C++ 期末复习

本人复习要点，仅供参考

0x01 初识

1. 常量

两种定义方法

`#define` 宏常量；`#define 常量名 常量值`

- 通常定义在文件的上方，表示一个定义

`const` 修饰的变量 `const 数据类型 常量名 = 常量值`

- 通常在变量定义前加关键字 const

2. sizeof 关键字

- sizeof 用来统计数据类型所占字符大小

`sizeof(数据类型/变量)`

3. 引用

- 也就是给变量起别名；`数据类型 &别名 = 原名`

这个别名，也会影响原本的变量，看一段代码就懂了。

变量的引用

```
int main(){
    int a = 10;
    int &b = a;
    cout<<a<<b<<endl;
    b = 100;
    cout<<a<<b<<endl;
}
// 运行结果：10 10   100 100
```

引用做函数参数

- 函数的传参，有两种传递，一种是值传递，一种是地址传递，代码如下

```
void trans(int a, int b){
    //do something
}
void trans2(int* a, int* b){
    // do something
}
```

还有一种方式叫做引用传递

```
void mySwap(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
}
int main(){
    int a = 10;
    int b = 20;
    mySwap(a, b);
}
```

地址传递和引用传递都可以修改实参。

引用函数作函数返回值

```
int& test(){
    static int a = 10; // 若不使用 static，则函数调用一次之后变量名便释放
}
int main(){
    int &ref = test();
    cout<<"ref = "<<ref<<endl;

    test = 1000;
    cout<<"ref = "<<ref<<endl;
}
// 输出 10 1000
```

0x02 函数

- 简单理解：方法其实就是面向对象编程中的函数

大致的思想和 Java 差不多，不扯了，主要关注一些考点。

1. 函数默认参数

- 首先，函数的定义模板 `返回值类型 函数名 (参数 = 默认值)`

比较简单的，但是要注意重载时候的二义性。

如果我定义了默认参数，但是又传参了，按照传参的来。

```
int func(int a, int b = 20, int c){
    return a + b + c;
}
int main(){
    func(10,30)
}
// 结果会输出 70
```

2. 函数重载

函数重载需要满足的条件：

- 同一作用域下
- 函数名称相同
- 函数参数类型不同，或个数不同，或顺序不同。

前两个是前提，最后一个是操作方法

注意：函数的返回值不可以作为函数重载的条件，也就是返回类型不一样不能重载

函数重载碰到默认参数会出现二义性，报错。

讲比较干涩，直接看代码

```
void func(int a, int b = 10){
    cout<<a<<b<<endl;
}
void func(int a){
    cout<<a<<endl;
}
int main(){
    func(10);  // 报错
}
```

0x03 面向对象编程

---

1. new 运算符

new 的基本用法

语法：`new 数据类型`

利用 new 创建的数据，会返回该数据对应的类型的指针

按照我们之前学习的语法：`int *p = new int(10);` 这样即可

delete 操作

- new 的根本内涵在于在堆区创建数据。所以这个创建的数据由程序员管理开辟，也有程序员管理释放。

一个完整的程序如下

```
int* func(){
    int *p = new int(10);
    return p;
}
void test(){
    int *p = func();
    cout<<*p<<endl;
    delete p;
    cout<<*p<<endl;  // 这里的 cout 会报错，因为 p 被我们释放了
}
int main(){
    test();
}
```

new 一个数组

```
void test01{
    int *p = new int[10];
    for (int i = 0;i < 10; i++){
        p[i] = i + 100;
    }
    for (int i = 0; i < 10; i++){
        cout<<p[i]<<endl;
    }
    delete [] p; //释放数组的时候加一个中括号即可
}
```

2. 四大特征：抽象，封装，继承，多态

这里和 Java 都差不多吧，就不多提了。

3. 类与结构体的区别

类中成员变量默认为私有，而结构体中则为公有

4. 构造函数与析构函数

- 构造函数自己还是比较清楚的，析构函数的话，不太懂。

从程序的完整性角度来说，我们需要一个构造函数与析构函数，所以 c++ 当中，已经自动帮我们写好了构造函数和析构函数，我们可以"重写"它们，如果不重写，则使用默认的空函数。

> 构造函数相当于手机的初始化
>
> 析构函数则是恢复参数设置，清理手机

构造函数的语法 `类名(){}`，构造函数可以由参数，因此可以发生重载。

析构函数的语法 `~类名(){}`，析构函数不可以有参数，因此不可以发生重载。

- 构造函数和析构函数是不需要调用的，是自动调用的。

5. 对象指针

对象指针，也叫 this指针

- 为什么要使用 this指针，这样做到底有什么好处？

(1)能够解决名称冲突

(2)用 `*this` 返回对象本身

使用方法 `this->变量 = 传参`

解决变量名称冲突

```
class Person{
    public:
    Person(int age){
        this->age = age;   // 如果这里不用对象指针，则会报错
    }
    int age;
}
int main(){
    Person p1(18);
    cout<<"年龄为:  "<<p1.age<<endl;
}
```

返回对象本身，要用引用的方式做一个返回

```
Person& PersonAddAge(Person &p){
    this-> age += p.age;
    return *this
}
int main(){
    Person p1(10);
    Person p2(10);


 p2.PersonAddage(p1).PersonAddage(p1).PersonAddAge(p1);
    cout<<"p2 的年龄为:  "<<p2.age<<endl;
}
```

6. 友元类

友元类的关键字为 `friend`

简单来说，让一个函数或者类访问另外一个类中的私有成员。

友元类的全局函数实现

- 在类当中添加 `friend 函数类型 函数名(参数)`，那么这个函数就是该类当中的友元类

让类做友元类

让一个类可以访问另外一个类中的私有成员。

- 实现方法：在需要访问的那个类中添加 `friend class 赋予访问权限的类`

成员函数做友元

类似的实现方法，在类当中添加 `friend 数据类型 函数名`

7. 继承

- 继承基本语法

在 Java 里面是 这样的 `class class_son extends father_class`

在 c++ 里面是这样的 `class class_son:public fath_class`

8. 纯虚函数

- 纯虚函数和抽象类，以及继承，密不可分。

使用方法 `virtual 返回值类型 函数名 (参数列表) = 0`

比起虚函数来说，多了个结尾 = 0

> 当类中有了纯虚函数，这个类也称为抽象类

所以子类要重写抽象类中的纯虚函数。