
    A a;
    cout << a.a << endl; //wrong
    return 0;
}
```

构造函数

构造函数在每次创建***新的对象***的时候执行，注意只有在新的对象被创建才会有，比如你只创建一个指针是不会运行的哦！这里可以出读程序，具体看后面的解析

构造函数和类同名，无返回值

```
class A
{
private:
    int a;
public :
    int A(){

    } //error
    A(){
    } //right
};
```

缺省构造

缺省构造在无自定义的构造函数的时候默认有，有自定义的时候就无

```
class A
{
private:
    int a;
public:
    A(int b){
        a = b;
    }
};

int main()
{
    A a; // error!
    return 0;
}
```

一般缺省构造的时候不用加（），但是在使用new动态分配时可以加

```
class A
{
private:
    int a;
};

int main()
{
    A a(); //error!这样是在声明一个返回值类型为A的函数
    A a;
    A *p = new A(); // 对的
    return 0;
}
```

析构函数

析构函数只在对象被销毁的时候才会调用，值得注意的是，如果使用new创建对象的数组，那么删除时只有调用delete []才能调用析构函数，删除对象

```
class A
{
    int a;
public :
    A(){
        cout << "西电启动:"  << endl;
        a++;
    }
    ~A(){
        cout << "回来吧,西电,我最骄傲的信仰" << endl;
    }
};
```

```
int main()
{
    A *p = new A[5];
    delete p; // 是错的但是编译器无法报错
    delete [] p; // 对的
    return 0;
}
```

友元函数

友元函数可以访问朋友的私有成员，并且友元函数在定义时不需要添加类的声明

```
class A
{
public :
    friend void c();
};
void c() //right
{

}
A::void c() // error!
{

}
```

命名空间

可以自定义命名空间，命名空间内可以有自己定义的函数、变量、类等，要调用命名空间的函数之类的，使用`using 命名空间的名字::函数名/变量名/类名`即可，如果要全局引用则使用`using namespace 命名空间的名字`

```
namespace MyNamespace
{
    int a=10;
    void c(){};
    class m
    {
    };
}
using namespace MyNamespace; //right
using namespace std; //right
using MyNamespace::a; //right
using namespace MyNamespace::a; //wrong!
```

继承、多态与虚函数

继承

如果父类无缺省的构造函数，那么子类必须实现父类的构造函数，并且是通过列表赋值的方法，这是因为父类必须比子类先建立而后于子类销毁，此处可以有读程序的题，请见后文

```
class A
{
    int a;
public :
    A(int b ){
        a = b;
        cout << "西电启动:"  << endl;
    }
    ~A(){
        cout << "回来吧,西电,我最骄傲的信仰" << endl;
    }
};

class B : public A
{
    B(){
        A(1);//wrong
    }
    B():A(1){//RIGHT

    }
};
```

多态

在c++中可以使用多种方法实现多态，值得注意的是，如果一个函数已经在子类中被重写了，那么子类只能调用该被重写的函数而不能调用父类中的函数，非常值得注意的是！重写时如果修改了参数列表，则就是一个新的函数，父类无法调用，因此，override重写时函数名，函数的参数列表都必须一致

```
class A
{
public :
    virtual void speak(){
        cout << "a" << endl;
    }
};

class B : public A
{
public:
    void speak(int m){//参数列表不同，实际是在创建一个新的函数
        cout << "b" << endl;
    }
};

int main()
{
```

```
    A * a = new B;
    a->speak(); // 可以！因为它本身是父类指针
    a->speak(2); // 不行，编译器只知道a是父类指针，不知道它指向子类对象，无法调用！
    B b;
    b.speak(); //error!speak已经被重载，无法调用父类的无参数的speak()
    return 0;
}
```

纯虚函数

如果一个类中含有纯虚函数，那么它就是一个抽象类，抽象类无法进行实例化

纯虚函数一般长这样：`virtual 返回类型 函数名 = 0`

```
class A
{
    int a;
public :
    A();
    void speak();
    void f();
    virtual void m()=0;
};
int main()
{
    A a; //error!抽象类无法被实例化
}
```

模板

模板的参数必须是编译时可以确认的常量

```
template <class T,int i> class Array{
    int sz;
public:
    Array():sz(i){

    }
};

int main()
{
    int x=100;//error!x不是常量
    Array<int,x> a;
    const int y=10;
    Array<int,y> b;//right
    Array<int,10> c;//right
}
```

并且，模板类在使用时一定要声明参数类型，模板类函数在类外定义时也需要再次声明模板类

```
template <class T> class Array{
    int sz;
public:
    Array():sz(i){

    }
    void f();
};
template <class T>//如果没有就是错误的
void Array<T>:: f(){

};
```

异常

异常处理时，默认的处理程序`catch(...)`会把其它程序都屏蔽，因此它一定要放在最后

```
int main() {
    try {
        int a = 9;
        throw a;
    }
        catch (...) {
        // 这个块将捕获其他类型的异常
        cout << "Caught an exception" << endl;
    }
    catch (int k) {
        // 这个块将捕获int类型的异常
        cout << "Caught int: " << k << endl;
    }
// error!，catch(...)将catch(int k)屏蔽
    return 0;
}
```

读程序

static关键字

修饰局部变量

static关键字修饰变量，变量被创建时只开辟一次空间，不会因为函数返回失效也不会产生多个副本（用起来有点像全局变量，顺带死去的记忆发起攻击，在OS一门课中，我们学习到全局变量和静态变量是存储在哪里的？局部变量又是存储在哪的？）

注意，当它修饰类的成员变量时不能用初始化列表

```
#include<iostream>
using namespace std;
```

```
class A
{
private:

public:
    void print() {
        static int count = 0;
        count ++;
        cout << count <<endl;
    }
};
int main()
{
    A a;
    a.print();
    A b;
    b.print();
    return 0;
}
```

输出

```
1
2
```

构造函数

构造函数只有在创建新对象的时候才会被调用

```
class A
{
private:
    static int a;
public :
    A(){
        cout << "西电启动:" << a << endl;
        a++;
    }
};

int A::a = 0;

int main()
{
    A *p; //启动失败
    A a; //启动成功
    A *b = new A; //启动成功
    return 0;
}
```

输出

```
西电启动:0
西电启动:1
```

析构函数

一个类仅有一个析构函数！析构函数无返回值无参数，和类同名但是前面要加~

析构函数在对象消亡的时候被自动调用，但是如果用***new***生成对象的话，则需要调用***delete***才能够调用析构函数

```
class A
{
    int a;
public :
    A(){
        cout << "西电启动:"  << endl;
        a++;
    }
    ~A(){
        cout << "回来吧,西电,我最骄傲的信仰" << endl;
    }
};


int main()
{
    A a;
    A *p = new A;
    delete p; //如果注释掉它会调用几次?
    return 0;
}
```

输出(未注释)

```
西电启动:
西电启动:
回来吧,西电,我最骄傲的信仰
回来吧,西电,我最骄傲的信仰
```

输出（注释）

```
西电启动:
西电启动:
回来吧,西电,我最骄傲的信仰
```

模板

继承、重载、多态与虚函数

继承

父类于子类先建立而后于子类被销毁

```
class A
{
    int a;
public :
    A(){
        cout << "西电启动:"  << endl;
    }
    ~A(){
        cout << "回来吧,西电,我最骄傲的信仰" << endl;
    }
};

class B : public A
{
public:
    B(){
        cout << "牢大,我们想你了" << endl;
    }
    ~B(){
        cout << "XDU,out!" << endl;
    }
};


int main()
{
    B a;
    return 0;
}
```

输出:

```
西电启动:
牢大,我们想你了
XDU,out!
回来吧,西电,我最骄傲的信仰
```

多态

父类指针指向子类对象时，如果父类中的虚函数被重写，则会调用子类中重写的函数

```
class A
{
```

```
public :
    virtual void speak(){
        cout << "a" << endl;
    }
};

class B : public A
{
public:
    void speak()override{ // 如果改变参数列表则是一个新的函数，考虑输出会怎么样？
        cout << "b" << endl;
    }
};


int main()
{
    A * a = new B;
    a->speak();
    B b;
    b.speak();
    return 0;
}
```

输出

```
b
b
```

改变参数列表

```
a
b
```

函数重载

经典函数重载，一般来说，在一堆同名函数里会选择类型转化最少的那个，模板类是最后选的，实在没人匹配了就选它

```
class A
{
    int a;
    public:
    A(int b):a(b){} //必须有这个构造函数
};

void foo(int a,int b){
    cout << "int " << endl;
}
void foo(A a,A b){
    cout << "A" << endl;
}
```

```
void foo(int& a,int &b){
    cout << "int& " << endl;
}
void foo(A& a,A& b){
    cout << "A&" << endl;
}
template<class T> void foo(T a,T b){
    cout << "T" << endl;
}
int main()
{
    A a(5);
    A b(3);
    foo(a,3.5); //由于有构造函数的存在，因此会考虑把3.5转化为A类，然后就会选第二个
    foo(a,b); //会报错，因为有两个符合条件的
    foo(2.5,3.5);
    return 0;
}
```

输出

```
A
A
T
```