**C++发展历史**

【概览】

1.C++关键字

2.命名空间

3.C++输入&输出

4.缺省函数

5.函数重载

6.引用、

7.内联函数·

auto关键字（C++11）

9.基于范围的for循环（C++11）

10.指针空值--nullptr

C的三方冲突

库

我

同事

C++解决方法：域作用限定符“::”

namespace Hanluo

{

int char = 'a';

int rand = 0;

}

int mian(void)

{

printf("%d\n",Hanluo::rand);

return 0;

}

命名空间可以定义函数、变量、类型

命名空间展开：

全部展开和部分展开

using namespace Hanluo;//全部展开（授权）

using Hanluo::rand;//部分展开（授权）

当多个头文件中创建不同的命名空间时，在.c文件中引用这些头文件全不会产生冲突，编译器会主动合并同一个命名空间，因此，（不同头文件同一命名空间不能存在同一个变量名或函数）

留提取和流输出

#include<iostream>

endl

缺省参数

全缺省、

半缺省

不允许声明和定义同时给缺省参数

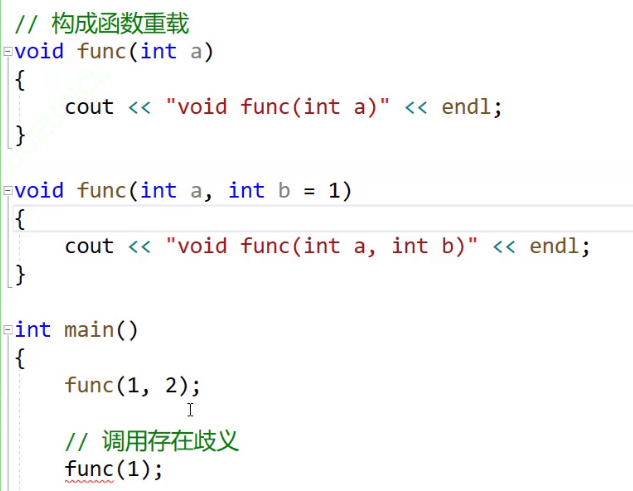
声明给，定义不给

因为在头文件中一班为声明，头文件一般不会重复，所以大工程后期会做去重

static修饰函数，使得当前函数只能在该文件中使用

重载：一词多义

重载前提：函数名可以相同，但是参数类型，数量，顺序必须不同；且在同一个作用域





**函数重载原理**

对同一个函数名的函数，根据不同的使用方法，在编译过程中使用不同的地址编号，做到对函数的区分

**引用**

**概念：**

引用不是定义一个变量，而是给已经存在的变量取一个别名，**编译器不会为引用变量开辟内存空间，**他和被他引用的变量公用一块空间

**使用方法：**

在类型紧邻的位置加上“&”，这样便算作引用。

//举例

int mian(void)

{

int a = 0;

int& b = a;//引用

cout << &a << endkl;\\取地址符

cout << &b << endl;

}

typedef struct ListNode

{

int data;

struct ListNode\* next;

}ListNode,\*PListNodde;

void PushBack(ListNode\*& phead,int x)\\引用传参，可以省略二级指针

{

//~~~~

}

void PushBack(PListNode& phead,int x)

{

//~~~

}

**特性：**

1.引用在定义时必须进行初始化

2、一个变量可以有多个引用

3、引用一旦引用一个实体，就不能再引用其他实体

void Text()

{

int a = 7;

// int& ra; //该语句会引起编译报错

int& ra = a;

int& rra = a;

printf("%p %p %p\n",&a, &ra ,&rra);

}

**常引用**

void Text(void)

{

const int a = 10;

//int& ra = a; // 该语句编译时会出错，a为常量

const int& ra = a;

// int& b = 10; // 该语句编译时会出错，b为常量

const int& b = 10;

double d = 12.34;

//int& rd = d; // 该语句编译时会出错，类型不同const int& rd = d

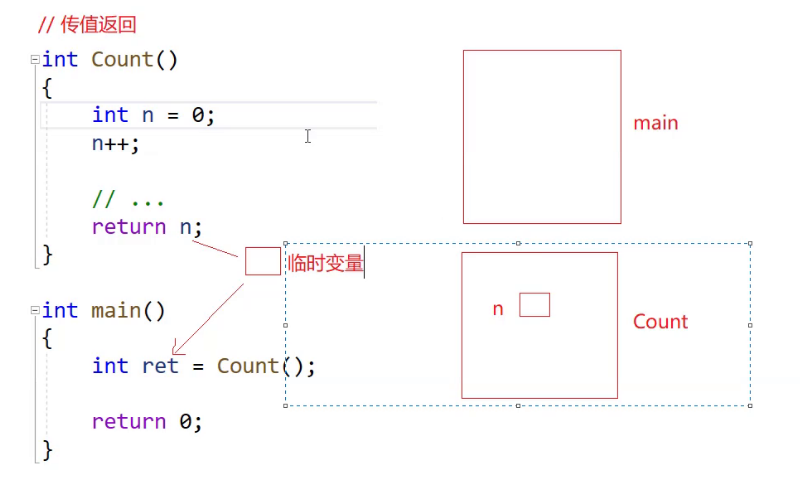
}

**使用场景**



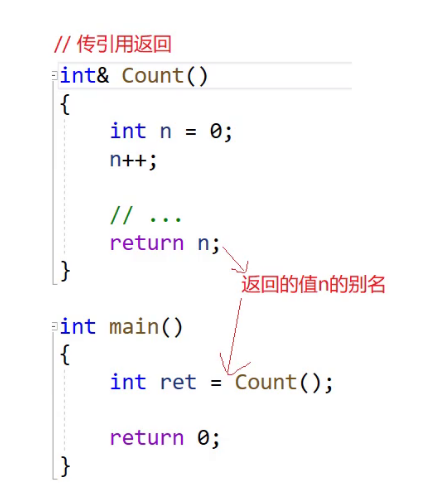
传值返回：

在栈帧中创建一块区域或者使用寄存器将变量进行临时存储，避免出了作用域后消失的情况。



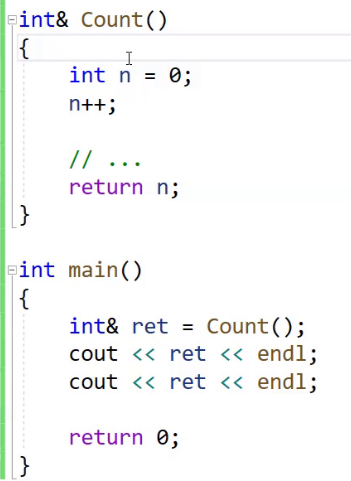
传引用返回：

当函数栈帧销毁时，空间仍然存在，我们使用“引用”这把钥匙去访问这个空间，可能出现无法预知的后果。类似于野指针，在房间1退还之前留存了一把钥匙。



函数调用先传参

所以调用以下代码时会出现不同的值的情况



因为之前的空间被覆盖，覆盖之前采用一次，覆盖之后再采用一次

还有如下

