尽管我们的手中空无一物 !

@[toc]  


# 函数提高

## 函数默认参数

在C++中，函数的形参列表中的形参是可以有默认值的。

语法：返回值类型 函数名 （参数= 默认值）{}

1. 如果某个位置参数有默认值，那么从这个位置往后，从左向右，**必须都要有默认值**
2. 如果函数**声明有默认值**，函数实现的时候就不能有默认参数

**示例：**

int func(int a, int b = 10, int c = 10) {  
 return a + b + c;  
}  
  
  
int func2(int a = 10, int b = 10);  
int func2(int a, int b) {  
 return a + b;  
}  
  
int main() {  
  
 cout << "ret = " << func(20, 20) << endl;  
 cout << "ret = " << func(100) << endl;  
  
 system("pause");  
  
 return 0;  
}

## 函数占位参数

C++中函数的形参列表里可以有占位参数，用来做占位，调用函数时必须填补该位置

**语法：** 返回值类型 函数名 (数据类型){}

在现阶段函数的占位参数存在意义不大，但是后面的课程中会用到该技术

**示例：**

//函数占位参数 ，占位参数也可以有默认参数  
void func(int a, int) {  
 cout << "this is func" << endl;  
}  
  
int main() {  
  
 func(10,10); //占位参数必须填补  
  
 system("pause");  
  
 return 0;  
}

## 函数重载

### 函数重载概述

**作用** ： 函数名可以相同，提高复用性

**函数重载满足条件：**

* 同一个作用域下
* 函数名称相同
* 函数参数**类型不同** 或者 **个数不同** 或者 **顺序不同**

**注意:** 函数的返回值不可以作为函数重载的条件

**示例：**

//函数重载需要函数都在同一个作用域下  
void func()  
{  
 cout << "func 的调用！" << endl;  
}  
void func(int a)  
{  
 cout << "func (int a) 的调用！" << endl;  
}  
void func(double a)  
{  
 cout << "func (double a)的调用！" << endl;  
}  
void func(int a ,double b)  
{  
 cout << "func (int a ,double b) 的调用！" << endl;  
}  
void func(double a ,int b)  
{  
 cout << "func (double a ,int b)的调用！" << endl;  
}  
  
//函数返回值不可以作为函数重载条件  
//int func(double a, int b)  
//{  
// cout << "func (double a ,int b)的调用！" << endl;  
//}  
  
  
int main() {  
  
 func();  
 func(10);  
 func(3.14);  
 func(10,3.14);  
 func(3.14 , 10);  
   
 system("pause");  
  
 return 0;  
}

### 函数重载注意事项

* 引用作为重载条件
* 函数重载碰到函数默认参数

**示例：**

//函数重载注意事项  
//1、引用作为重载条件  
  
void func(int &a)  
{  
 cout << "func (int &a) 调用 " << endl;  
}  
  
void func(const int &a)  
{  
 cout << "func (const int &a) 调用 " << endl;  
}  
  
  
//2、函数重载碰到函数默认参数  
  
void func2(int a, int b = 10)  
{  
 cout << "func2(int a, int b = 10) 调用" << endl;  
}  
  
void func2(int a)  
{  
 cout << "func2(int a) 调用" << endl;  
}  
  
int main() {  
   
 int a = 10;  
 func(a); //调用无const  
 func(10);//调用有const  
  
  
 //func2(10); //碰到默认参数产生歧义，需要避免  
  
 system("pause");  
  
 return 0;  
}

**func(a); //调用无const**  
**func(10);//调用有const**

**原因解释** ：

int a=10  
int & a = 10 合法

直接传入 10  
const int &a = 10

int& ref = 10; 引用本身需要一个合法的内存空间，因此这行错误   
加入const就可以了，编译器优化代码，  
int temp = 10;  
const int& ref = temp;

# 类和对象 （核心）

* C++面向对象的三大特性为：封装、继承、多态
* 对象具有属性和行为

## 封装

**封装的概念**：

* 将属性和行为作为一个整体，表现生活中的事物
* 类中的属性和行为我们统一称为**成员**
* 将属性和行为加以权限控制
* 属性别名 成员属性、成员变量
* 行为别名 成员函数、成员方法

**封装意义一：**

在设计类的时候，属性和行为写在一起，表现事物

**语法：** class 类名{ 访问权限： 属性 / 行为 };

**封装意义二：**

类在设计时，可以把属性和行为放在不同的权限下，加以控制

访问权限有三种：

1. public 公共权限
2. protected 保护权限
3. private 私有权限

* 公共权限 public 类内可以访问 类外可以访问
* 保护权限 protected 类内可以访问 类外不可以访问
* 私有权限 private 类内可以访问 类外不可以访问

#include <iostream>  
using namespace std ;  
#include <string>  
  
  
double PI = 3.14;  
  
class Circle {  
public :  
  
 int r ;  
 double fun() {  
 return 2 \* PI \* r;  
 }  
};  
  
class Student {  
public :  
 int age;  
 string name;  
  
  
 void setName(string name) {  
 name = name;  
 }  
  
 void setAge(int age) {  
 age = age;  
 }  
  
 void Print() {  
 cout << "姓名= " << name << "年龄=" << age<<endl;  
 }  
  
   
};  
  
int main() {  
  
 Circle c1;  
 c1.r = 10;  
  
 cout << "圆的周长= " << c1.fun() << endl;  
  
 cout << "我是分节符" << endl;  
  
 Student stu1;  
 stu1.age = 18;  
 stu1.name = "花阳";  
 stu1.Print();  
  
 system("pause");  
 return 0;  
}

## 关键字struct和class区别

在C++中 struct和class唯一的**区别**就在于 **默认的访问权限不同**

区别：

* struct 默认权限为公共
* class 默认权限为私有

## 成员属性设置为私有

**优点1** ：将所有成员属性设置为私有，可以自己控制读写权限

**优点2**：对于写权限，我们可以检测数据的有效性

## 关于多个类

* 1、在类中可以让另一个类作为本来中的成员
* 2、可以将一个文件中的多个类，分解（头文件、源文件）

Circle :: setR(int r) 这里的语法表示成员函数作用域