

C++程序设计

电子科技大学

余盛季

主讲课程

程序设计、程序设计语言、编译原理、云计算

课程安排

学时分配

总学时（32）： 课堂讲授（16） + 上机实验（16）

成绩构成

课后作业（码图评分， 20分）

课堂实践（码图评分， 20分）

精灵游戏（人工评分， 10分）

期末考试（码图评分， 50分）

教材

C与C++程序设计（戴波）

参考书

The C++ Programming Language（Bjarne Stroustrup）

C++ Primer（Stanley B. Lippman）

课程MOOC











www.icourse163.org/course/UESTC-1001774006

什么是C++

C + 面向对象 + 其它

为什么要学习C++

<https://www.tiobe.com/tiobe-index>

Sep 2022	Sep 2021	Change	Programming Language		Ratings	Change
1	2	▲		Python	15.74%	+4.07%
2	1	▼		C	13.96%	+2.13%
3	3			Java	11.72%	+0.60%
4	4			C++	9.76%	+2.63%
5	5			C#	4.88%	-0.89%
6	6			Visual Basic	4.39%	-0.22%
7	7			JavaScript	2.82%	+0.27%
8	8			Assembly language	2.49%	+0.07%
9	10	▲		SQL	2.01%	+0.21%
10	9	▼		PHP	1.68%	-0.17%

如何学习C++

读代码 + 写代码 + 调代码

实验环境

编辑器 + 编译器 + 调试器

编辑器

Visual Studio Code

下载地址: [Visual Studio Code](#)

安装说明: [C/C++ for Visual Studio Code](#)

插件安装: C/C++、Marp for VS Code

编译器

GCC (GNU Compiler Collection)

下载地址: [MSYS2 Distribution](#)

安装说明: [MSYS2 Installation](#)

扩展阅读: [GCC参数详解](#)

调试器

GDB (The GNU Project Debugger)

下载地址: [MSYS2 Distribution](#)

安装说明: [MSYS2 Installation](#)

扩展阅读: [C++ GDB调试大全](#)

调试实例

调试选项: `gcc -g *.c -o test.exe`

启动调试: `gdb test.exe`

退出调试: `quit`

设置断点: `break, info, del, enable, disable, watch ...`

程序执行: `run, next, step, continue, finish, set, call ...`

程序状态: `print, display, backtrace, list ...`

云端开发

场景一

进入新部门，新环境不熟悉，不会独立搭建开发环境.....

场景二

项目临时需要搭建开发环境，搭建过程复杂，用完即删.....

场景三

下班后系统出现问题需要紧急处理，开发人员无法及时赶回公司.....

场景四

不同工作地点、不同操作系统平台的开发人员合作进行项目开发.....

本地开发

在本地环境中进行开发。

云端开发

在云端环境中进行开发（云——就是网络）。

是对本地开发的补充，而不是代替。

开发环境

华为云开发环境：[CloudIDE](#)、[CloudHub](#)

C++程序实例

第一个C++程序

```
int main()  
{  
    return 0; /* 1、 2、 3..... */  
}
```

编译： g++ test.cpp -o test.exe

执行： test.exe

查看结果（返回值）：

```
echo %errorlevel%
```

第二个C++程序

```
#include <stdio.h>
int main()
{
    printf("Hello, I'm a C++ program.\n");
    return 0;
}
```

库函数: printf

头文件: [stdio.h](#)

第三个C++程序

```
#include <iostream>
using namespace std;
int main()
{
    cout << "Hello, I'm really a C++ program." << endl;
    return 0;
}
```

输出流对象: cout

头文件: [iostream](#)

输入与输出

输出流对象：cout

左移运算符：<<

```
int a = 1, b = 2;  
printf("a + b = %d \n", a + b);  
  
int c = 3, d = 4;  
cout << "c + d = " << c + d << endl;
```

扩展阅读：[cout格式化输出](#)

输入流对象：cin

右移运算符：>>

```
int a, b;  
scanf("%d %d", &a, &b);  
printf("a + b = %d \n", a + b);  
  
int c, d;  
cin >> c >> d;  
cout << "c + d = " << c + d << endl;
```

扩展阅读：[cin深入分析](#)

数据类型

逻辑类型 + 引用类型 + 类类型

逻辑类型

```
int main()
{
    bool a = true;
    bool b = false;
    cout << "a = " << a << ", " << "b = " << b << endl;

    int i=a;
    int j=b;
    cout << "i = " << i << ", " << "j = " << j << endl;

    a = !i;
    b = !j;
    cout << "a = " << a << ", " << "b = " << b << endl;
    return 0;
}
```

思考：下列表达式的值

```
(int) true  
(int) false  
(bool) 0  
(bool) 1  
(bool) 2  
(bool) -1  
(bool) -2  
a+1  
a+b  
a+1+b  
a+b+1
```

不同的编译器与调试器有不同的处理方式。

引用类型

独立引用

引用就是别名。

```
int number = 10;  
int & n = number;  
n = 20;  
cout << number << " " << n << endl;
```


引用的初始化

必须在定义的时候初始化，之后一直作为该变量的别名，不能再作为其它的变量的别名。

```
int a=1, b=2;  
int &n = a;  
n = b;  
n = 3;  
cout << a << " " << b << " " << endl;
```

如何验证引用是另一个变量的别名？

常引用

普通引用不能作为常量的别名。

常引用可以作为常量的别名。

```
const int & n = 1;
```

或

```
int const & n = 1;
```

常引用也可以作为变量的别名。

```
int a=1;  
const int & n = a;
```

或

```
int a=1;  
int const & n = a;
```

无论怎样，常引用的值都不能修改（编译时）。

```
n = 3; /* 错误 */
```

常量与宏

思考：常量与宏的区别

```
#define PI 3.14

int main()
{
    float const pi=3.14;
    cout << PI << " " << pi << endl;
    return 0;
}
```

不同的编译器与调试器有不同的处理方式（-g3、macro expand）。

常量与指针

```
int main()
{
    int x=0, y=0;
    int * const p1=&x;
    // p1=&y;
    // *p1=9;
    int const * p2=&x;
    // p2=&y;
    // *p2=9;
    int const * const p3=&x;
    // p3=&y;
    // *p3=9;
    cout << x << " " << y << endl;
    return 0;
}
```

几个容易混淆的概念

指针常量：指针类型的常量（别乱指）

常量指针：指向常量的指针（别乱动）

两者相加：指向常量的指针常量（别乱指 + 别乱动）

函数参数

普通变量作为参数

```
void swap(int x, int y)
{
    int tmp;
    tmp=x;
    x=y;
    y=tmp;
}
```

指针变量作为参数

```
void swap(int * x, int * y)
{
    int tmp;
    tmp=*x;
    *x=*y;
    *y=tmp;
}
```

引用作为参数

```
void swap(int & x, int & y)
{
    int tmp;
    tmp=x;
    x=y;
    y=tmp;
}
```

引用作为形参时，在函数调用时用实参对其进行初始化。

函数返回值

返回普通类型（右值， r-value）

```
int x=1;
int f()
{
    x=2;
    return x;
}
int main()
{
    int a=3;
    a = f();
    cout << a << endl;
}
```

右值代表一个“值”，只能放在赋值符号的右边。

返回引用（左值，l-value）

```
int x=1;
int & f()
{
    x=2;
    return x;
}
int main()
{
    int a=3;
    f() = a;
    cout << x << endl;
}
```

左值代表一个“内存单元”，可以放在赋值符号的左边和右边。（思考）

内联函数

```
inline int add(int value)
{
    int x;
    x = value + 1;
    x = x + 2;
    x = x + 3;
    return x;
}

int main()
{
    int a,b;
    cin >> a;
    b=add(a);
    cout << b << endl;
    return 0;
}
```


内联与编译优化

inline仅仅是程序员对编译器提出的一个优化建议，是否采纳，还要看编译器自己。

即使不提这个建议，编译器也可能在适当的时候自动进行内联。

编译优化选项： `-O`

```
g++ -O test.cpp -o test.exe
```

思考：验证函数是否被内联

函数重载

```
int GetMax2(int x, int y)
{
    if(x>y)
        return x;
    else
        return y;
}
```

```
int GetMax3(int x, int y, int z)
{
    int tmp1,tmp2;
    tmp1=GetMax2(x,y);
    tmp2=GetMax2(tmp1,z);
    return tmp2;
}
```

在编译的过程中，C++ 编译器会对函数名做一些修改。

C 编译器：

```
gcc test.c -o test.exe
nm test.exe | find "GetMax"
004015c0 T GetMax2
004015d5 T GetMax3
```

C++ 编译器：

```
g++ test.cpp -o test.exe
nm test.exe | find "GetMax"
004015c0 T _Z7GetMax2ii
004015d5 T _Z7GetMax3iii
```

C++ 编译器的修改方式，使得源代码中即使使用了两个完全相同的函数名，在编译的过程中也不会发生重名冲突，只要满足以下两个条件之一即可：

1. 参数个数不同
2. 参数类型不同

参数个数不同:

```
int GetMax(int x, int y)
int GetMax(int x, int y, int z)
```

修改后的函数名:

```
nm test.exe | find "GetMax"
004015c0 T _Z7GetMaxii
004015d5 T _Z7GetMaxiii
```

参数类型不同：

```
int GetMax(int x, int y)
int GetMax(float x, float y)
```

修改后的函数名：

```
nm test.exe | find "GetMax"
004015d5 T _Z7GetMaxff
004015c0 T _Z7GetMaxii
```

当存在形参个数或类型不同的同名函数，在函数调用时，编译器会根据实参的类型及个数的最佳匹配来选择调用哪一个函数。

通过调试器观察函数何时被调用：

```
(gdb) b *0x004015c0  
Breakpoint 3 at 0x4015c0: file test.cpp, line 5.  
(gdb) b *0x004015d5  
Breakpoint 4 at 0x4015d5: file test.cpp, line 13.
```


默认形参值

可以在定义函数时给出默认的形参值。调用函数时如果给出了实参值，则使用给出的实参值；如果未给出实参值，则使用默认的形参值。

```
int register(int number, char const * name, int age=18, char const * country="China")
{
    cout << "Number: " << number << endl;
    cout << "Name: " << name << endl;
    cout << "Age: " << age << endl;
    cout << "Country: " << country << endl << endl;
    return 0;
}
int main()
{
    register(1, "ZhangSan");
    register(2, "LiSi");
    register(3, "WangWu", 20);
    register(4, "Tom", 18, "England");
    return 0;
}
```

形参的顺序

无默认值的形参在前，有默认值的形参在后。

注意

本身不是函数重载，但可能会和函数重载相互影响。

编译器处理函数调用时，如果不能确定该调用哪个函数，则会出现编译错误；如果能够确定该调用哪个函数，则不会出现编译错误。

```
int register(int number, char const * name, int age=18, char const * country="China")
{
    cout << "Number: " << number << endl;
    cout << "Name: " << name << endl;
    cout << "Age: " << age << endl;
    cout << "Country: " << country << endl << endl;
    return 0;
}
```

```
int register(int number, char const * name)
{
    cout << "Number: " << number << endl;
    cout << "Name: " << name << endl;
    cout << "Age: " << 18 << endl;
    cout << "Country: " << "China" << endl << endl;
    return 0;
}
```

```
int main()
{
    register(1,"ZhangSan"); /* error */
    register(2,"LiSi"); /* error */
    register(3,"WangWu", 20);
    register(4,"Tom", 18, "England");
    return 0;
}
```

动态内存管理

C语言的动态内存分配和释放使用**函数**： malloc()、 free()、

它们来自标准**函数库**。

```
#include <stdlib.h>

int main()
{
    int * a;
    a = malloc(sizeof(int));
    free(a);
    return 0;
}
```

C++语言的动态内存分配和释放使用**运算符**：new、delete

它们是**语言**的组成部分。

```
int main()
{
    int * a;
    a = new int;
    delete a;
    return 0;
}
```


分配和释放一个变量的空间

```
int main()
{
    int * a;
    a = new int;
    *a = 123;
    cout << *a;
    delete a;
    return 0;
}
```

分配和释放一个变量的空间（并初始化）

```
int main()
{
    int * a;
    a = new int(456);
    cout << *a;
    delete a;
    return 0;
}
```

分配和释放一个数组的空间

```
#include <iostream>
#include <string.h>
using namespace std;

int main()
{
    char * a;
    a = new char[10];
    strcpy(a, "hello");
    cout << a;
    delete []a;
    return 0;
}
```

分配和释放一个数组的空间（并初始化）

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

int main()
{
    char * a;
    a = new char[10]{'h','e', 'l', 'l', 'o'};
    cout << a;
    delete []a;
    return 0;
}
```

new与delete必须配合使用

未delete或重复delete，都可能导致程序崩溃。

这与操作系统的内存管理方式紧密相关。

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

void func(int i)
{
    char *p;
    p=new char[1000000];
    cout << i << endl;
    delete []p;
}
```

```
int main()
{
    for(int i=1; i<=4000; i++){
        func(i);
    }

    cout << "OK" << endl;
    return 0;
}
```

课后复习

课程MOOC

www.icourse163.org/course/UESTC-1001774006

第七章 C++基础

码图作业

matu.uestc.edu.cn

第7章 作业1

第7章 作业2