

面向对象程序设计

C/C++ 程序设计基础

2020 年 春

耿楠

计算机科学系
信息工程学院

西北农林科技大学
NORTHWEST A&F UNIVERSITY

中国·杨凌



OBJECT
ORIENTED
PROGRAMMING—
OOP

知识回顾

对 C 的扩充

输入输出

cin/cout

格式控制

类型与表达式

基本类型

新类型

运算符与表达式

控制结构

基本结构

范围 for 语句

复杂类型

指针与内存

指针与常量

引用类型

C++ 类类型

类型转换

函数

内联函数

constexpr 函数

形参默认值

重载函数

函数模板

系统函数

程序结构

1

▶ 对象、类及其特性

▶ 什么是对象

▶ 什么是类

▶ 四大特性（数据抽象、封装、继承和多态）

▶ 面向对象程序设计语言发展史

▶ 基本 C++ 程序

▶ cin 和 cout

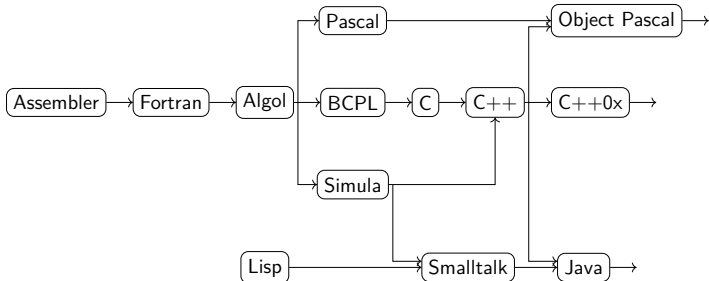
▶ using namespace

62



OBJECT
ORIENTED
PROGRAMMING—
OOP

► 族谱 (Family tree)



知识回顾

对 C 的扩充

输入输出

cin/cout
格式控制

类型与表达式

基本类型
新类型
运算符与表达式

控制结构

基本结构
范围 for 语句

复杂类型

指针与内存
指针与常量
引用类型
C++ 类类型
类型转换

函数

内联函数
constexpr 函数
形参默认值
重载函数
函数模板
系统函数

程序结构



OBJECT
ORIENTED
PROGRAMMING—
OOP

知识回顾

对 C 的扩充

输入输出

cin/cout
格式控制

类型与表达式

基本类型
新类型
运算符与表达式

控制结构

基本结构
范围 for 语句

复杂类型

指针与内存
指针与常量
引用类型
C++ 类类型
类型转换

函数

内联函数
constexpr 函数
形参默认值
重载函数
函数模板
系统函数

程序结构

3

▶ Hello, your name!

```
// 例 01-01: ex01-01.cpp
#include <iostream>
using namespace std;
int main( )
{
    char strName[32];
    cin >> strName;
    cout << "Hello, " << strName << "!" << endl;
    return 0;
}
```

▶ 名字空间

▶ using namespace

▶ 文件后缀名

▶ Windows: .cpp

▶ Unix/Linux: .cpp, .cc or .c

62



OBJECT
ORIENTED
PROGRAMMING—
OOP

知识回顾

对 C 的扩充

输入输出

cin/cout

格式控制

类型与表达式

基本类型

新类型

运算符与表达式

控制结构

基本结构

范围 for 语句

复杂类型

指针与内存

指针与常量

引用类型

C++ 类类型

类型转换

函数

内联函数

constexpr 函数

形参默认值

重载函数

函数模板

系统函数

程序结构

4

▶ 格式化输入输出

▶ 基本数据类型与表达式

▶ 控制结构

▶ 构造数据类型

▶ 函数

▶ 大型程序结构控制

62



► cin/cout

```
// 例 02-01: ex02-01.cpp
//求三个数的平均值, 演示 C++ 简单 I/O
#include <iostream>
using namespace std;
int main()
{
    float num1, num2, num3;    //定义三个数

    cout << "Please input three numbers:" ;
    cin >> num1 >> num2 >> num3;

    cout << "The average of " << num1 << ", " << num2 << "and " << num3;
    cout << " is: " << (num1 + num2 + num3) / 3 << endl;

    return 0;
}
```

```
testCpp
Please input three numbers:101 201 300
The average of 101, 201and 300 is: 200.667

Process returned 0 (0x0)   execution time : 10.174 s
Press ENTER to continue.
```



► 格式控制

操作符	作用	说明
oct	数据以 8 进制形式输出	作用范围为后续输出的整数，对小数不起作用
dec	数据以 10 进制形式输出 (默认)	
hex	数据以 16 进制形式输出	
endl	换行并刷新输出流	
setw(n)	设置输出宽度	作用范围为后续对象
setprecision(n)	设置输出精度 (默认为 6)	

注意:

► `#include <iomanip>`

► 默认情况下, `setprecision(n)` 仅对带有小数的数有效, `n` 为整数与小数位数之和



► 设置输出格式

```
// 例 02-02: ex02-02.cpp
// 求三个数的平均值, 演示 C++ 简单 I/O 格式控制
#include <iostream>
#include <iomanip>
using namespace std;
int main()
{
    float num1, num2, num3; //定义三个数

    cout << "Please input three numbers:";
    cin >> num1 >> num2 >> num3;

    cout << setw(20) << setprecision(12);
    cout << "The average of " << num1 << " , " << num2 << " and " << num3;
    cout << " is:" << setw(20) << (num1 + num2 + num3) / 3 << endl;

    return 0;
}
```

```
testCpp
Please input three numbers:101 201 300
The average of 101 , 201 and 300 is: 200.666671753

Process returned 0 (0x0) execution time : 10.591 s
Press ENTER to continue.
```




▶ 基本数据类型

- ▶ `int`, `float`, `double`, `void`, `char`
- ▶ 布尔型: `bool` (`true`⇒1, `false`⇒0)

▶ 变量与常量

▶ 变量的定义与赋初值

- ▶ `int sum=100`; `double pi=3.1416`; `char c='a'`;
- ▶ `int sum(100)`; `double pi(3.1416)`; `char c('a')`;

▶ 符号常量与常变量

- ▶ `#define PI 3.1416`
- ▶ `const float PI=3.1416`;
- ▶ `PI = 3.1415926535898`; `// 错误!`



▶ 常量表达式

```
const int size = 20;           // size 是常量表达式
const int limits = size + 1;   // limits 是常量表达式
int max = 80;                  // max 不是常量表达式, 80 是字面值常量,
                                // 但 max 不是 const, 不保证运行是不变。
const int lines = get_size();  // lines 不是常量表达式
                                // lines 是常量, 但 get_size() 运行时才能确定
```

▶ constexpr 类型 (验证是不是常量表达式)

```
constexpr int size = 20;       // 20 是常量表达式
constexpr int limits = size + 10; // size + 10 是常量表达式
constexpr int max = length();  // 取决于 length() 函数是不是常量函数
```

▶ constexpr 与 const

```
constexpr int a = length(); // 必须在编译时能计算出 length() 返回值
const int b = length();     // b 的值可以在运行时获得, 之后不再改变
```



► auto 类型说明符

```

auto x = 5;           // 5 是 int 类型, x 则是 int 类型
auto size = sizeof(int); // size 是表示内存字节数的类型
auto name = "world";  // name 是保存字符串的类型
cout << "hello " << name; // 可以使用 name
auto a;              // 错误! 没有初始值无法确定类型
auto r = 1, pi = 3.14; // 错误! 类型混淆

```

► decltype 类型指示符, 返回操作数类型

```

decltype(sizeof(int)) size; // sizeof(int) 结果的类型
const int ci = 0;           // 是常量表达式
decltype(ci) x = 1;         // const int 类型
decltype(f()) y = sum;      // 函数 f() 的返回值类型

```



▶ 运算符与表达式

- ▶ 算术运算符：+(正号), -(负号), *, /, %(取余)
- ▶ 关系运算符：>, <, >=, <=, ==, !=
- ▶ 逻辑运算符：!, &&, ||
- ▶ 位运算符：~, <<, >>, &, ^(异或), |
- ▶ 赋值运算符：=, *=, /=, %+=, +=, -=, ...
- ▶ 递增递减运算符：++, --

$$ax^2 + bx + c = 0$$

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

```
if(abs(b * b - 4 * a * c) > 1.0e - 10)
{
    x1 = (-b + sqrt(b * b - 4 * a * c)) / (2 * a);
    x2 = (-b - sqrt(b * b - 4 * a * c)) / (2 * a);
}
```



OBJECT
ORIENTED
PROGRAMMING—
OOP

知识回顾

对 C 的扩充

输入输出

cin/cout
格式控制

类型与表达式

基本类型
新类型
运算符与表达式

控制结构

基本结构
范围 for 语句

复杂类型

指针与内存
指针与常量
引用类型
C++ 类类型
类型转换

函数

内联函数
constexpr 函数
形参默认值
重载函数
函数模板
系统函数

程序结构

▶ 判断

- ▶ `if ... else ...`
- ▶ `if ... else if ... else ...`
- ▶ `switch ... case ...`

▶ 循环

- ▶ `for(exp1;exp2;exp3){...}`
- ▶ `while(exp){...}`
- ▶ `do ... while(exp);`

▶ 转移

- ▶ `break`
- ▶ `continue`
- ▶ `goto`



▶ 范围for语句

```
for(declaration : expression){
    statement;
}
// 其中:
// expression 必须是一个序列 (列表、数组、vector、string 等),
// 能返回 begin 和 end 对象。
// declaration 是一个变量, 序列中每个元素都能够转换为该类型,
// 常用 auto 声明
```

▶ 范围for示例

```
// 累加 20 以内的素数
int sum = 0;
for(int e : {2, 3, 5, 7, 11, 13, 17, 19}) // 用 auto 类型更合理
    sum += e;
cout << sum << endl;
int arr[] = {1, 3, 5, 7, 9};
for(auto ele : arr){
    ele = ele * 2;
    cout << ele << " ";
}
cout << endl;
for(auto ele : arr)
    cout << ele << " ";
cout << endl;

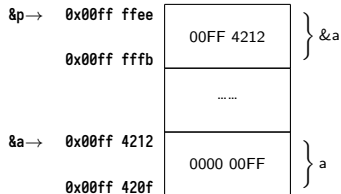
// 输出结果 77
// 声明数组 arr, 初始化为 5 个奇数
// 声明 ele, 与数组 arr 关联在一起, 用了 auto
// 修改数组每个元素的值
// 输出 ele, 2 6 10 14 18

// 没有改变: 1 3 5 7 9
```



► 指针

```
int a=255;
int *p;
p=&a;
```



```
float x[5];
float *p = x;
double sum = 0.0;
for (int i = 0; i < 5; i++)
{
    sum += *p++;
}
```



▶ 动态内存分配

▶ malloc 和 free

▶ new 和 delete

```
// C 语言
float *x = (float *)malloc(n*sizeof(float));
free (x);
```

```
// C++ 语言
float *x = new float[n];
delete []x;
```

```
int **mat;
int m, n;
mat = new int *[m];

for (i = 0; i < m; i++)
    mat[i] = new int[n];

for (i = 0; i < m; i++)
    delete [] mat[i];
delete [] mat;
```




▶ 定位new表达式

▶ 语法: new (指针) 类型

```
#include <iostream>
#include <new> // 必须包含该头文件

using namespace std;

char * buf = new char[1000]; // 预分配空间

int main()
{
    int * pi = new (buf) int; // 在 buf 中创建一个 int 对象

    return 0;
}
```



▶ 指针常量

```
int a = 2, b = 3;  
int * const p = &a; //定义时必须赋初值  
p = &b;             //错误, 地址不能被修改  
*p = b;             //正确, 内容可以被修改
```

▶ 常量指针

```
int a = 2, b = 3;  
const int * p;  
p = &b;             //正确, 地址可以被修改  
*p = b;             //错误, 内容不可以被修改
```

▶ 常指针常量

```
int a = 2, b = 3;  
const int * const p = &a; //定义时必须赋初值  
p = &b;                 //错误, 地址不能被修改  
*p = b;                 //错误, 内容不可以被修改
```



► 引用是已存在的变量的别名

```
int i = 3;
int &j = i;    //引用必须赋初值
int &j = 3;    //错误，初值必须为变量
j = 4;
```

► 引用和指针的区别与联系

```
int i = 3;
int &j = i;
int *k = &i;
cout << &i << endl;
cout << &j << endl;
cout << &k << endl;
```

k →	0x0016	FDEC
	0x0016 FE04
i j →	0x0016	FE04
	0x0000 0003



► 引用作为函数参数（例 1）

```
// 例 02-03: ex02-03.cpp
#include <iostream>
using namespace std;

void swap(int &a, int &b)
{
    int temp = a;
    a = b;
    b = temp;
}

int main()
{
    int m = 3, n = 4;

    cout << "before swap:";
    cout << m << ", " << n << endl;

    swap(m, n);

    cout << "after swap:";
    cout << m << ", " << n << endl;

    return 0;
}
```

```
// 例 02-04: ex02-04.cpp
#include <iostream>
using namespace std;

void swap(int *a, int *b)
{
    int temp = *a;
    *a = *b;
    *b = temp;
}

int main()
{
    int m = 3, n = 4;

    cout << "before swap:";
    cout << m << ", " << n << endl;

    swap(&m, &n);

    cout << "after swap:";
    cout << m << ", " << n << endl;

    return 0;
}
```



► 引用作为函数参数（例 2）

```
struct StuNode
{
    int ID;
    char name[128];
    bool gender;
    int age;
    struct StuNode *next;
};
```

```
// 例 02-05: ex02-05.cpp
void CreateHeadNode(StuNode **pHead)
{
    StuNode *p;
    p = new StuNode;
    if (p == NULL) return;
    p->next = NULL;
    *pHead = p;
}

int main()
{
    StuNode *pHead = NULL;
    CreateHeadNode(&pHead);

    return 0;
}
```



► 引用作为函数参数（例 2）

```
struct StuNode
{
    int ID;
    char name[128];
    bool gender;
    int age;
    struct StuNode *next;
};
```

```
// 例 02-06: ex02-06.cpp
void CreateHeadNode(StuNode *&pHead)
{
    StuNode *p;
    p = new StuNode;
    if (p == NULL) return;
    p->next = NULL;
    pHead = p;
}

int main()
{
    StuNode *pHead = NULL;
    CreateHeadNode(pHead);
    return 0;
}
```



▶ 常引用

```
int i = 100;  
const int &r1 = 100;    //正确  
const int &r2 = i;      //必须初始化  
r2 = 200;               //错误
```

▶ 常引用参数

```
int fun(const int &a, const int &b)  
{  
    return (a + b) / 2;    //参数不能被修改  
}
```



▶ 常引用

```
int i = 100;  
const int &r1 = 100;    //正确  
const int &r2 = i;      //必须初始化  
r2 = 200;               //错误
```

▶ 常引用参数

```
int fun(const int &a, const int &b)  
{  
    return (a + b) / 2;    //参数不能被修改  
}
```




▶ begin() 和 end()

▶ 语法

```
begin(数组名)  
end(数组名)
```

▶ 运算

- ▶ 解引用
- ▶ 自增、自减
- ▶ 加或减整数、
- ▶ 指针相减
- ▶ 指针比较

```
#include<iterator> // 迭代器运算头文件  
...  
int ia[5] = {1, 2, 3, 4, 5};  
int *pb = begin(ia);  
int *pe = begin(ia);  
while(pb != pe && *pb >= 0)  
{  
    ++pb;  
}
```



▶ 字符数组和 C 风格字符串

- ▶ 以 `'\0'` 结束字符串
- ▶ 需要使用头文件: `#include <cstring>`

▶ C++ 的 string 类

- ▶ 需要使用头文件: `#include <string>`
- ▶ 丰富的字符串处理函数
- ▶ 便捷的运算符重载
- ▶ 单字符处理
 - ▶ 需要使用头文件: `#include <cctype>`
 - ▶ 基本循环
 - ▶ 范围 for



▶ 标准类型 vector

- ▶ 同种类型对象的集合
- ▶ 长度可变

▶ 定义和初始化

- ▶ 语法: `vector<元素类型> 变量名;`
- ▶ 初始化方法

方 法	说 明
<code>vector<T> v1</code>	v1 为空, 元素是 T 类型, 默认初始化
<code>vector<T> v2(v1)</code>	声明 v2 向量, 用 v1 初始化, 是 v1 的副本
<code>vector<T> v2 = v1</code>	等价于 <code>v2(v1)</code>
<code>vector<T> v3(n, val)</code>	v3 有 n 个 T 类型的重复元素, 每个元素的值都是 val
<code>vector<T> v4(n)</code>	v4 有 n 个重复默认值初始化的元素
<code>vector<T> v5{a,b,c,...}</code>	v5 元素个数为初始化式中元素个数
<code>vector<T> v5={a,b,c,...}</code>	等价于 <code>v5{a,b,c,...}</code>



▶ 强制类型转换

▶ C 风格

```
float x = 3.5;  
int roundX = (int)(x + 0.5);
```

▶ C++ 风格: castname<类型名>(表达式)

- ▶ static_cast
- ▶ dynamic_cast
- ▶ const_cast
- ▶ reinterpret_cast

```
int roundX = static_cast <int> (x+0.5);
```

▶ 强制指针类型转换

```
char *p;  
void *q = malloc(sizeof(char)*1024);  
p = q;    //错误! 无法从“void *”转换为“char *”,C++  
p = (char *)q;
```



OBJECT ORIENTED PROGRAMMING— OOP

知识回顾

对 C 的扩充

输入输出

cin/cout
格式控制

类型与表达式

基本类型
新类型
运算符与表达式

控制结构

基本结构
范围 for 语句

复杂类型

指针与内存
指针与常量
引用类型
C++ 类类型
类型转换

函数

内联函数
constexpr 函数
形参默认值
重载函数
函数模板
系统函数

程序结构

▶ 函数调用执行过程

▶ 内联函数

▶ 带默认形参的函数

▶ 函数重载

▶ 函数模板

▶ 系统函数

28

62



OBJECT
ORIENTED
PROGRAMMING—
OOP

知识回顾

对 C 的扩充

输入输出

cin/cout
格式控制

类型与表达式

基本类型
新类型
运算符与表达式

控制结构

基本结构
范围 for 语句

复杂类型

指针与内存
指针与常量
引用类型
C++ 类类型
类型转换

函数

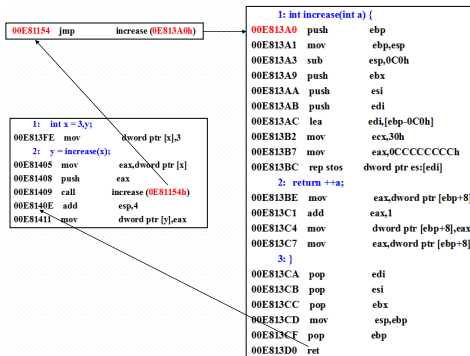
内联函数
constexpr 函数
形参默认值
重载函数
函数模板
系统函数

程序结构

- ▶ 函数调用
 - ▶ 参数和函数返回地址入栈
- ▶ 执行函数体
 - ▶ 寄存器进出栈, 通过栈访问参数
- ▶ 函数返回
 - ▶ 返回到调用函数的下一条语句执行

```
int increase(int a)
{
    return ++a;
}

int main()
{
    int x = 3, y;
    y = increase(x);
    return 0;
}
```





▶ 普通函数调用缺陷

▶ 时间开销

▶ 内联函数

▶ 在编译时将函数体代码插入到调用处

▶ 适用于代码短、频繁调用的场合

▶ 定义

```
inline 函数类型 函数名 (参数表)
{
    函数体;
}
```



► 本质是预处理后展开

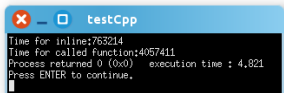
```
inline int increase(int a)
{
    return ++a;
}
int main()
{
    int x = 3, y;
    y = increase(x);
    return 0;
}
```



```
int main()
{
    int x = 3, y;
    int a = x;
    y = ++a;
    return 0;
}
```

► 效率测试

```
inline float getCos_inline(int &x)
{
    float r;
    x = rand();
    r = cos(2 * 3.1416 * x / (float)RAND_MAX);
    return r;
}
```



测试环境 (Code:Blocks GCC Release)

效率比 = 4057411ms/763214ms = 5.3

例 02-07: ex02-07.cpp



知识回顾

对 C 的扩充

输入输出

cin/cout

格式控制

类型与表达式

基本类型

新类型

运算符与表达式

控制结构

基本结构

范围 for 语句

复杂类型

指针与内存

指针与常量

引用类型

C++ 类类型

类型转换

函数

内联函数

constexpr 函数

形参默认值

重载函数

函数模板

系统函数

程序结构

► 注意事项

► 不能出现递归

► 代码不宜过长

► 不宜出现循环

► 不宜含有复杂控制语句如switch等

► 有些编译器会智能处理是否为内联函数



▶ 语法

▶ constexpr 函数（常量表达式函数）

▶ 基本要求

▶ 只有一句return可执行语句，可有别名、using等

▶ 必须有返回类型，返回类型不能是void

▶ 使用前必须定义（不只是声明）

▶ return中不能有非常量表达式



► 是编译时求值，不是运行是调用

```
constexpr int data() // 错误，函数体只能有一条 return 可执行语句
{
    const int i = 1;
    return i;
}

constexpr int data() // 正确
{
    return 1;
}

constexpr void f() // 错误，无法获得常量
{
}
```

► 是函数使用 (编译时)，不是函数调用 (运行时)

```
constexpr int f(); // 只有 constexpr 函数的声明，没有定义
int a = f(); // 正确，可以将编译时的计算转换为运行时的调用
const int b = f(); // 正确，编译器将 f() 转换为一个运行时的调用
constexpr int c = f(); // 错误，c 是 constexpr，要求使用 f()，在编译时计算
constexpr int f() // constexpr 函数的定义
{
    return 1;
}

constexpr int d = f(); // 正确，f() 已定义，可以使用 f()
```



▶ return 中不能包含运行时才能确定的函数

```
const int e()
{
    return 1;
}
constexpr int g()
{
    return e();    // 错误，调用了非 constexpr 函数
}
constexpr int e()
{
    return 1;
}
constexpr int g()
{
    return e();    // 正确，函数 e() 是常量表达式函数
}
```

▶ 用 constexpr 函数初始化 constexpr 变量

```
constexpr int new_sz()
{
    return 100;
}
constexpr int size = new_sz();
```



▶ 在函数定义或说明中为形参赋默认值

▶ 作用

- ▶ 若调用给出实参值，则形参采用实参值
- ▶ 若调用未给出实参值，则调用默认参数值

```
// 例 02-08: ex02-08.cpp
void SetNetCamera (char *UserName = "guest", char *Password = "321",
                  char *URL = "219.145.198.100", char *ServerName = "654",
                  float Zoom = 0.2, float Alpha = 10.0, float Beta = 15.0)
{
    cout << UserName << " ";
    cout << Password << " ";
    cout << URL << " ";
    cout << ServerName << " ";
    cout << Zoom << " ";
    cout << Alpha << " ";
    cout << Beta << endl;
}
```



► 基本要求

- 调用函数时，如省去某个实参，则该实参右边所有实参都要省略
- 默认形参必须自最右向左连续定义
- 若函数声明（原型）中给出默认形参值，则函数定义时不能重复指定

```
// 例 02-09: ex02-09.cpp
void SetNetCamera (char *UserName = "guest", char *Password = "321",
                  char *URL = "219.145.198.100", char *ServerName = "654",
                  float Zoom = 0.2, float Alpha = 10.0, float Beta = 15.0)
{
    cout << UserName << " " << Password << " "
         << URL << " " << ServerName << " "
         << Zoom << " " << Alpha << " " << Beta << endl;
}
int main()
{
    SetNetCamera();
    SetNetCamera("Xinji", "class1&2", "219.145.198.105", "654");
    SetNetCamera("Administrator", "nwsuaf", "219.145.198.108", "654",
                1.0, 15.0, 30.0);

    return 0;
}
```



知识回顾

对 C 的扩充

输入输出

cin/cout
格式控制

类型与表达式

基本类型
新类型
运算符与表达式

控制结构

基本结构
范围 for 语句

复杂类型

指针与内存
指针与常量
引用类型
C++ 类类型
类型转换

函数

内联函数
constexpr 函数

形参默认值
重载函数
函数模板
系统函数

程序结构

▶ 中间参数不能省略

```
// 例 02-10: ex02-10.cpp
void SetNetCamera (char *UserName = "guest", char *Password = "321",
                  char *URL = "219.145.198.100", char *ServerName = "654",
                  float Zoom = 0.2, float Alpha = 10.0, float Beta = 15.0)
{
    cout << UserName << " " << Password << " "
         << URL << " " << ServerName << " "
         << Zoom << " " << Alpha << " " << Beta << endl;
}
int main()
{
    SetNetCamera();
    SetNetCamera("Xinji", "class1&2", "219.145.198.105", "654");
    SetNetCamera("Administrator", "nwsuaf", , , 1.0, 15.0, 30.0);
    return 0;
}
```



► 不可重复指定参数默认值

```
// 例 02-11: ex02-11.cpp
void SetNetCamera (char *UserName = "Guest", char *Password = "321",
                  char *URL = "219.145.198.100", char *ServerName = "654",
                  float Zoom = 0.2, float Alpha = 10.0, float Beta = 15.0) ;

int main()
{
    SetNetCamera();
}

void SetNetCamera (char *UserName, char *Password, char *URL,
                  char *ServerName, float Zoom, float Alpha, float Beta)
{
    cout << UserName << " " << Password << " "
         << URL << " " << ServerName << " "
         << Zoom << " " << Alpha << " " << Beta << endl;
}
```




▶ 默认形参必须自最右向左连续定义

```
// 例 02-12: ex02-12.cpp
void SetNetCamera (char *UserName = "Guest", char *Password = "321",
                  char *URL = "219.145.198.100", char *ServerName = "654",
                  float Zoom, float Alpha, float Beta)
{
    cout << UserName << " " << Password << " "
         << URL << " " << ServerName << " "
         << Zoom << " " << Alpha << " " << Beta << endl;
}
```



▶ 默认形参必须自最右向左连续定义

```
// 例 02-13: ex02-13.cpp
void SetNetCamera (char *UserName = "Guest", char *Password = "321",
                  char *URL, char *ServerName,
                  float Zoom = 0.2, float Alpha = 10.0, float Beta = 15.0)
{
    cout << UserName << " " << Password << " "
         << URL << " " << ServerName << " "
         << Zoom << " " << Alpha << " " << Beta << endl;
}
```



▶ 必须为无默认值的参数提供实参

```
// 例 02-14: ex02-14.cpp
void SetNetCamera (char *UserName, char *Password,
                  char *URL = "219.145.198.100", char *ServerName = "654",
                  float Zoom = 0.2, float Alpha = 10.0, float Beta = 15.0)
{
    cout << UserName << " " << Password << " "
         << URL << " " << ServerName << " "
         << Zoom << " " << Alpha << " " << Beta << endl;
}
int main()
{
    SetNetCamera();
    SetNetCamera("Guest", "321");
    SetNetCamera("Xinong", "class1 & 2", "219.145.198.105", "654");
    SetNetCamera("Administrator", "nwsuaf");
    return 0;
}
```



▶ 形参的初始化可以是函数

```
// 例 02-15: ex02-15.cpp
#include <iostream>
using namespace std;
float RadianToAngle(float radian)
{
    return radian * 180.0 / 3.1416;
}
void SetNetCamera (char *UserName, char *Password,
                  char *URL = "219.145.198.100", char *ServerName = "654",
                  float Zoom = 0.2, float Alpha = RadianToAngle(0.174),
                  float Beta = RadianToAngle(0.262))
{
    cout << UserName << " " << Password << " "
         << URL << " " << ServerName << " "
         << Zoom << " " << Alpha << " " << Beta << endl;
}
int main()
{
    SetNetCamera("Xinji", "class1&2", "219.145.198.105", "654");
    return 0;
}
```



▶ 重载：同一符号或函数名对应多种操作

▶ 操作符重载

▶ 函数重载

▶ 函数重载

▶ 共性函数拥有相同函数名字

```
int sum_int(int *a, int size);  
float sum_float(float *a, int size);  
double sum_double(double *a, int size);
```

```
int sum(int *a, int size);  
float sum(float *a, int size);  
double sum(double *a, int size);
```



► C++ 函数重载实现机理

- 函数名
- 参数类型
- 参数个数

► 参数个数不同情况下的实现重载

```
float dis_2d(float x0, float y0, float x1, float y1);  
float dis_3d(float x0, float y0, float z0,  
             float x1, float y1, float z1);
```

```
float dis(float x0, float y0, float x1, float y1);  
float dis(float x0, float y0, float z0,  
          float x1, float y1, float z1);
```



► 注意事项

► 避免二义性

```
void my_fun(int a, int b);  
void my_fun(int &a, int &b);
```

► 避免将不适宜重载的函数重载

如果不同的函数名所提供的信息可使程序更容易理解，则不必使用重载

```
void rotate(float r);  
void translate(float x, float y);
```

```
void transform(float r);  
void transform(float x, float y);
```



► 用一个函数表示逻辑功能相同但参数类型不同的函数

```
int sum(int *a, int size);  
float sum(float *a, int size);  
double sum(double *a, int size);
```

► 定义

`template <class 类型名 1, class 类型名 1, ...>` 返回类型 函数名 (形参表)

```
{  
    函数体;  
}
```

```
// 例 02-16: ex02-16.cpp  
template <class _T>  
_T sum(_T *a, int size)  
{  
    _T result = 0;  
    for (int i = 0; i < size; i++)  
    {  
        result += a[i];  
    }  
    return result;  
}
```




知识回顾

对 C 的扩充

输入输出

cin/cout

格式控制

类型与表达式

基本类型

新类型

运算符与表达式

控制结构

基本结构

范围 for 语句

复杂类型

指针与内存

指针与常量

引用类型

C++ 类类型

类型转换

函数

内联函数

constexpr 函数

形参默认值

重载函数

函数模板

系统函数

程序结构

▶ 带有两个通用类型的函数模板

```
// 例 02-17: ex02-17.cpp
template <class T1, class T2>
void myfunc(T1 x, T2 y)
{
    cout << x << ' ' << y << endl;
}

int main()
{
    myfunc(10, "I hate C++");
    myfunc(98.6, 19L);
}
```



► 优先级别

- 如果同时定义重载函数，将优先使用重载函数，若不能找到精确匹配，再使用函数模板

► 应用

- 数据结构中的链表、堆栈等
- C++ 的标准模板库 (排序等)
- 通用类



► 应用（例：冒泡排序）

```
// 例 02-18: ex02-18.cpp
template <class _T>
void bubble(_T *items, int count)
{
    register int a, b;
    _T t;
    for(a = 1; a < count; a++)
        for(b = count - 1; b >= a; b--)
            if(items[b - 1] > items[b])
            {
                t = items[b - 1];
                items[b - 1] = items[b];
                items[b] = t;
            }
}
```



▶ cmath

▶ iostream

▶ 包含 ctype.h, string.h, memory.h, stdlib.h 等 isdigit(), strcpy(), memcpy(), atoi(), rand() 等

▶ ctime

▶ time_t, clock() 等



▶ extern

- ▶ 大型程序设计中所有模块共同使用的全局变量 (函数)
- ▶ 在一个模块中定义全局变量 (函数), 其它模块中用 **extern** 说明 “外来” 的全局变量 (函数)

```
// 例 02-19-01: ex02-19-01.cpp
#include <iostream>
using namespace std;
extern int G;
void p1dispG()
{
    G = 11;
    cout << "in p1 G=" << G << endl;
}
```

```
// 例 02-19-02: ex02-19-02.cpp
#include <iostream>
using namespace std;
extern int G;
extern int g;
void p2dispG(){
    G = 22;
    cout << "in p2 G=" << G << endl;
}
void p2dispg(){
    g = 33;
    cout << "in p2 g=" << g << endl;
}
```

```
// 例 02-19-00: ex02-19-00.cpp
#include <iostream>
using namespace std;
extern void p1dispG();
extern void p2dispG();
extern void p2dispg();
int G = 0, g = 0;
int main(){
    p1dispG();
    p2dispG();
    p2dispg();
    cout << "in p G=" << G << endl;
    cout << "in p g=" << g << endl;
    return 0;
}
```



▶ static

- ▶ **static** 可用来声明全局静态变量和局部静态变量。当声明全局静态变量时，全局静态变量只能供本模块使用，不能被其它模块再声明为 **extern** 变量

```
// 例 02-20-01: ex02-20-01.cpp
extern void p1dispG();
static int G=0;

int main() {
    p1dispG();
    cout<<"in p G="<<G<<endl;
    return 0;
}
```

```
// 例 02-20-02: ex02-20-02.cpp
extern int G;

void p1dispG(){
    G=11;
    cout<<"in p1 G="<<G<<endl;
}
```



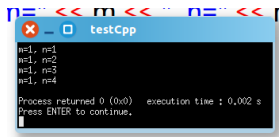
▶ static

▶ 当一个局部变量声明为 **static** 变量，它既具有局部变量的性质，又具有全局变量的性质

```
// 例 02-21: ex02-21.cpp
#include <iostream>
using namespace std;

void fun()
{
    static int n = 0;
    int m = 0;
    n++;
    m++;
    cout << "m=" << m << ", n=" << n << endl;
}

int main()
{
    for (int i = 0; i < 4; i++)
        fun();
    return 0;
}
```





▶ 多文件操作使用#include

▶ #include < 系统文件 >

▶ 到编译器指定的文件包含目录查找

▶ #include " | 自定义文件.h"



► 同一程序在不同的编译条件下得到不同的目标代码

```
// 例 02-22: ex02-22.cpp
#define USA      0
#define CHINA    1
#define ENGLAND  2
#define ACTIVE_COUNTRY USA

#if ACTIVE_COUNTRY == USA
char *currency = "dollar";
#elif ACTIVE_COUNTRY == ENGLAND
char *currency = "pound";
#else
char *currency = "yuan";
#endif
int main ( )
{
    float money;
    cin >> money;
    cout << money << currency << endl;
    return 0;
}
```



知识回顾

对 C 的扩充

输入输出

cin/cout
格式控制

类型与表达式

基本类型
新类型
运算符与表达式

控制结构

基本结构
范围 for 语句

复杂类型

指针与内存
指针与常量
引用类型
C++ 类类型
类型转换

函数

内联函数
constexpr 函数
形参默认值
重载函数
函数模板
系统函数

程序结构

► 便于程序移植或跨平台

```
// 例 02-23-01: ex02-23-01.cpp
#ifdef _WIN32
#include <windows.h>
...
#elif defined __APPLE__
...
#else
#include <unistd.h>
...
#endif
```

```
// 例 02-23-02: ex02-23-02.cpp
#ifdef __cplusplus
#include <iostream>
#else
#include <stdio.h>
#endif
```



▶ 避免重复包含头文件如 “MyHeader.h”

```
// 例 02-24: ex02-24.cpp
#ifndef __MYHEADER_H
#define __MYHEADER_H
...
#endif
```

▶ 便于调试程序

```
// 例 02-25: ex02-25.cpp
#define _DEBUG
#ifdef _DEBUG
...
#endif
```



▶ 不同程序员撰写的软件模块可能使用相同标志符

▶ 为避免冲突，将可能存在相同标志符的程序模块放入名字空间

▶ 定义：

```
namespace 名称  
{  
    成员变量或成员函数;  
}
```



▶ 声明方式（作用域分辨符） 名字空间名::成员变量或成员函数

```
// 例 02-25-02: ex02-25-02.cpp
//Xinong.h
#ifndef XINONG_H_INCLUDED
#define XINONG_H_INCLUDED

namespace Xinong
{
    int year = 2011;
    char name[] = "Xinong";
    void ShowName()
    {
        cout << name << " " <<
            year << endl;
    }
}

#endif // XINONG_H_INCLUDED
```

```
// 例 02-25-01: ex02-25-01.cpp
#include <iostream>
using namespace std;
#include "Xinong.h"

int main()
{
    Xinong::ShowName();
    return 0;
}
```



▶ 声明方式（作用域分辨符） 名字空间名::成员变量或成员函数

```
// 例 02-26-02: ex02-26-02.cpp
```

```
//Xilin.h
```

```
#ifndef XILIN_H_INCLUDED
```

```
#define XILIN_H_INCLUDED
```

```
namespace Xilin
```

```
{
```

```
    int year = 2011;
```

```
    char name[] = "Xilin";
```

```
    void ShowName()
```

```
    {
```

```
        cout << name << " " <<
```

```
        year << endl;
```

```
    }
```

```
}
```

```
#endif // XILIN_H_INCLUDED
```

```
// 例 02-26-01: ex02-26-01.cpp
```

```
#include <iostream>
```

```
using namespace std;
```

```
#include "Xilin.h"
```

```
int main()
```

```
{
```

```
    Xilin::ShowName();
```

```
    return 0;
```

```
}
```



OBJECT
ORIENTED
PROGRAMMING—
OOP

知识回顾

对 C 的扩充

输入输出

cin/cout

格式控制

类型与表达式

基本类型

新类型

运算符与表达式

控制结构

基本结构

范围 for 语句

复杂类型

指针与内存

指针与常量

引用类型

C++ 类类型

类型转换

函数

内联函数

constexpr 函数

形参默认值

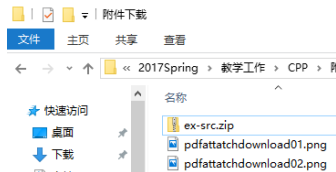
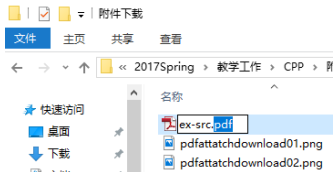
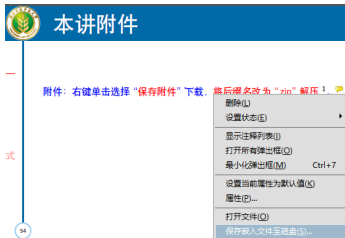
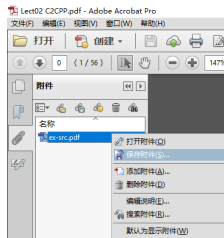
重载函数

函数模板

系统函数

程序结构

附件：右键单击该链接，选择“保存附件”下载，将后缀名改为“.zip”解压^{1 2}。



¹ 请退出全屏模式后点击该链接。

² 以 Adobe Acrobat Reader 为例。

本讲结束，谢谢！
欢迎多提宝贵意见和建议

西北农林科技大学
NORTHWEST A&F UNIVERSITY
中国·杨凌