



# 嵌入式系统工程师





---

# C++基础知识

---

- C++简介
- C++基本语法知识

## ➤ C++简介

### ➤ C++的起源

### ➤ C++的特点

## ➤ C++基本语法知识

## ➤ C++的起源

- C++是美国贝尔实验室在C语言的基础上，增加了面向对象的特征，于1980年开发出来的一种过程性与对象性结合的程序语言
- 最初把它称为“带类的C”，83年后才取名为C++

➤ C语言已经被公认为是非常好的一种中级语言，但它也有一些局限：

- C的类型检查机制相对较弱，这使得程序中的一些错误不能在编译阶段由编译器检查出来

如： `float x;`

`scanf ("%d", &x);`

- C语言本身几乎没有支持代码重用的语言结构。如：求一个数的绝对值：

对于int型数据： `int abs(int x);`

对于float型数据： `float fabs(float x);`

- C语言不适合开发大型程序，当程序达到一定规模时，程序员很难控制程序的复杂性

- C++正是为了解决上述问题而设计的
- C++继承了C原有的精髓（如高效率、灵活性），扩充增加了对开发大型软件颇为有效的面向对象的机制等等，成为一种既可用于表现过程模型，又可用于表现对象模型的优秀的设计语言之一

## ➤ C++的特点

### ➤ C++与C语言相比:

- 保持与C语言的兼容
- 可重用性、可扩充性、可维护性、可靠性都有很大提高
- 支持面向对象的机制

### ➤ C++与其他面向对象语言相比:

- 可读性更好, 可直接在程序中映射问题空间的结构
- 代码质量高, 比其他面向对象语言执行效率高得多



## ➤ C++与C不同之处:

- C源程序文件扩展名为.c，而C++为.cpp
- 在Windows下，当给定扩展名为.c时，启动C的编译器；而当给定扩展名为.cpp时则启动C++的编译器
- 在linux下，后缀名字只是给人看的，编译时需要制定编译器，`g++ main.cpp`

➤ C++简介

➤ C++基本语法知识

## ➤ C++基本语法知识

### ➤ 命名空间的简介

### ➤ 新的I/O流

### ➤ 作用域运算符“::”

### ➤ 结构、联合和枚举名可直接作为类型名

### ➤ const修饰符

### ➤ 内联函数

### ➤ 带有缺省参数的函数

### ➤ 函数重载

### ➤ 强制类型转换

### ➤ new和delete运算符

### ➤ 引用

## ➤ 为什么要使用命名空间:

- 解决命名冲突的问题
- 在C++中, 名称可以是: 变量、函数、结构、枚举、类。工程越大, 名称互相冲突性的可能性越大。
- 另外使用多个厂商的类库时, 也可能导致名称冲突



fun ()



fun ()

## ► 命名空间的定义:

```
► namespace myName {  
    //namespace member
```

.....

~~} ;~~ //注意没有分号

```
例: namespace CompanyA {  
    void fun() {//.....}  
    int num; }
```

```
CompanyA::fun(); CompanyA::num = 100;
```

## ➤ C++基本语法知识

➤ 命名空间的简介

➤ 新的I/O流

➤ 作用域运算符“::”

➤ 结构、联合和枚举名可直接作为类型名

➤ const修饰符

➤ 内联函数

➤ 带有缺省参数的函数

➤ 函数重载

➤ 强制类型转换

➤ new和delete运算符

➤ 引用

- I/O流：从某种I/O设备上输入或输出的字符序列
- 流对象的引入：cout对象和cin对象
- 输入输出方法
  - cin可以连续输入多个数据; cout可以连续输出多个数据
  - 默认情况下使用了系统缺省的格式，但可以进行格式控制
  - endl操纵符

## 例1.1 输入输出

## ➤说明:

- 用cout格式控制输出时, 可以控制输出浮点数的精确度和整型数据的格式 (浮点数不可以)
- 用cin>>name和cin.getline(name, sizeof(name))的区别:
  - 用cin直接获得字符串当遇到空格符、“tab”和回车符表示结束, 并且不进行字符个数检查有可能会越界;
  - 用cin.getline(..,..)函数时, 可以获得空格符, 同时进行字符个数检查, 最大获得sizeof(name)-1个字符



## ➤ C++基本语法知识

➤ 新的I/O流

➤ 作用域运算符“::”

➤ 结构、联合和枚举名可直接作为类型名

➤ const修饰符

➤ 内联函数

➤ 带有缺省参数的函数

➤ 函数重载

➤ 强制类型转换

➤ new和delete运算符

➤ 引用

- 作用域：变量在程序中的起作用范围
- 简单分为：全局作用域、局部作用域、语句作用域
- 作用域优先级：范围越小优先级越高
- 作用域运算符：“::”
  - 如果希望在局部变量的作用域内使用同名的全局变量，可以在该变量前加上“::”，“::”称为作用域运算符 例1.2作用域
- 思考：
  - 若将句1改为::aver=20, 结果如何?
  - 若将句1, 2都改为::aver, 结果为多少?

## ➤ C++基本语法知识

- 新的I/O流
- 作用域运算符“::”
- 结构、联合和枚举名可直接作为类型名
- const修饰符
- 内联函数
- 带有缺省参数的函数
- 函数重载
- 强制类型转换
- new和delete运算符
- 引用

- 在C++中，结构名、联合名（共用体名）、枚举名都是类型名。如：

```
enum BOOL {FALSE, TRUE};  
struct STRING {  
    char* prt;  
    int length;  
};
```

- 在C中：

```
enum    BOOL done;  
struct  STRING str;
```

- 在C++中：

```
BOOL done;  
STRING str;
```

## ➤ C++基本语法知识

- 新的I/O流
- 作用域运算符“::”
- 结构、联合和枚举名可直接作为类型名
- `const`修饰符
- 内联函数
- 带有缺省参数的函数
- 函数重载
- 强制类型转换
- `new`和`delete`运算符
- 引用

➤ 用 const 修饰符定义常量。一般的语法格式：

const 类型名 常量名=常量值（表达式）；

对于上例，用 const 定义为：

```
const int LIMIT = 100;
```

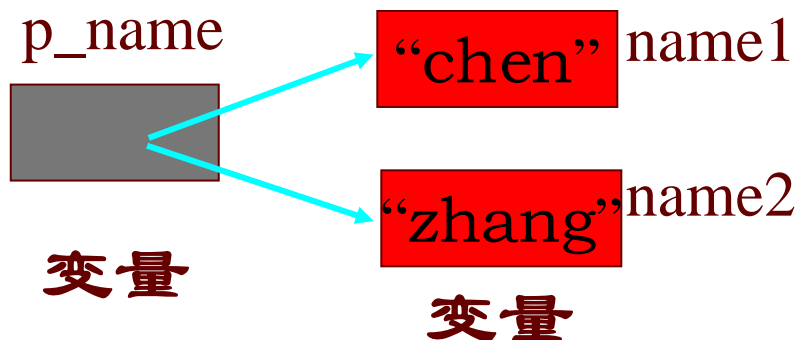
```
LIMIT = 100; //error
```

```
int* p = &LIMIT; //error
```

```
fun(&LIMIT); //error void fun(int  
*a);
```

## ➤ 指向常量的指针变量:

如: `const char * p_name = name1;`

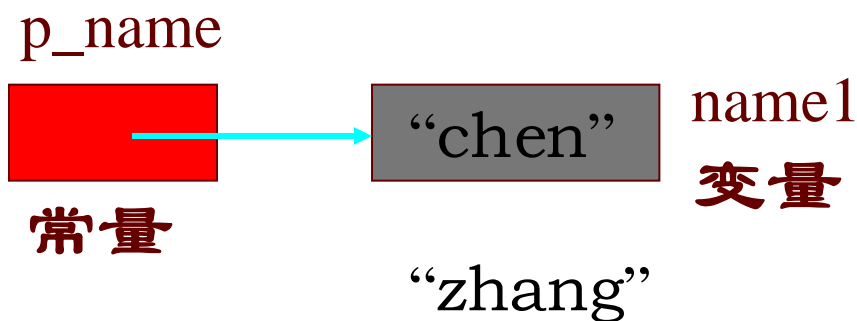


以下语句:

```
p_name[2]='a';           // 错误
p_name=name2;             // 正确
```

## ➤ 常(量)指针:

如: `char *const p_name = name1;`



以下语句:

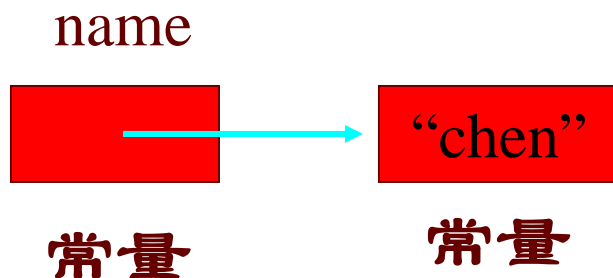
`p_name[2]='a';` // 正确

`p_name="zhang";` // 错误



## ➤ 指向常量的常(量)指针:

如: `const char * const name = "chen";`



以下语句:

`name[2]='a';`                      // 错误

`name="zhang";`                      // 错误

### 例1.3 const与指针

## ➤ C++基本语法知识

- 新的I/O流
- 作用域运算符“::”
- 结构、联合和枚举名可直接作为类型名
- const修饰符
- 内联函数
- 带有缺省参数的函数
- 函数重载
- 强制类型转换
- new和delete运算符
- 引用

- 在函数定义前冠以关键字“inline”，则该函数就被声明为内联函数
- 每当程序中出现对该函数的调用时，C++编译器使用函数体内的代码替代函数调用表达式
  - 优点：效率高、安全、可操作类的成员

- 使用内联函数替代宏定义，可以消除宏定义的不安全性。内联函数具有宏定义的所有优点而没有缺点

## 例1.4使用宏定义 & 内联函数

## ➤ C++基本语法知识

- 新的I/O流
- 作用域运算符“::”
- 结构、联合和枚举名可直接作为类型名
- const修饰符
- 内联函数
- 带有缺省参数的函数
- 函数重载
- 强制类型转换
- new和delete运算符
- 引用

- C++在声明函数原型时，可为一个或多个参数指定缺省参数值，以后调用此函数，若省略其中某一参数，C++自动地以缺省值作为相应参数的值。例如函数原型说明为：

```
int special(int x=5, float y=5.3);
```

当进行函数调用时，可以有以下几种形式：

- 1) `special(100, 79.8);`     `// x=100, y=79.8`
- 2) `special(25);`             `// x=25, y=5.3`
- 3) `special();`                `// x=5, y=5.3`

## ➤ 说明:

- 只能在声明时设置默认参数
- 缺省参数都必须是从右到左定义, 如:

```
int fun(int i, int j=5, int k);
```

//错误, k未使用

- 调用时实参对形参的初始化必须是从左向右的

例1.5 默认参数

练习1

## ➤ C++基本语法知识

- 新的I/O流
- 作用域运算符“::”
- 结构、联合和枚举名可直接作为类型名
- const修饰符
- 内联函数
- 带有缺省参数的函数
- 函数重载
- 强制类型转换
- new和delete运算符
- 引用



- 在传统的C语言中，函数名必须是唯一的，程序中不允许出现同名的函数
- C++中允许出现同名的函数，这种现象称为函数重载，是C++的多态特性之一
- 只要函数参数的类型不同，或者参数的个数不同，或者二者兼而有之，两个或两个以上的函数可以使用相同的函数名

- 当要求编写求整数、浮点数和双精度的平方数的函数时:

iSquare(int x)	}	Square(int x)
fSquare(float x)		Square(float x)
dSquare(double x)		Square(double x)

## 例1.6 函数重载

## ➤ 说明:

- 重载函数是根据参数个数或参数类型来确定调用哪一个重载版本的

错误示例:

```
int mul(int x, int y);
```

```
double mul(int x, int y);
```

- 一般而言，重载函数应执行相同的功能，例如abs()函数一般用来返回一个数的绝对值，如果重载abs()函数，让它返回一个数的平方根，则是不可取的

- 如果一组重载函数（可能带有默认参数）都允许相同实参个数的调用，将会引起调用的**二义性**

如有：

```
void func(int x);           //1
```

```
void func(int x, int y=4);  //2
```

```
void func(int x=3, int y=4, int z=5);    //3
```

调用时：

```
func(7);           //错误，调用哪一个？
```

```
func(20, 30);      //错误，调用后两个中的哪一个？
```

## ➤ C++基本语法知识

- 新的I/O流
- 作用域运算符“::”
- 结构、联合和枚举名可直接作为类型名
- const修饰符
- 内联函数
- 带有缺省参数的函数
- 函数重载
- 强制类型转换
- new和delete运算符
- 引用

## ➤ 强制类型转换

- C中使用形如 `(int) x` 的形式
- C++中还允许另一种形式，如 `int (x)`
- 以上两种方法C++都能接受，推荐使用后一种方法

## ➤ C++基本语法知识

- 新的I/O流
- 作用域运算符“::”
- 结构、联合和枚举名可直接作为类型名
- const修饰符
- 内联函数
- 带有缺省参数的函数
- 函数重载
- 强制类型转换
- new和delete运算符
- 引用

➤ C中动态内存操作:

malloc() 和 free() 两个函数

➤ C++中动态内存操作:

new和delete运算符

分配内存: `p = new 数据类型;`

释放内存: `delete p;`



## ➤ 说明: 例1.7

- new在为简单变量分配内存的同时进行初始化  
如: `int *p; p=new int (90);`
- 使用new可以为数组动态分配内存  
如: `int *p=new int [10];`
- 释放动态分配的数组内存区时  
如: `delete p; delete []p;`
- 使用new动态分配内存时, 如果没有足够的内存时, 将返回空指针(NULL)

## ➤ C++基本语法知识


- 新的I/O流
- 作用域运算符“::”
- 结构、联合和枚举名可直接作为类型名
- const修饰符
- 内联函数
- 带有缺省参数的函数
- 函数重载
- 强制类型转换
- new和delete运算符
- 引用

- 引用就是给变量一个别名，使指针运算更加方便
- 引用的定义

类型名 &别名=变量名或别名;

如: `int a = 5;` `int &b = a;`

例1.8引用

a  
 (b)

## ➤ 注意:


- 定义引用时必须初始化，之后不可重新赋新值

错误示例:

```
int a;  
int &b;    //错误  
b = a;
```

- 初始化可以为另一个引用名，如下:

```
int a;  
int &b = a;  
int &c = b;
```

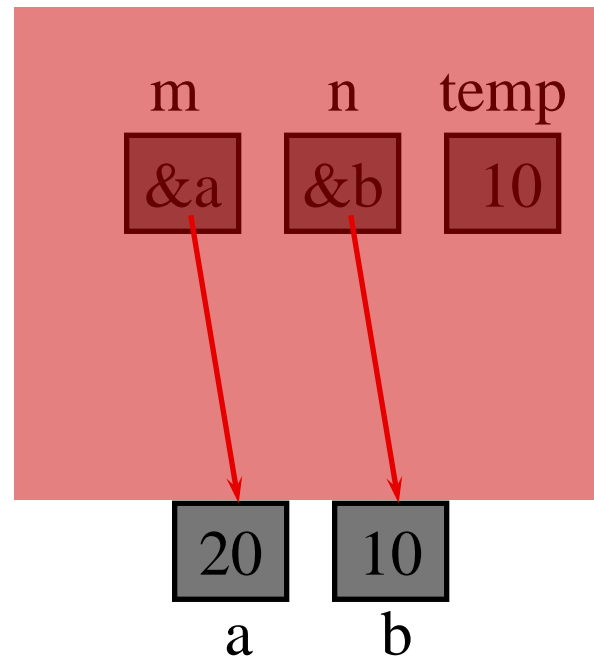
a  
(c)  (b)

## ➤ 引用作参数——引用的重要用途

如: `int func(int &x, int &y)`

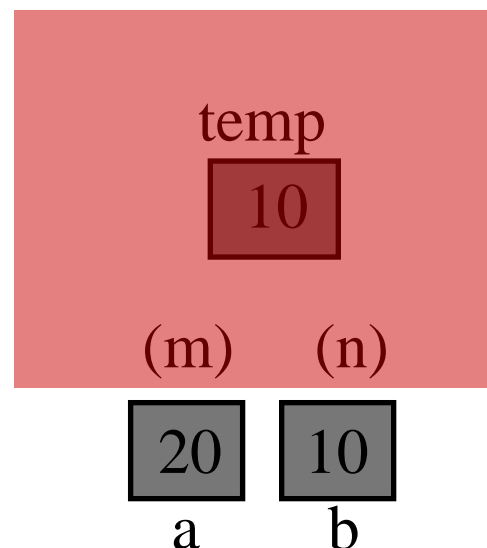
## ➤ 两个数互换的函数 例1.9 使用指针

```
void swap(int *m, int *n) {  
    int temp;  
    temp=*m;    *m=*n;    *n=temp;  
}  
  
int main() {  
    int a=10, b=20;  
    swap(&a, &b);  
    cout<<a<<b<<endl;  
}
```



## ➤ 两个数互换的函数

```
void swap(int &m, int &n) {  
    int temp;  
    temp=m;  m=n;  n=temp;  
}  
  
int main() {  
    int a=10, b=20;  
    swap(a, b);  
    cout<<a<<b<<endl;  
}
```

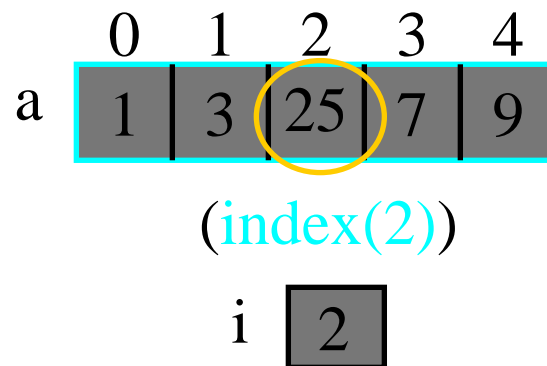


- 通过引用参数产生的效果同按地址传递是一样的
- 引用的语法更清楚简单
  - 函数调用时传递的实参不必加“&”符
  - 在被调函数中不必在参数前加“\*”符
- C++主张用引用传递取代地址传递的方式，因为引用语法容易且不易出错

## ➤ 函数返回值为引用

为了将该函数用在赋值运算符的左边，可将函数的返回值说明为引用 例1.10 使用引用

```
#include <iostream>
int a[]={1, 3, 5, 7, 9};
int &index(int); //声明返回引用的函数
int main() {
    index(2)=25;
    cout<<index(2);
}
int &index(int i) {
    return a[i];}
```



输出： 25



- 本章主要介绍了C++在非面向对象方面的一些重要特性:
  - 命名空间: 避免在同一作用域内重名现象的错误
  - 新的I/O流;
  - 作用域运算符“::”; 结构、联合和枚举名可直接作为类型名;
  - const修饰符; 内联函数;
  - 函数重载;
  - 带有缺省参数的函数;
  - 强制类型转换; new和delete运算符;
  - 引用
- 这些新特性使C++比C更简洁或更安全, 使得C++更好用



值得信赖的教育品牌

Tel: 400-705-9680 , Email: edu@sunplusapp.com , BBS: bbs.sunplusedu.com

