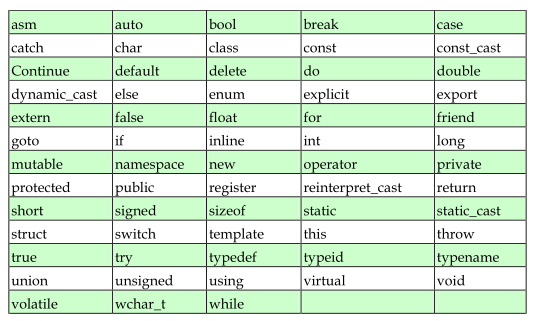
# C/C++语言基础讲义

## C/C++高级程序语言

1. C语言基础知识
2. 标识符 常量 变量
3. 标识符

* 定义：变量、符号常量、函数、数组、类型、文件、标签和其他各种用户定义的对象的名称。
* 命名规则：
* 第一个字符必须是字母或下划线
* 后续字符可以是字母、数字或下划线
* 标识符不能和关键字相同
* 标识符区分大小写
* 最好也不要和系统预定义标识符同名
* 标识符命名要做到“见名知义”
* 应该避免使用可能引起混淆的字母
* 系统关键字（C/C++系统保留字）



1. 常量

当一个数值，例如1出现在程序中时，它被称为文字常量（literal constant）称之为文字，是因为我们只能以它的值的形式指代它，称之为“常量”是因为它的值不能被改变。

* 直接常量（字面常量）：
* 整型常量：12、0、-3（值的后面加L或l）

可改写成十进制、八进制（024）、十六进制（0x14/0X14）

* 浮点型常量：4.6F、-1.23F（值的后面加F、f、L、l，只能用于十进制）
* 字符型常量： ‘a’、‘b’

一部分不可打印的字符、单引号、双引号以及反斜杠可以用如下的转义序列来表示（转义序列以反斜杠“\”开头）

* + **newline(换行符) \n**
  + **horizontal tab(水平制表键) \t**
  + **vertical tab(垂直制表键) \v**
  + **backspace(退格键) \b**
  + **carriage return (回车键) \r**
  + **formfeed (换页符) \f**
  + **alert (beel) (响铃符) \a**
  + **backslash (反斜杠键) \\**
  + **question mark (问号) \?**
  + **single quote (单引号) \'**
  + **double quote (双引号) \"**

字符文字前面可以加“L” 例如：L'a',这称为宽字符文字。

* 字符串型常量：字符串文字常量由零个或多个用双引号括起来的字符组成，不可打印字符可以由相应的转义序列来表示，而一个字符串文字可以扩展到多行，在一行的最后加上一个反斜杠，表明字符串文字在下一行继续。

例如：

* + **"" (空字符串)**
  + **"a"**
  + **"\nCC\toptions\tfile.[cC]\n"**
  + **"a multi-line \**

**string literal signals its \**

**continuation with a backslash"**

字符串文字前面可以加“L” 例如：L"a wide string literal"，表示宽字符串文字。

* 布尔（bool）型常量：true和false
* 符号常量：
* 用标示符代表一个常量，称之为符号常量。符号常量在使用之前必须先定义，其一般形式为： #define 标识符 常量
* 符号常量的值在其作用域内不能改变，也不能被再赋值。
* 符号常量的标识符用大写字母。

#define PRICE 30 **//PRICE为大写的标识符常量**

const int LENGTH = 20； **// LENGTH为大写的标识符常量**

1. 变量

* 变量命名：
* 变量为我们提供了一个有名字的内存存储区，可以通过程序对其进行读、写和处理。C++中的每个符号变量都与一个特定的数据类型相关联，这个类型决定了相关内存的大小、布局、能够存储在该内存区的值的范围以及可以应用其上的操作集。我们也可以把变量说成对象（object）。变量标识符用小写字母表示，以区别符号常量。

如：var = PRICE **// var 为变量，小写；PRICE为符号常量，大写**。

* 变量名，即变量的标识符（identifier），可以由字母、数字以及下划线字符组成，它必须以字母或下划线开头，并且区分大写字母和小写字母，语言本身对变量名的长度没有限制但是为用户着想，它不应该过长。
* 基本数据类型：
* 七种基本的 C++ 数据类型：

|  |  |
| --- | --- |
| **类型** | **关键字** |
| 布尔型 | **bool** |
| 字符型 | **char** |
| 整型 | **int** |
| 浮点型 | **float** |
| 双浮点型 | **double** |
| 无类型 | **void** |
| 宽字符型 | **wchar\_t** |

* 一些基本类型可以使用一个或多个类型修饰符进行修饰：
* signed
* unsigned
* short
* long
* 各种变量类型在内存中存储值时需要占用的内存，以及该类型的变量所能存储的最大值和最小值。变量的大小会根据编译器和所使用的电脑而有所不同。

|  |  |  |
| --- | --- | --- |
| 类型 | 字节/位 | 范围 |
| char | 1 个字节 | -128 到 127 或者 0 到 255 |
| unsigned char | 1 个字节 | 0 到 255 |
| signed char | 1 个字节 | -128 到 127 |
| int | 4 个字节 | -2147483648 到 2147483647 |
| unsigned int | 4 个字节 | 0 到 4294967295 |
| signed int | 4 个字节 | -2147483648 到 2147483647 |
| short int | 2 个字节 | -32768 到 32767 |
| unsigned short int | 2 个字节 | 0 到 65,535 |
| signed short int | 2 个字节 | -32768 到 32767 |
| long int | 8 个字节 | -9,223,372,036,854,775,808~ 9,223,372,036,854,775,807 |
| signed long int | 8 个字节 | -9,223,372,036,854,775,808~ 9,223,372,036,854,775,807 |
| unsigned long int | 8 个字节 | 0 到 18,446,744,073,709,551,615 |
| float | 4 个字节 | 精度型占4个字节（32位）内存空间，+/- 3.4e +/- 38 (~7 个数字) |
| double | 8 个字节 | 双精度型占8 个字节（64位）内存空间，+/- 1.7e +/- 308 (~15 个数字) |
| long double | 16个字节 | 长双精度型 16 个字节（128位）内存空间，可提供18-19位有效数字。 |
| wchar\_t | 2或4个字节 | 1 个宽字符 |

* 变量定义：
* 一个简单的定义指定了变量的类型和标识符，它并不提供初始值，必须对其进行初始化，如果一个变量是在全局域（global scope）内定义的，那么系统会保证给它提供初始值0。
* 使用赋值操作符的显式语法定义形式，C语言和C++相同。

**int var1= 1024;**

**string project = "Fantasia 2000";**

* 在隐式定义形式中，初始值被放在括号中，只能在C++中这样定义。

**int var1( 1024 );**

**string project( "Fantasia 2001" ))；**

1. 算术表达式
   1. 算术操作符

|  |  |  |
| --- | --- | --- |
| 操作符 | 描述 | 实例 |
| + | 把两个操作数相加 | A + B 将得到 30 |
| - | 从第一个操作数中减去第二个操作数 | A - B 将得到 -10 |
| \* | 把两个操作数相乘 | A \* B 将得到 200 |
| / | 分子除以分母 | B / A 将得到 2 |
| % | 取模运算符，整除后的余数 | B % A 将得到 0 |
| ++ | 自增运算符，整数值增加 1 | A++ 将得到 11 |
| -- | 自减运算符，整数值减少 1 | A-- 将得到 9 |

* 1. 位操作符

|  |  |  |
| --- | --- | --- |
| **操作符** | **描述描述** | **实例** |
| **&** | 如果同时存在于两个操作数中，二进制 AND 运算符复制一位到结果中。 | (A & B) 将得到 12，即为 0000 1100 |
| **|** | 如果存在于任一操作数中，二进制 OR 运算符复制一位到结果中。 | (A | B) 将得到 61，即为 0011 1101 |
| **^** | 如果存在于其中一个操作数中但不同时存在于两个操作数中，二进制异或运算符复制一位到结果中。 | (A ^ B) 将得到 49，即为 0011 0001 |
| **~** | 二进制补码运算符是一元运算符，具有"翻转"位效果，即0变成1，1变成0。 | (~A ) 将得到 -61，即为 1100 0011，一个有符号二进制数的补码形式。 |
| **<<** | 二进制左移运算符。左操作数的值向左移动右操作数指定的位数。 | A << 2 将得到 240，即为 1111 0000 |
| **>>** | 二进制右移运算符。左操作数的值向右移动右操作数指定的位数。 | A >> 2 将得到 15，即为 0000 1111 |

1. 强制类型转换表达式
   1. 隐式类型转换

C++定义了一组内置类型对象之间的标准转换，在必要时它们被编译器隐式地应用到对象上。 隐式类型转换发生在下列这些典型的情况下：

* 在混合类型的算术表达式中

**int ival = 3;**

**double dval = 3.14159;**

**// ival 被提升为 double 类型: 3.0**

**ival + dval;**

* 用一种类型的表达式赋值给另一种类型的对象

**// 0 被转换成 int\*类型的空指针值**

**int \*pi = 0;**

**// dval 被截取为 int 值 3**

**ival = dval;**

* 把一个表达式传递给一个函数调用，表达式的类型与形式参数的类型不相同。在这种情况下，目标转换类型是形式参数的类型。

**extern double sqrt( double );**

**// 2 被提升为 double 类型: 2.0**

**cout << "The square root of 2 is "**

**<< sqrt( 2 ) << endl;**

* 从一个函数返回一个表达式，表达式的类型与返回类型不相同。在这种情况下，目标转换类型是函数的返回类型

**double difference( int ival1, int ival2 )**

**{**

**// 返回值被提升为 double 类型**

**return ival1 - ival2;**

**}**

* 1. 算数转换

算术转换保证了二元操作符（如加法或乘法）的两个操作数被提升为共同的类型，然后再用它表示结果的类型，两个通用的指导原则如下：

* 为防止精度损失，如果必要的话，类型总是被提升为较宽的类型
* 所有含有小于整型的有序类型的算术表达式，在计算之前，其类型都会被转换成整型
* 如果一个操作数的类型是long double，那么另一个操作数无论是什么类型，都将被转换成 long double。

**3.14159L + 'a';**

* 如果两个操作数都不是long double 型，那么当其中一个操作数的类型是 double 型 则另一个就将被转换成 double 型。

**int ival;**

**float fval;**

**double dval;**

**// 在计算加法前 fval 和 ival 都被转换成 double**

**dval + fval + ival;**

* 如果两个操作数都不是 double 型，而其中一个操作数是 float 型，则另一个被转换成 float 型。

**char cval;**

**int ival;**

**float fval;**

**// 在计算加法前 ival 和 cval 都被转换成 double**

**cval + fval + ival;**

* 在进行整值提升时，类型 char、signed char、unsigned char 和 short int 都被提升为类型int。
* 如果机器上的 m 型足够表示所有 unsinned shoft 型的值，则 unsigned short int 也被转换成 int 否则，它会被提升为unsigned int。
* wchar\_t 和枚举类型被提升为能够表示其底层类型 underlying type 所有值的最小整数类型。
* 如果两个操作数的类型都不是 unsigned long 而其中一个操作数是 long 型，则另一个也被转换成 long 型。
* long 类型的一般转换有一个例外，如果一个操作数是 long 型，而另一个是 unsigned int型，那么只有机器上的 long 型足够长以便能够存放 unsigned int 的所有值时，unsigned int 才会被转换为 long 型 否则两个操作数都被提升为 unsigned long 型。
  1. 显式转换（强制类型转换）
* 显式转换也被称为强制类型转换（cast）包括下列命名的强制类型转换操作符static\_cast、 dynamic\_cast、 const\_cast和reinterpret\_cast。

形式如: **cast-name< type >( expression );**

* 如我们用隐式转换：

**double dval;**

**int ival;**

**ival += dval;**

通过显式地将dval强制转换成int型，消除了把ival从int型到double型的不必要提升：

**ival += static\_cast< int >( dval );**

* 作用：避免出现多种转换可能的歧义情况
  1. 旧式强制类型转换
* 旧式强制转换符号有下列两种形式

**// C++强制转换符号**

**type (expr);**

**// C 语言强制转换符号**

**(type) expr;**

* 旧式强制转换符号的例子

**const char \*pc = (const char\*) pcom;**

**int ival = (int) 3.14159;**

**extern char \*rewrite\_str( char\* );**

**char \*pc2 = rewrite\_str( (char\*) pc );**

**int addr\_value = int( &ival );**

1. 赋值表达式

|  |  |  |
| --- | --- | --- |
| 操作符 | 操作描述 | 示例 |
| = | 简单的赋值运算符，把右边操作数的值赋给左边操作数 | C = A + B 将把 A + B 的值赋给 C |
| += | 加且赋值运算符，把右边操作数加上左边操作数的结果赋值给左边操作数 | C += A 相当于 C = C + A |
| -= | 减且赋值运算符，把左边操作数减去右边操作数的结果赋值给左边操作数 | C -= A 相当于 C = C - A |
| \*= | 乘且赋值运算符，把右边操作数乘以左边操作数的结果赋值给左边操作数 | C \*= A 相当于 C = C \* A |
| /= | 除且赋值运算符，把左边操作数除以右边操作数的结果赋值给左边操作数 | C /= A 相当于 C = C / A |
| %= | 求模且赋值运算符，求两个操作数的模赋值给左边操作数 | C %= A 相当于 C = C % A |
| <<= | 左移且赋值运算符 | C <<= 2 等同于 C = C << 2 |
| >>= | 右移且赋值运算符 | C >>= 2 等同于 C = C >> 2 |
| &= | 按位与且赋值运算符 | C &= 2 等同于 C = C & 2 |
| ^= | 按位异或且赋值运算符 | C ^= 2 等同于 C = C ^ 2 |
| |= | 按位或且赋值运算符 | C |= 2 等同于 C = C | 2 |

* 格式：
* =

操作数1 = 操作数2

* 操作数2必须是一个值或者是一个初始化的对象。
* 操作数1和操作数2应该是相同类型的对象
* 操作数1和操作数2同为相同类型对象时，可 操作数1=操作数2=值
* 复合赋值操作符：

**+= -= \*= /= %=**

**<<= >>= &= ^= |=**

1. ++和--

1、功能

递增 ++ 和递减 -- 操作符为对象加 1 或减 1 操作提供了方便简短的表示

2、表达式

* 递增/减 操作符后置
* 先使用当前值，然后再递增或递减1
* var++，var--
* 递增/减 操作符前置
* 先将当前值递增或递减1，然后再使用
* ++var，--var

1. 逗号表达式

逗号表达式是一系列由逗号分开的表达式，如：

**int var1，var2；**

**int var3=0，var4=3；**

**for（…,…;…;…）**

**int var5=(var3++,var4+var3)**

1. 输入和输出语句

* 在Ｃ语言中，所谓输入输出是以计算机为主体而言的。是向标准输出设备显示器输出数据的语句。所有的数据输入／输出都是由库函数完成的。 因此都是函数语句。在使用Ｃ语言库函数时，要用预编译命令

**#include ‘…’**

将有关“头文件”包括到源文件中。使用标准输入输出库函数时要用到 “stdio.h”文件，因此源文件开头应有以下预编译命令：

**#include< stdio.h >**

或

**#include ‘tdio.h’**

stdio 是 standard input & outupt 的意思。

* 在C++中，输入/输出功能由输入/输出流<iostream> 库提供。终端输入也被称为标准输入（standard input）与预定义的 iostream 对象 cin 发音为see-in 绑定在一起，直接向终端输出，也被称为标准输出（standard output）；与预定义的iostream 对象 cout 发音为 see-out 绑定在一起，第三个预定义 iostream 对象 cerr 发音为see-err 称为标准错误（standard error）也与终端绑定 cerr 通常用来产生给程序用户的警告或错误信息。任何要想使用 iostream 库的程序必须包含相关的系统头文件的预编译命令：

**#include <iostream>。**

1. 输出语句
2. C语言的输出：

* printf 函数称为格式输出函数，其关键字最末一个字母 f 即为“格式”(format)之意。其功能是按用户指定的格式，把指定的数据显示到显示器屏幕上。
* printf 函数调用的一般形式

printf 函数是一个标准库函数，它的函数原型在头文件“stdio.h”中。

**printf(“格式控制字符串”，输出表列)**

* 其中格式控制字符串用于指定输出格式。

格式控制串可由格式字符串和非格式字符串两种组成。格式字符串是以%开头的字符串，在%后面跟有各种格式字符，以说明输出数据的类型、形式、长度、小数位数等。如：

“%d”表示按十进制整型输出；

“%ld”表示按十进制长整型输出；

“%c”表示按字符型输出等。

非格式字符串在输出时原样，在显示中起提示作用。

输出表列中给出了各个输出项，要求格式字符串和各输出项在数量和类型上应该一一对应。

1. C++的输出

* 输出操作符“<<”用来将一个值导向到标准输出 cout 或标准错误 cerr 上。

**cout << "Please enter input and output file names: ";**

**cout << "The sum of v1 + v2 = " << v1 + v2 << endl;**

**cerr << "Sorry! We were unable to open the file!\n";**

1. 输入语句
2. C语言的输入：

* scanf 函数称为格式输入函数，即按用户指定的格式从键盘上把数据输入到指定的变量之中。
  + scanf 函数的一般形式

scanf 函数是一个标准库函数，它的函数原型在头文件“stdio.h”中

**scanf(“格式控制字符串”，地址表列);**

* + 其中格式控制字符串的作用与 printf 函数相同，但不能显示非格式字符串，也就是不能显示提示字符串。地址表列中给出各变量的地址。地址是由地址运算符“&”后跟变量名组成的。

例如:

&a, &b分别表示变量 a 和变量 b 的地址。

这个地址就是编译系统在内存中给 a,b 变量分配的地址。在Ｃ语言中，使用了地址这个概念，这是与其它语言不同的。应该把变量的值和变量的地址这两个不同的概念区别开来。变量的地址是 C 编译系统分配的，用户不必关心具体的地址是多少。

* + 变量的地址和变量值的关系如下：

在赋值表达式中给变量赋值，如：

**a=567**

则，a 为变量名，567 是变量的值，&a 是变量 a 的地址。

但在赋值号左边是变量名，不能写地址，而 scanf 函数在本质上也是给变量赋值，但要求写变量的地址，如&a。 这两者在形式上是不同的。&是一个取地址运算符，&a 是一个表达式，其功能是求变量的地址。

1. C++的输入：

* 输入操作符“>>”用来从标准输入读入一个值。

**string ifile, ofile;**

**// ...**

**cout << "Please enter input and output file names: ";**

**cin >> ifile >> ofile;**

1. 分支结构
2. 关系运算与逻辑运算



1. 关系运算符都是双目运算符，其结合性均为左结合。关系运算符的优先级低于算术运算符，高于赋值运算符。

关系表达式的一般形式： **表达式1 关系运算符** **表达式2**

**例如：**

**a+b > c-d**

**x >= 3/2**

**‘a’+1 < c**

**-i-5\*j == k+1**

**a > (b > c)**

**a != (c == d)**

1. 与运算符&&和或运算符||均为双目运算符。具有左结合性。非运算符!为单目运算符，具有右结合性。

逻辑表达式的一般形式： **表达式1 逻辑运算符 表达式22**

**例如：**

**a>b && c>d**

**!b = = c || d<a**

**a+b > c && x+y < b**

1. 条件表达式
2. 如果在条件语句中，只执行单个的赋值语句时， 常可使用条件表达式来实现。不但使程序简洁，也提高了运行效率。
3. 条件运算符组成条件表达式的一般形式为：

**表达式1 ? 表达式2： 表达式3**

1. 其求值规则为：如果表达式 1 的值为真，则以表达式 2 的值作为条件表达式的值，否则以表达式 3的值作为整个条件表达式的值。
2. 使用条件表达式时，还应注意以下几点：

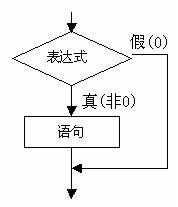
* 条件运算符的运算优先级低于关系运算符和算术运算符，但高于赋值符。因此**max=(a>b)?a:b**可以去掉括号而写为**max=a>b?a:b**
* 条件运算符?和：是一对运算符，不能分开单独使用。
* 条件运算符的结合方向是自右至左。

例如：

**a>b?a:c>d?c:d**应理解为**a>b?a:(c>d?c:d)**

这也就是条件表达式嵌套的情形，即其中的表达式3又是一个条件表达式。

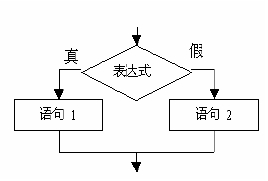
1. If语句
2. 三种基本形式

* 第一种形式为基本形式：
* 格式：

**if(表达式) 语句**

* 流程图
* 示例：

**if (max<b) max=b;**

* 第二种形式为: if-else
* 格式：

**if(表达式)**

**语句 1；**

**else**

**语句 2；**

* 流程图

示例：

**if(a>b)**

**printf("max=%d\n",a);**

**else**

**printf("max=%d\n",b);**

* 第三种形式:if-else-if
* 格式：

**if(表达式 1)**

**语句 1；**

**else if(表达式 2)**

**语句 2；**

**else if(表达式 3)**

**语句 3；**

**…**

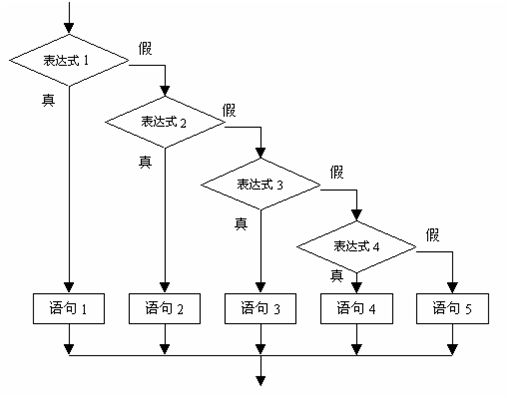
**else if(表达式 m)**

**语句 m；**

**else**

**语句 n；**

* 流程图



* 示例：

**if(c<32)**

**printf("This is a control character\n");**

**else if(c>='0'&&c<='9')**

**printf("This is a digit\n");**

**else if(c>='A'&&c<='Z')**

**printf("This is a capital letter\n");**

**else if(c>='a'&&c<='z')**

**printf("This is a small letter\n");**

**else**

**printf("This is an other character\n");**

1. 使用 if 语句中还应注意以下问题:

* 在三种形式的 if 语句中，在 if 关键字之后均为表达式。 该表达式通常是逻辑表达式或关系表达式， 但也可以是其它表达式，如赋值表达式等，甚至也可以是一个变量。

**if(a=5) 语句；**

**if(b) 语句；**

**if(a=5)…；**

* 在 if 语句中，条件判断表达式必须用括号括起来，在语句之后必须加分号。
* 在 if 语句的三种形式中，所有的语句应为单个语句，如果要想在满足条件时执行一组(多个)语句，则必须把这一组语句用{}括起来组成一个复合语句。但要注意的是在}之后不能再加分号。

**if(a>b)**

**{a++;**

**b++;}**

**else**

**{a=0;**

**b=10;}**

1. if语句的嵌套

* 当 if 语句中的执行语句又是 if 语句时，则构成了 if 语句嵌套的情形。为了二义性，规定，else 总是与它前面最近的 if 配对
* 格式：

**if(表达式 1)**

**if(表达式 2)**

**语句 1；**

**else**

**语句 2；**

* 示例

**if(a!=b)**

**if(a>b)**

**printf("A>B\n");**

**else**

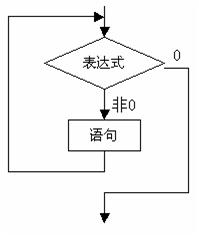
**printf("A<B\n");**

**else**

**printf("A=B\n");**

1. 循环结构
2. while语句
3. while 语句的一般形式为：

**while(表达式) 语句**

1. while 语句的语义是：计算表达式的值，当值为真(非 0)时， 执行循环体语句。
2. 流程图：
3. 示例：

* **while(getchar()!='\n') n++;**
* **while(i<=100)**

**{**

**sum=sum+I;**

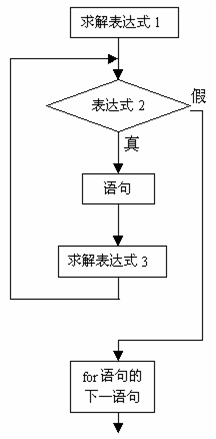
**i++;**

**}**

1. for语句
2. for 语句使用最为灵活，它完全可以取代 while 语句。它的一般形式为:

**for(表达式1；表达式2；表达式3) 语句**

1. 它的执行过程如下：
2. 先求解**表达式1**。
3. 求解**表达式 2**，若其值为真（非 0），则执行 for 语句中指定的内嵌**语句**，然后执行下面第 3rd步；若其值为假（0），则结束循环，转到第 5th步。
4. 求解**表达式3**。
5. 转回上面第 2nd步继续执行。
6. 循环结束，执行 for 循环结构外的语句。
7. 流程图



1. for循环语句解释

**for(循环变量赋初值；循环条件；循环变量增量) 语句**

**循环变量赋初值**总是一个赋值语句, 它用来给循环控制变量赋初值; **循环条件**是一个关系表达式,它决定什么时候退出循环；**循环变量增量**，定义循环控制变量每循环一次后按什么方式变化。这三个部分之间用“；”分开。

例如:

**for(i=1; i<=100; i++) sum=sum+i;**

对于 for 循环中语句的一般形式，就是如下的 while 循环形式：

**循环变量赋初值；**

**while（循环条件）**

**{**

**语句**

**循环变量增量；**

**}**

1. 注意:

* for 循环中的“表达式1（循环变量赋初值）”、“表达式2(循环条件)”和“表达式3(循环变量增量)”都是选择项, 即可以缺省,但“；”不能缺省。
* 省略了“表达式 1（循环变量赋初值）”, 表示不对循环控制变量赋初值。
* 省略了“表达式 2(循环条件)”, 则不做其它处理时便成为死循环。
* 省略了“表达式 3(循环变量增量)”, 则不对循环控制变量进行操作,这时可在语句体中加入修改循环控制变量的语句。
* 省略了“表达式 1（循环变量赋初值）”和“表达式 3(循环变量增量)”。
* 3个表达式都可以省略。
* 表达式 1 可以是设置循环变量的初值的赋值表达式，也可以是其他表达式。
* 表达式 1 和表达式 3 可以是一个简单表达式也可以是逗号表达式。
* 表达式 2 一般是关系表达式或逻辑表达式，但也可是数值表达式或字符表达式，只要其值非零，就执行循环体。

1. 循环的嵌套

* 当循环结构语句中的执行语句又是循环结构语句时，则构成了循环结构语句嵌套的情形。
* 格式：

**while(表达式1)**

**while(表达式2)**

**语句(块)；**

**for(表达式1; 表达式2;表达式3)**

**for(表达式1;表达式2;表达式3)**

**语句(块)；**

1. break与continue语句
2. break语句

当 break 语句用于 while(do-while)、for循环语句中时,可使程序终止循环而执行循环

后面的语句, 通常 break 语句总是与 if 语句联在一起。即满足条件时便跳出循环。

例如：

* while

格式：

**while(表达式1)**

**{**

**…**

**if(表达式2)**

**break;**

**…**

**}**

while循环应用举例

**while(1)**

**{**

**c=’\0’;**

**c=getch();**

**if(c==27)**

**break;**

**i++**

**printf(“%d”,i);**

**}**

* for

格式：

**for(表达式1; 表达式2; 表达式3)**

**{**

**…**

**if(表达式4)**

**break;**

**…**

**}**

for循环应用举例

**for(i=0;i<100;i++)**

**for(j=0;j<100-i-1;j++)**

**{**

**if(j%5==0)**

**break;**

**printf(“%d\*%d=”,i\*j);**

**}**

1. continue语句

continue 语句的作用是跳过循环本中剩余的语句而强行执行下一次循环。continue 语句只用在 for、while（do-while）等循环体中,常与 if 条件语句一起使用,用来加速循环。

* while

格式：

**while(表达式1)**

**{**

**…**

**if(表达式2)**

**continue;**

**…**

**}**

while循环应用举例

**while(1)**

**{**

**c=’\0’;**

**c=getch();**

**if(c==27)**

**continue;**

**i++**

**printf(“%d”,i);**

**}**

* for

格式：

**for(表达式1; 表达式2; 表达式3)**

**{**

**…**

**if(表达式4)**

**continue;**

**…**

**}**

for循环应用举例

**for(i=0;i<100;i++)**

**for(j=0;j<100-i-1;j++)**

**{**

**if(j%5==0)**

**continue;**

**printf(“%d\*%d=”,i\*j);**

**}**

1. 库函数
2. 函数的说明
3. 说明

C/C++源程序是由函数组成的。虽然在前面各章的程序中大都只有一个主函数**main()**，但实用程序往往由多个函数组成。函数是C/C++源程序的基本模块，通过对函数模块的调用实现特定的功能。C/C++语言中的函数相当于其它高级语言的子程序。C/C++语言不仅提供了极为丰富的库函数，还允许用户建立自己定义的函数。用户可把自己的算法编成一个个相对独立的函数模块，然后用调用的方法来使用函数。可以说C/C++程序的全部工作都是由各式各样的函数完成的，所以也把C/C++语言称为函数式语言。由于采用了函数模块式的结构，C/C++语言易于实现结构化程序设计。使程序的层次结构清晰,便于程序的编写、阅读、调试。

1. 在C/C++语言中可从不同的角度对函数分类。
   * 从函数定义的角度看，函数可分为库函数和用户定义函数两种。
   * 库函数：由C/C++系统提供，用户无须定义，也不必在程序中作类型说明，只需在程序前包含有该函数原型的头文件即可在程序中直接调用。printf、scanf、getchar、putchar、gets、puts、strcat 等函数均属此类。
   * 用户定义函数：由用户按需要写的函数。对于用户自定义函数，不仅要在程序中定义函数本身，而且在主调函数模块中还必须对该被调函数进行类型说明，然后才能使用。
   * C/C++语言的函数兼有其它语言中的函数和过程两种功能，从这个角度看，又可把函数分为有返回值函数和无返回值函数两种。
   * 有返回值函数：此类函数被调用执行完后将向调用者返回一个执行结果，称为函数返回值。如数学函数即属于此类函数。由用户定义的这种要返回函数值的函数，必须在函数定义和函数说明中明确返回值的类型。
   * 无返回值函数：此类函数用于完成某项特定的处理任务，执行完成后不向调用者返回函数值。这类函数类似于其它语言的过程。由于函数无须返回值，用户在定义此类函数时可指定它的返回为“空类型”， 空类型的说明符为“void”。
   * 从主调函数和被调函数之间数据传送的角度看又可分为无参函数和有参函数两种。
   * 无参函数：函数定义、函数说明及函数调用中均不带参数。主调函数和被调函数之间不进行参数传送。此类函数通常用来完成一组指定的功能，可以返回或不返回函数值。
   * 有参函数：也称为带参函数。在函数定义及函数说明时都有参数，称为形式参数(简称为形参)。在函数调用时也必须给出参数，称为实际参数(简称为实参)。进行函数调用时，主调函数将把实参的值传送给形参，供被调函数使用。
   * C/C++语言提供了极为丰富的库函数，这些库函数又可从功能角度作以下分类。
   * 字符类型分类函数：用于对字符按 ASCII 码分类：字母，数字，控制字符，分隔符，大小写字母等。
   * 转换函数：用于字符或字符串的转换；在字符量和各类数字量(整型，实型等)之间进行转换；在大、小写之间进行转换。
   * 目录路径函数：用于文件目录和路径操作。
   * 诊断函数：用于内部错误检测。
   * 图形函数：用于屏幕管理和各种图形功能。
   * 输入输出函数：用于完成输入输出功能。
   * 接口函数：用于与 DOS，BIOS 和硬件的接口。
   * 字符串函数：用于字符串操作和处理。
   * 内存管理函数：用于内存管理。
   * 数学函数：用于数学函数计算。
   * 日期和时间函数：用于日期，时间转换操作。
   * 进程控制函数：用于进程管理和控制。
   * 其它函数：用于其它各种功能。
   * 还应该指出的是，在C/C++语言中，所有的函数定义，包括主函数 main 在内，都是平行的。也就是说，在一个函数的函数体内，不能再定义另一个函数，即不能嵌套定义。但是函数之间允许相互调用，也允许嵌套调用。习惯上把调用者称为主调函数。函数还可以自己调用自己，称为递归调用。main 函数是主函数，它可以调用其它函数，而不允许被其它函数调用。因此，C/C++程序的执行总是从 main 函数开始，完成对其它函数的调用后再返回到 main 函数，最后由 main函数结束整个程序。一个C/C++源程序必须有，也只能有一个主函数 main。
2. 库函数
3. C++标准库介绍

* C++ 标准库可以分为两部分：
  + 标准函数库： 这个库是由通用的、独立的、不属于任何类的函数组成的。函数库继承自 C 语言。C++ 标准库包含了所有的 C 标准库，为了支持类型安全，做了一定的添加和修改。
* 标准函数库分为以下几类：
* 输入/输出 I/O
* 字符串和字符处理
* 数学
* 时间、日期和本地化
* 动态分配
* 宽字符函数
* 其他
  + 面向对象类库： 这个库是类及其相关函数的集合。标准的 C++ 面向对象类库定义了大量支持一些常见操作的类，比如输入/输出 I/O、字符串处理、数值处理。
* 面向对象类库包含以下内容：
* 标准的 C++ I/O 类
* String 类
* 数值类
* STL 容器类
* STL 算法
* STL 函数对象
* STL 迭代器
* STL 分配器
* 本地化库
* 异常处理类
* 杂项支持库

1. C++库函数介绍

* 库函数：由C /C++定义，也不必在程序中作类型说明，只需在程序前包含有该函数原型的头文件即可在程序中直接调用。
* C /C++言的头文件中包括了各个标准库函数的函数原型。因此，凡是在程序中调用一个库函数时，都必须包含该函数原型所在的头文件。
* C++标准库头文件



* C++库函数



1. 函数的定义和返回值
2. 函数定义的一般形式

* 函数可以被看作是一个由用户定义的操作。一般来说，函数由一个名字来表示，函数的操作数，称为参数 （parameter）由一个位于括号中，并且用逗号分隔的参数表（parameter list）指定。函数的结果被称为返回值（return value）返问值的类型被称为函数返回类型（return type）。不产生值的函数，返回类型是 void 意思是什么都不返回。函数执行的动作在函数体（body）中指定，函数体包含在花括号中。有时也称为函数块（function block）。函数返回类型，以及其后的函数名，参数表和函数体构成了函数定义。
* 无参函数的定义形式

**类型标识符 函数名()**

**{**

**声明部分**

**语句**

**}**

* + 其中类型标识符和函数名称为函数头。类型标识符指明了本函数的类型，函数的类型实际上是函数返回值的类型。 该类型标识符与前面介绍的各种说明符相同。函 数名是由用户定义的标识符，函数名后有一个空括号，其中无参数，但括号不可少。{}中的内容称为函数体。在函数体中**声明部分**，是对函数体内部所用到的变量的类型说明。
  + 在很多情况下都不要求无参函数有返回值，此时函数类型符可以写为 void。我们可以改写一个函数定义：

**void Hello()**

**{**

**printf ("Hello,world \n");**

**}**

这里，只把 main 改为 Hello 作为函数名，其余不变。Hello 函数是一个无参函数，当被其它函数调用时，输出 Hello world 字符串。

* 有参函数定义的一般形式

**类型标识符 函数名(形式参数表列)**

**{**

**声明部分**

**语句**

**}**

* 有参函数比无参函数多了一个内容，即形式参数表列。在形参表中给出的参数称为形式参数，它们可以是各种类型的变量，各参数之间用逗号间隔。在进行函数调用时，主调函数将赋予这些形式参数实际的值。形参既然是变量，必须在形参表中给出形参的类型说明。
* 例如，定义一个函数，用于求两个数中的大数，可写为：

**int max(int a, int b)**

**{**

**if (a>b) return a;**

**else return b;**

**}**

* 有参函数的形参和实参

前面已经介绍过，函数的参数分为形参和实参两种。形参出现在函数定义中，在整个函数体内都可以使用，离开该函数则不能使用。实参出现在主调函数中，进入被调函数后，实参变量也不能使用。形参和实参的功能是作数据传送。发生函数调用时，主调函数把实参的值传送给被调函数的形参从而实现主调函数向被调函数的数据传送。

函数的形参和实参具有以下特点：

* 形参变量只有在被调用时才分配内存单元，在调用结束时，即刻释放所分配的内存单元。因此，形参只有在函数内部有效。函数调用结束返回主调函数后则不能再使用该形参变量。
* 实参可以是常量、变量、表达式、函数等，无论实参是何种类型的量，在进行函数调用时，它们都必须具有确定的值，以便把这些值传送给形参。因此应预先用赋值，输入等办法使实参获得确定值。
* 实参和形参在数量上,类型上，顺序上应严格一致，否则会发生类型不匹配的错误。
* 函数调用中发生的数据传送是单向的。即只能把实参的值传送给形参，而不能把形参的值反向地传送给实参。因此在函数调用过程中，形参的值发生改变，而实参中的值不会变化。

例如：以下代码中，变量x，y是实参，变量a，b是形参。

**int main（）{**

**int x,y,z;**

**scanf(“%d%d”,&x,&y);**

**z=max(x,y);**

**printf(“%d”,z);**

**}**

**int max(int a, int b)**

**{**

**if (a>b) return a;**

**else return b;**

**}**

* 局部变量和全局变量

在讨论函数的形参变量时曾经提到，形参变量只在被调用期间才分配内存单元，调用结束立即释放。这一点表明形参变量只有在函数内才是有效的，离开该函数就不能再使用了。这种变量有效性的范围称变量的作用域。不仅对于形参变量，C/C++语言中所有的量都有自己的作用域。变量说明的方式不同，其作用域也不同。C/C++语言中的变量，按作用域范围可分为两种，即局部变量和全局变量。

* 局部变量

局部变量也称为内部变量。局部变量是在函数内作定义说明的。其作用域仅限于函数内，离开该函数后再使用这种变量是非法的。

* 主函数中定义的变量也只能在主函数中使用，不能在其它函数中使用。同时，主函数中也不能使用其它函数中定义的变量。因为主函数也是一个函数，它与其它函数是平行关系。这一点是与其它语言不同的，应予以注意。
* 形参变量是属于被调函数的局部变量，实参变量是属于主调函数的局部变量。
* 允许在不同的函数中使用相同的变量名，它们代表不同的对象，分配不同的单元，互不干扰，也不会发生混淆。如在前例中，形参和实参的变量名都为 n，是完全允许的。
* 在复合语句中也可定义变量，其作用域只在复合语句范围内。
  + 全局变量

全局变量也称为外部变量,它是在函数外部定义的变量。它不属于哪一个函数,它属于一个源程序文件,其作用域是整个源程序。在函数中使用全局变量,一般应作全局变量说明。只有在函数内经过说明的全局变量才能使用。全局变量的说明符为 extern。但在一个函数之前定义的全局变量,在该函数内使用可不再加以说明。外部变量（即全局变量）是在函数的外部定义的，它的作用域为从变量定义处开始，到本程序文件的末尾。如果外部变量不在文件的开头定义，其有效的作用范围只限于定义处到文件终了。如果在定义点之前的函数想引用该外部变量，则应该在引用之前用关键字extern对该变量作“外部变量声明”。表示该变量是一个已经定义的外部变量。有了此声明，就可以从“声明”处起，合法地使用该外部变量。

* 动态存储方式与静态动态存储方式

从变量值存在的作时间（即生存期）角度来分，可以分为静态存储方式和动态存储方式。

静态存储方式：是指在程序运行期间分配固定的存储空间的方式。

动态存储方式：是在程序运行期间根据需要进行动态的分配存储空间的方式。

* + auto 变量

函数中的局部变量，如不专门声明为 static 存储类别，都是动态地分配存储空间的，数据存储在动态存储区中。函数中的形参和在函数中定义的变量（包括在复合语句中定义的变量），都属此类，在调用该函数时系统会给它们分配存储空间，在函数调用结束时就自动释放这些存储空间。这类局部变量称为自动变量。自动变量用关键字 auto 作存储类别的声明。关键字 auto 可以省略，auto 不写则隐含定为“自动存储类别”，属于动态存储方式。

* + static 声明局部变量

有时希望函数中的局部变量的值在函数调用结束后不消失而保留原值，这时就应该指定局部变量为“静态局部变量”，用关键字 static 进行声明。

对静态局部变量的说明：

* + - 静态局部变量属于静态存储类别，在静态存储区内分配存储单元。在程序整个运行期间都不释放。而自动变量（即动态局部变量）属于动态存储类别，占动态存储空间，函数调用结束后即释放。
    - 静态局部变量在编译时赋初值，即只赋初值一次；而对自动变量赋初值是在函数调用时进行，每调用一次函数重新给一次初值，相当于执行一次赋值语句。
    - 如果在定义局部变量时不赋初值的话，则对静态局部变量来说，编译时自动赋初值 0（对数值型变量）或空字符（对字符变量）。而对自动变量来说，如果不赋初值则它的值是一个不确定的值。

1. 函数的返回值

函数的值是指函数被调用之后，执行函数体中的程序段所取得的并返回给主调函数的值。如调用正弦函数取得正弦值，调用max 函数取得的最大数等。对函数的值(或称函数返回值)有以下一些说明：

* 函数的值只能通过 return 语句返回主调函数。

return 语句的一般形式为：

**return 表达式；**

或者为：

**return (表达式)；**

该语句的功能是计算表达式的值，并返回给主调函数。在函数中允许有多个 return

语句，但每次调用只能有一个 return 语句被执行，因此只能返回一个函数值。

* 函数值的类型和函数定义中函数的类型应保持一致。如果两者不一致，则以函数类型为准，自动进行类型转换。
* 如函数值为整型，在函数定义时可以省去类型说明。
* 不返回函数值的函数，可以明确定义为“空类型”，类型说明符为“void”。可定义为：

**void s(int n)**

**{**

**……**

**}**

一旦函数被定义为空类型后，就不能在主调函数中使用被调函数的函数值了。为了使程序有良好的可读性并减少出错， 凡不要求返回值的函数都应定义为空类型。

1. 函数的调用
2. 函数调用的一般形式

C/C++语言中，函数调用的一般形式为：

**函数名(实际参数表)**

对无参函数调用时则无实际参数表。实际参数表中的参数可以是常数，变量或其它构造类型数据及表达式。各实参之间用逗号分隔。

1. 函数调用方式

* 函数表达式：

函数作为表达式中的一项出现在表达式中，以函数返回值参与表达式的运算。这种方式要求函数是有返回值的。例如：**z=max(x,y)**是一个赋值表达式，把 max的返回值赋予变量 z。

* 函数语句：

函数调用的一般形式加上分号即构成函数语句。

例如：

**printf("%d",a);**

**scanf ("%d",&b);**

都是以函数语句的方式调用函数。

* 函数实参：

函数作为另一个函数调用的实际参数出现。这种情况是把该函数的返回值作为实参进行传送，因此要求该函数必须是有返回值的。

例如：

**printf("%d",max(x,y));**

1. 被调用函数的声明和函数原型

* 在主调函数中调用某函数之前应对该被调函数进行声明，这与使用变量之前要先进行变量声明是一样的。在主调函数中对被调函数作说明的目的是使编译系统知道被调函数返回值的类型，以便在主调函数中按此种类型对返回值作相应的处理。

其一般形式为：

**类型说明符 被调函数名(类型 形参，类型 形参…)；**

或为：

**类型说明符 被调函数名(类型，类型…)；**

括号内给出了形参的类型和形参名，或只给出形参类型。这便于编译系统进行检错，以防止可能出现的错误。

* C/C++语言中又规定在以下几种情况时可以省去主调函数中对被调函数的函数说明。
  + 如果被调函数的返回值是整型或字符型时，可以不对被调函数作说明，而直接调用。这时系统将自动对被调函数返回值按整型处理。
  + 当被调函数的函数定义出现在主调函数之前时，在主调函数中也可以不对被调函数再作说明而直接调用。
  + 如在所有函数定义之前，在函数外预先说明了各个函数的类型，则在以后的各主调函数中，可不再对被调函数作说明。例如：

**char str(int a);**

**float f(float b);**

**main()**

**{**

**…**

**str(x);**

**f(y);**

**…**

**}**

**char str(int a)**

**{**

**……**

**}**

**float f(float b)**

**{**

**……**

**}**

* + 其中第一，二行对str()函数和f() 函数预先作了说明。因此在以后各函数中无须对 str（）和 f（）函数再作说明就可直接调用。
  + 对库函数的调用不需要再作说明，但必须把该函数的头文件用 include 命令包含在源文件前部。

1. 调用函数和被调用函数之间的数据传递
2. 数组
3. 一维数组的定义
4. 二维数组的定义
5. 字符串
6. 用一维字符数组存放字符串
7. 字符串的输入与输出
8. 字符串数组
9. 用字符串处理的函数
10. 文件和结构
11. 文件操作
12. 结构体
13. C++常用模板
14. sort
15. string

## 常用数据结构和算法

1. 排序算法
2. 冒泡排序
3. 选择排序
4. 快速排序
5. 归并排序（逆序对）
6. 递推和贪心算法
7. 递推
8. 贪心
9. 栈和队列
10. 栈
11. 队列
12. 单调队列
13. 搜索
14. 深度优先搜索（DFS）
15. 宽度优先搜索（BFS）