C教程

v1.1

# 前言

C可用于高性能数值计算，文档基于Win10、TDM-GCC。

更新：<https://www.misaraty.cc/2019-04-16_c-c-/>。

misaraty

misaraty@163.com

2016.12.12

# 目录

[前言 1](#_Toc469344260)

[目录 2](#_Toc469344261)

[安装 3](#_Toc469344262)

[C程序结构 4](#_Toc469344263)

[C基本语法 5](#_Toc469344264)

[C数据类型 7](#_Toc469344265)

[C变量 10](#_Toc469344266)

[C常量 13](#_Toc469344267)

[C存储类 18](#_Toc469344268)

[C运算符 19](#_Toc469344269)

[C判断 25](#_Toc469344270)

[C循环 26](#_Toc469344271)

[C函数 35](#_Toc469344272)

[C作用域规则 38](#_Toc469344273)

[C数组 41](#_Toc469344274)

[C指针 46](#_Toc469344275)

[C字符串 49](#_Toc469344276)

[C结构体 51](#_Toc469344277)

[C共用体 52](#_Toc469344278)

[C输入&输出 53](#_Toc469344279)

[C文件读写 55](#_Toc469344280)

[C预处理器 57](#_Toc469344281)

[C头文件 58](#_Toc469344282)

[C递归 59](#_Toc469344283)

[调用函数 60](#_Toc469344284)

[C 语言经典例子 62](#_Toc469344285)

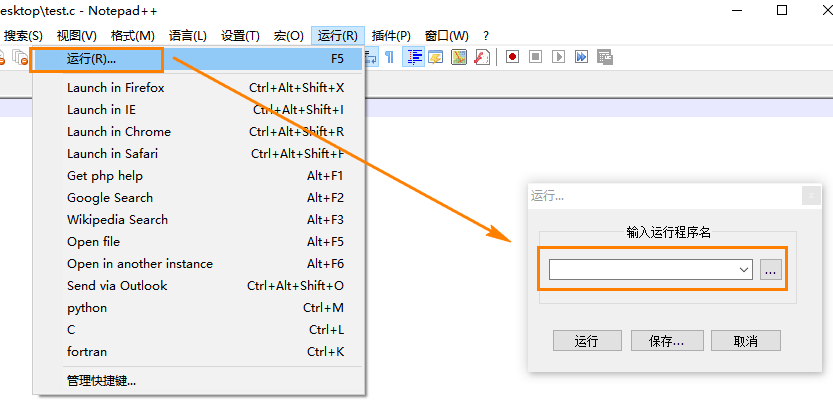
[参考 68](#_Toc469344286)

# 安装

下载[TDM-GCC](http://tdm-gcc.tdragon.net/)，64位，默认加入path。

配置notepad++，

输入cmd /k gcc -o "$(CURRENT\_DIRECTORY)\$(NAME\_PART).exe" "$(FULL\_CURRENT\_PATH)" && CLS && "$(CURRENT\_DIRECTORY)\$(NAME\_PART).exe" & PAUSE & EXIT



Linux下编译，

gcc \*.c

生成为\*.out新文件，

./\*.out即可运行。

若是保存该可执行文件，需要对其重命名。

windows采用GB2312编码，gcc在win下编译，中文乱码；而linux采用UTF-8编码，gcc编译后中文正常。

# C程序结构

C 程序主要包括以下部分：

预处理器指令

函数

变量

语句 & 表达式

注释

让我们看一段简单的代码，可以输出单词 "Hello World"：

#include <stdio.h>

int main()

{

/\* 我的第一个 C 程序 \*/

printf("Hello, World! \n");

return 0;

}

接下来我们讲解一下上面这段程序：

1.程序的第一行 #include <stdio.h> 是预处理器指令，告诉 C 编译器在实际编译之前要包含 stdio.h 文件。

2.下一行 int main() 是主函数，程序从这里开始执行。

3.下一行 /\*...\*/ 将会被编译器忽略，这里放置程序的注释内容。它们被称为程序的注释。

4.下一行 printf(...) 是 C 中另一个可用的函数，会在屏幕上显示消息 "Hello, World!"。

5.下一行 return 0; 终止 main() 函数，并返回值 0。

编译 & 执行 C 程序

接下来让我们看看如何把源代码保存在一个文件中，以及如何编译并运行它。下面是简单的步骤：

1.打开一个文本编辑器，添加上述代码。

2.保存文件为 hello.c。

3.打开命令提示符，进入到保存文件所在的目录。

4.键入 gcc hello.c，输入回车，编译代码。

5.如果代码中没有错误，命令提示符会跳到下一行，并生成 a.out 可执行文件。

6.现在，键入 a.out 来执行程序。

7.您可以看到屏幕上显示 "Hello World"。

$ gcc hello.c

$ ./a.out

Hello, World!

# C基本语法

C 的令牌（Tokens）

C 程序由各种令牌组成，令牌可以是关键字、标识符、常量、字符串值，或者是一个符号。例如，下面的 C 语句包括五个令牌：

printf("Hello, World! \n");

这五个令牌分别是：

printf

(

"Hello, World! \n"

)

;

分号 ;

在 C 程序中，分号是语句结束符。也就是说，每个语句必须以分号结束。它表明一个逻辑实体的结束。

例如，下面是两个不同的语句：

printf("Hello, World! \n");

return 0;

注释

注释就像是 C 程序中的帮助文本，它们会被编译器忽略。它们以 /\* 开始，以字符 \*/ 终止，如下所示：

/\* 我的第一个 C 程序 \*/

您不能在注释内嵌套注释，注释也不能出现在字符串或字符值中。

标识符

C 标识符是用来标识变量、函数，或任何其他用户自定义项目的名称。一个标识符以字母 A-Z 或 a-z 或下划线 \_ 开始，后跟零个或多个字母、下划线和数字（0-9）。

C 标识符内不允许出现标点字符，比如 @、$ 和 %。C 是区分大小写的编程语言。因此，在 C 中，Manpower 和 manpower 是两个不同的标识符。下面列出几个有效的标识符：

mohd zara abc move\_name a\_123

myname50 \_temp j a23b9 retVal

关键字

下表列出了 C 中的保留字。这些保留字不能作为常量名、变量名或其他标识符名称。

auto else long switch

break enum register typedef

case extern return union

char float short unsigned

const for signed void

continue goto sizeof volatile

default if static while

do int struct \_Packed

double

C 中的空格

只包含空格的行，被称为空白行，可能带有注释，C 编译器会完全忽略它。

在 C 中，空格用于描述空白符、制表符、换行符和注释。空格分隔语句的各个部分，让编译器能识别语句中的某个元素（比如 int）在哪里结束，下一个元素在哪里开始。因此，在下面的语句中：

int age;

在这里，int 和 age 之间必须至少有一个空格字符（通常是一个空白符），这样编译器才能够区分它们。另一方面，在下面的语句中：

fruit = apples + oranges; // 获取水果的总数

fruit 和 =，或者 = 和 apples 之间的空格字符不是必需的，但是为了增强可读性，您可以根据需要适当增加一些空格。

# C数据类型

在 C 语言中，数据类型指的是用于声明不同类型的变量或函数的一个广泛的系统。变量的类型决定了变量存储占用的空间，以及如何解释存储的位模式。

C 中的类型可分为以下几种：

序号

类型与描述

1 基本类型：

它们是算术类型，包括两种类型：整数类型和浮点类型。

2 枚举类型：

它们也是算术类型，被用来定义在程序中只能赋予其一定的离散整数值的变量。

3 void 类型：

类型说明符 void 表明没有可用的值。

4 派生类型：

它们包括：指针类型、数组类型、结构类型、共用体类型和函数类型。

数组类型和结构类型统称为聚合类型。函数的类型指的是函数返回值的类型。

整数类型

下表列出了关于标准整数类型的存储大小和值范围的细节：

类型 存储大小 值范围

char 1 字节 -128 到 127 或 0 到 255

unsigned char 1 字节 0 到 255

signed char 1 字节 -128 到 127

int 2 或 4 字节 -32,768 到 32,767 或 -2,147,483,648 到 2,147,483,647

unsigned int 2 或 4 字节 0 到 65,535 或 0 到 4,294,967,295

short 2 字节 -32,768 到 32,767

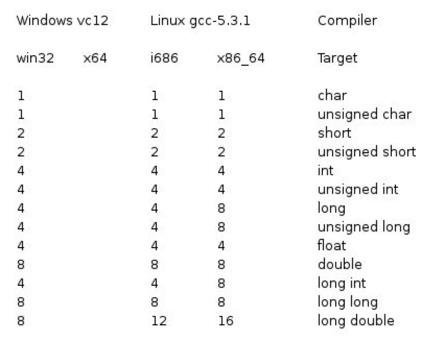
unsigned short 2 字节 0 到 65,535

long 4 字节 -2,147,483,648 到 2,147,483,647

unsigned long 4 字节 0 到 4,294,967,295

注意，各种类型的存储大小与系统位数有关，但目前通用的以64位系统为主。

以下列出了32位系统与64位系统的存储大小的差别（windows 相同）：



为了得到某个类型或某个变量在特定平台上的准确大小，您可以使用 sizeof 运算符。表达式 sizeof(type) 得到对象或类型的存储字节大小。下面的实例演示了获取 int 类型的大小：

#include <stdio.h>

#include <limits.h>

int main()

{

printf("int 存储大小 : %lu \n", sizeof(int));

return 0;

}

当您在 Linux 上编译并执行上面的程序时，它会产生下列结果：

int 存储大小 : 4

当您在 Win 上编译并执行上面的程序时，它会产生下列结果：

int 瀛樺偍澶у皬 : 4

浮点类型

下表列出了关于标准浮点类型的存储大小、值范围和精度的细节：

类型 存储大小 值范围 精度

float 4 字节 1.2E-38 到 3.4E+38 6 位小数

double 8 字节 2.3E-308 到 1.7E+308 15 位小数

long double 16 字节 3.4E-4932 到 1.1E+4932 19 位小数

头文件 float.h 定义了宏，在程序中可以使用这些值和其他有关实数二进制表示的细节。下面的实例将输出浮点类型占用的存储空间以及它的范围值：

#include <stdio.h>

#include <float.h>

int main()

{

printf("float 存储最大字节数 : %lu \n", sizeof(float));

printf("float 最小值: %E\n", FLT\_MIN );

printf("float 最大值: %E\n", FLT\_MAX );

printf("精度值: %d\n", FLT\_DIG );

return 0;

}

当您在 Linux 上编译并执行上面的程序时，它会产生下列结果：

float 存储最大字节数 : 4

float 最小值: 1.175494E-38

float 最大值: 3.402823E+38

精度值: 6

void 类型

void 类型指定没有可用的值。它通常用于以下三种情况下：

序号 类型与描述

1 函数返回为空

C 中有各种函数都不返回值，或者您可以说它们返回空。不返回值的函数的返回类型为空。例如 void exit (int status);

2 函数参数为空

C 中有各种函数不接受任何参数。不带参数的函数可以接受一个 void。例如 int rand(void);

3 指针指向 void

类型为 void \* 的指针代表对象的地址，而不是类型。例如，内存分配函数 void \*malloc( size\_t size ); 返回指向 void 的指针，可以转换为任何数据类型。

# C变量

变量其实只不过是程序可操作的存储区的名称。C 中每个变量都有特定的类型，类型决定了变量存储的大小和布局，该范围内的值都可以存储在内存中，运算符可应用于变量上。

变量的名称可以由字母、数字和下划线字符组成。它必须以字母或下划线开头。大写字母和小写字母是不同的，因为 C 是大小写敏感的。基于前一章讲解的基本类型，有以下几种基本的变量类型：

类型 描述

char 通常是一个八位字节（一个字节）。这是一个整数类型。

int 对机器而言，整数的最自然的大小。

float 单精度浮点值。

double 双精度浮点值。

void 表示类型的缺失。

C 语言也允许定义各种其他类型的变量，比如枚举、指针、数组、结构、共用体等等，这将会在后续的章节中进行讲解，本章节我们先讲解基本变量类型。

C 中的变量定义

变量定义就是告诉编译器在何处创建变量的存储，以及如何创建变量的存储。变量定义指定一个数据类型，并包含了该类型的一个或多个变量的列表，如下所示：

type variable\_list;

在这里，type 必须是一个有效的 C 数据类型，可以是 char、w\_char、int、float、double、bool 或任何用户自定义的对象，variable\_list 可以由一个或多个标识符名称组成，多个标识符之间用逗号分隔。下面列出几个有效的声明：

int i, j, k;

char c, ch;

float f, salary;

double d;

行 int i, j, k; 声明并定义了变量 i、j 和 k，这指示编译器创建类型为 int 的名为 i、j、k 的变量。

变量可以在声明的时候被初始化（指定一个初始值）。初始化器由一个等号，后跟一个常量表达式组成，如下所示：

type variable\_name = value;

下面列举几个实例：

extern int d = 3, f = 5; // d 和 f 的声明, 这就是单纯的声明

int d = 3, f = 5; // 定义并初始化 d 和 f

byte z = 22; // 定义并初始化 z

char x = 'x'; // 变量 x 的值为 'x'

不带初始化的定义：带有静态存储持续时间的变量会被隐式初始化为 NULL（所有字节的值都是 0），其他所有变量的初始值是未定义的。

C 中的变量声明

变量声明向编译器保证变量以指定的类型和名称存在，这样编译器在不需要知道变量完整细节的情况下也能继续进一步的编译。变量声明只在编译时有它的意义，在程序连接时编译器需要实际的变量声明。

变量的声明有两种情况：

1、一种是需要建立存储空间的。例如：int a 在声明的时候就已经建立了存储空间。

2、另一种是不需要建立存储空间的，通过使用extern关键字声明变量名而不定义它。 例如：extern int a 其中变量 a 可以在别的文件中定义的。

除非有extern关键字，否则都是变量的定义。

extern int i; //声明，不是定义

int i; //声明，也是定义

实例

尝试下面的实例，其中，变量在头部就已经被声明，但是定义与初始化在主函数内：

#include <stdio.h>

// 变量声明

extern int a, b;

extern int c;

extern float f;

int main ()

{

/\* 变量定义 \*/

int a, b;

int c;

float f;

/\* 初始化 \*/

a = 10;

b = 20;

c = a + b;

printf("value of c : %d \n", c);

f = 70.0/3.0;

printf("value of f : %f \n", f);

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

value of c : 30

value of f : 23.333334

C 中的左值（Lvalues）和右值（Rvalues）

C 中有两种类型的表达式：

1.左值（lvalue）：指向内存位置的表达式被称为左值（lvalue）表达式。左值可以出现在赋值号的左边或右边。

2.右值（rvalue）：术语右值（rvalue）指的是存储在内存中某些地址的数值。右值是不能对其进行赋值的表达式，也就是说，右值可以出现在赋值号的右边，但不能出现在赋值号的左边。

变量是左值，因此可以出现在赋值号的左边。数值型的字面值是右值，因此不能被赋值，不能出现在赋值号的左边。下面是一个有效的语句：

int g = 20;

但是下面这个就不是一个有效的语句，会生成编译时错误：

10 = 20;

# C常量

常量是固定值，在程序执行期间不会改变。这些固定的值，又叫做字面量。

常量可以是任何的基本数据类型，比如整数常量、浮点常量、字符常量，或字符串字面值，也有枚举常量。

常量就像是常规的变量，只不过常量的值在定义后不能进行修改。

整数常量

整数常量可以是十进制、八进制或十六进制的常量。前缀指定基数：0x 或 0X 表示十六进制，0 表示八进制，不带前缀则默认表示十进制。

整数常量也可以带一个后缀，后缀是 U 和 L 的组合，U 表示无符号整数（unsigned），L 表示长整数（long）。后缀可以是大写，也可以是小写，U 和 L 的顺序任意。

下面列举几个整数常量的实例：

212 /\* 合法的 \*/

215u /\* 合法的 \*/

0xFeeL /\* 合法的 \*/

078 /\* 非法的：8 不是八进制的数字 \*/

032UU /\* 非法的：不能重复后缀 \*/

以下是各种类型的整数常量的实例：

85 /\* 十进制 \*/

0213 /\* 八进制 \*/

0x4b /\* 十六进制 \*/

30 /\* 整数 \*/

30u /\* 无符号整数 \*/

30l /\* 长整数 \*/

30ul /\* 无符号长整数 \*/

浮点常量

实型常量又称实数或浮点数。在C语言中可以用两种形式表示一个实型常量。

小数形式是由数字和小数点组成的一种实数表示形式，例如0.123、.123、123.、0.0等都是合法的实型常量。

注意：小数形式表示的实型常量必须要有小数点。

指数形式。在数学中，一个可以用幂的形式来表示，如2.3026可以表示为0.23026×10^1 2.3026×10^0 23.026×10^-1等形式。在C语言中，则以“e”或“E”后跟一个整数来表示以“10”为底数的幂数。2.3026可以表示为0.23026E1、2.3026e0、23.026e-1。C语言语法规定，字母e或E之前必须要有数字，且e或E后面的指数必须为整数。如e3、5e3.6、.e、e等都是非法的指数形式。注意：在字母e或E的前后以及数字之间不得插入空格。

程序运行的过程中，其值不能被改变的量称为常量。常量有不同类型，其中12、0、-5为整型常量。'a''b'为字符常量。而4.6、-8.7则为实型常量。

一个实型常量可以赋给一个 float 型、double 型或 long double 变量。根据变量的类型截取实型常量中相应的有效位数字。

字符常量

字符常量是括在单引号中，例如，'x' 可以存储在 char 类型的简单变量中。

字符常量可以是一个普通的字符（例如 'x'）、一个转义序列（例如 '\t'），或一个通用的字符（例如 '\u02C0'）。

在 C 中，有一些特定的字符，当它们前面有反斜杠时，它们就具有特殊的含义，被用来表示如换行符（\n）或制表符（\t）等。下表列出了一些这样的转义序列码：

转义序列 含义

\\ \ 字符

\' ' 字符

\" " 字符

\? ? 字符

\a 警报铃声

\b 退格键

\f 换页符

\n 换行符

\r 回车

\t 水平制表符

\v 垂直制表符

\ooo 一到三位的八进制数

\xhh . . . 一个或多个数字的十六进制数

下面的实例显示了一些转义序列字符：

#include <stdio.h>

int main()

{

printf("Hello\tWorld\n\n");

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

Hello World

字符串常量

字符串字面值或常量是括在双引号 "" 中的。一个字符串包含类似于字符常量的字符：普通的字符、转义序列和通用的字符。

您可以使用空格做分隔符，把一个很长的字符串常量进行分行。

下面的实例显示了一些字符串常量。下面这三种形式所显示的字符串是相同的。

"hello, dear"

"hello, \

dear"

"hello, " "d" "ear"

定义常量

在 C 中，有两种简单的定义常量的方式：

1.使用 #define 预处理器。

2.使用 const 关键字。

#define 预处理器

下面是使用 #define 预处理器定义常量的形式：

#define identifier value

具体请看下面的实例：

#include <stdio.h>

#define LENGTH 10

#define WIDTH 5

#define NEWLINE '\n'

int main()

{

int area;

area = LENGTH \* WIDTH;

printf("value of area : %d", area);

printf("%c", NEWLINE);

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

value of area : 50

const 关键字

您可以使用 const 前缀声明指定类型的常量，如下所示：

const type variable = value;

具体请看下面的实例：

#include <stdio.h>

int main()

{

const int LENGTH = 10;

const int WIDTH = 5;

const char NEWLINE = '\n';

int area;

area = LENGTH \* WIDTH;

printf("value of area : %d", area);

printf("%c", NEWLINE);

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

value of area : 50

请注意，把常量定义为大写字母形式，是一个很好的编程实践。

# C存储类

存储类定义 C 程序中变量/函数的范围（可见性）和生命周期。这些说明符放置在它们所修饰的类型之前。下面列出 C 程序中可用的存储类：

auto

register

static

extern

auto 存储类

auto 存储类是所有局部变量默认的存储类。

register 存储类

register 存储类用于定义存储在寄存器中而不是 RAM 中的局部变量。这意味着变量的最大尺寸等于寄存器的大小（通常是一个词），且不能对它应用一元的 '&' 运算符（因为它没有内存位置）。

static 存储类

static 存储类指示编译器在程序的生命周期内保持局部变量的存在，而不需要在每次它进入和离开作用域时进行创建和销毁。因此，使用 static 修饰局部变量可以在函数调用之间保持局部变量的值。

static 修饰符也可以应用于全局变量。当 static 修饰全局变量时，会使变量的作用域限制在声明它的文件内。

在 C 编程中，当 static 用在类数据成员上时，会导致仅有一个该成员的副本被类的所有对象共享。

extern 存储类

extern 存储类用于提供一个全局变量的引用，全局变量对所有的程序文件都是可见的。当您使用 'extern' 时，对于无法初始化的变量，会把变量名指向一个之前定义过的存储位置。

当您有多个文件且定义了一个可以在其他文件中使用的全局变量或函数时，可以在其他文件中使用 extern 来得到已定义的变量或函数的引用。可以这么理解，extern 是用来在另一个文件中声明一个全局变量或函数。

# C运算符

运算符是一种告诉编译器执行特定的数学或逻辑操作的符号。C 语言内置了丰富的运算符，并提供了以下类型的运算符：

算术运算符

关系运算符

逻辑运算符

位运算符

赋值运算符

杂项运算符

本章将逐一介绍算术运算符、关系运算符、逻辑运算符、位运算符、赋值运算符和其他运算符。

算术运算符

下表显示了 C 语言支持的所有算术运算符。假设变量 A 的值为 10，变量 B 的值为 20，则：

运算符 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

实例

请看下面的实例，了解 C 语言中所有可用的算术运算符：

#include <stdio.h>

main()

{

int a = 21;

int b = 10;

int c ;

c = a + b;

printf("Line 1 - c 的值是 %d\n", c );

c = a - b;

printf("Line 2 - c 的值是 %d\n", c );

c = a \* b;

printf("Line 3 - c 的值是 %d\n", c );

c = a / b;

printf("Line 4 - c 的值是 %d\n", c );

c = a % b;

printf("Line 5 - c 的值是 %d\n", c );

c = a++;

printf("Line 6 - c 的值是 %d\n", c );

c = a--;

printf("Line 7 - c 的值是 %d\n", c );

}

当上面的代码被编译和执行时，它会产生下列结果：

Line 1 - c 的值是 31

Line 2 - c 的值是 11

Line 3 - c 的值是 210

Line 4 - c 的值是 2

Line 5 - c 的值是 1

Line 6 - c 的值是 21

Line 7 - c 的值是 22

关系运算符

下表显示了 C 语言支持的所有关系运算符。假设变量 A 的值为 10，变量 B 的值为 20，则：

运算符 描述 实例

== 检查两个操作数的值是否相等，如果相等则条件为真。 (A == B) 不为真。

!= 检查两个操作数的值是否相等，如果不相等则条件为真。 (A != B) 为真。

> 检查左操作数的值是否大于右操作数的值，如果是则条件为真。 (A > B) 不为真。

< 检查左操作数的值是否小于右操作数的值，如果是则条件为真。 (A < B) 为真。

>= 检查左操作数的值是否大于或等于右操作数的值，如果是则条件为真。 (A >= B) 不为真。

<= 检查左操作数的值是否小于或等于右操作数的值，如果是则条件为真。 (A <= B) 为真。

实例

请看下面的实例，了解 C 语言中所有可用的关系运算符：

#include <stdio.h>

main()

{

int a = 21;

int b = 10;

int c ;

if( a == b )

{

printf("Line 1 - a 等于 b\n" );

}

else

{

printf("Line 1 - a 不等于 b\n" );

}

if ( a < b )

{

printf("Line 2 - a 小于 b\n" );

}

else

{

printf("Line 2 - a 不小于 b\n" );

}

if ( a > b )

{

printf("Line 3 - a 大于 b\n" );

}

else

{

printf("Line 3 - a 不大于 b\n" );

}

/\* 改变 a 和 b 的值 \*/

a = 5;

b = 20;

if ( a <= b )

{

printf("Line 4 - a 小于或等于 b\n" );

}

if ( b >= a )

{

printf("Line 5 - b 大于或等于 b\n" );

}

}

当上面的代码被编译和执行时，它会产生下列结果：

Line 1 - a 不等于 b

Line 2 - a 不小于 b

Line 3 - a 大于 b

Line 4 - a 小于或等于 b

Line 5 - b 大于或等于 b

逻辑运算符

下表显示了 C 语言支持的所有关系逻辑运算符。假设变量 A 的值为 1，变量 B 的值为 0，则：

运算符 描述 实例

&& 称为逻辑与运算符。如果两个操作数都非零，则条件为真。 (A && B) 为假。

|| 称为逻辑或运算符。如果两个操作数中有任意一个非零，则条件为真。 (A || B) 为真。

! 称为逻辑非运算符。用来逆转操作数的逻辑状态。如果条件为真则逻辑非运算符将使其为假。 !(A && B) 为真。

位运算符

位运算符作用于位，并逐位执行操作。&、 | 和 ^ 的真值表如下所示：

p q p&q p|q p^q

0 0 0 0 0

0 1 0 1 1

1 1 1 1 0

1 0 0 1 1

赋值运算符

下表列出了 C 语言支持的赋值运算符：

运算符 描述 实例

= 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

杂项运算符 ↦ sizeof & 三元

下表列出了 C 语言支持的其他一些重要的运算符，包括 sizeof 和 ? :。

运算符 描述 实例

sizeof() 返回变量的大小。 sizeof(a) 将返回 4，其中 a 是整数。

& 返回变量的地址。 &a; 将给出变量的实际地址。

\* 指向一个变量。 \*a; 将指向一个变量。

? : 条件表达式 如果条件为真 ? 则值为 X : 否则值为 Y

C 中的运算符优先级

运算符的优先级确定表达式中项的组合。这会影响到一个表达式如何计算。某些运算符比其他运算符有更高的优先级，例如，乘除运算符具有比加减运算符更高的优先级。

例如 x = 7 + 3 \* 2，在这里，x 被赋值为 13，而不是 20，因为运算符 \* 具有比 + 更高的优先级，所以首先计算乘法 3\*2，然后再加上 7。

下表将按运算符优先级从高到低列出各个运算符，具有较高优先级的运算符出现在表格的上面，具有较低优先级的运算符出现在表格的下面。在表达式中，较高优先级的运算符会优先被计算。

类别 运算符 结合性

后缀 () [] -> . ++ - - 从左到右

一元 + - ! ~ ++ - - (type)\* & sizeof 从右到左

乘除 \* / % 从左到右

加减 + - 从左到右

移位 << >> 从左到右

关系 < <= > >= 从左到右

相等 == != 从左到右

位与 AND & 从左到右

位异或 XOR ^ 从左到右

位或 OR | 从左到右

逻辑与 AND && 从左到右

逻辑或 OR || 从左到右

条件 ?: 从右到左

赋值 = += -= \*= /= %=>>= <<= &= ^= |= 从右到左

逗号 , 从左到右

# C判断

判断结构要求程序员指定一个或多个要评估或测试的条件，以及条件为真时要执行的语句（必需的）和条件为假时要执行的语句（可选的）。

C 语言把任何非零和非空的值假定为 true，把零或 null 假定为 false。

判断语句

C 语言提供了以下类型的判断语句。点击链接查看每个语句的细节。

语句 描述

if 语句 一个 if 语句 由一个布尔表达式后跟一个或多个语句组成。

if...else 语句 一个 if 语句 后可跟一个可选的 else 语句，else 语句在布尔表达式为假时执行。

嵌套 if 语句 您可以在一个 if 或 else if 语句内使用另一个 if 或 else if 语句。

switch 语句 一个 switch 语句允许测试一个变量等于多个值时的情况。

嵌套 switch 语句 您可以在一个 switch 语句内使用另一个 switch 语句。

? : 运算符

我们已经在前面的章节中讲解了 条件运算符 ? :，可以用来替代 if...else 语句。它的一般形式如下：

Exp1 ? Exp2 : Exp3;

其中，Exp1、Exp2 和 Exp3 是表达式。请注意，冒号的使用和位置。

? 表达式的值是由 Exp1 决定的。如果 Exp1 为真，则计算 Exp2 的值，结果即为整个 ? 表达式的值。如果 Exp1 为假，则计算 Exp3 的值，结果即为整个 ? 表达式的值。

# C循环

有的时候，我们可能需要多次执行同一块代码。一般情况下，语句是按顺序执行的：函数中的第一个语句先执行，接着是第二个语句，依此类推。

编程语言提供了更为复杂执行路径的多种控制结构。

循环语句允许我们多次执行一个语句或语句组。

循环类型

C 语言提供了以下几种循环类型。点击链接查看每个类型的细节。

循环类型 描述

while 循环 当给定条件为真时，重复语句或语句组。它会在执行循环主体之前测试条件。

for 循环 多次执行一个语句序列，简化管理循环变量的代码。

do...while 循环 除了它是在循环主体结尾测试条件外，其他与 while 语句类似。

嵌套循环 您可以在 while、for 或 do..while 循环内使用一个或多个循环。

while 循环

语法

C 语言中 while 循环的语法：

while(condition)

{

statement(s);

}

在这里，statement(s) 可以是一个单独的语句，也可以是几个语句组成的代码块。condition 可以是任意的表达式，当为任意非零值时都为 true。当条件为 true 时执行循环。

当条件为 false 时，程序流将继续执行紧接着循环的下一条语句。

#include <stdio.h>

int main ()

{

/\* 局部变量定义 \*/

int a = 10;

/\* while 循环执行 \*/

while( a < 20 )

{

printf("a 的值： %d\n", a);

a++;

}

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

a 的值： 10

a 的值： 11

a 的值： 12

a 的值： 13

a 的值： 14

a 的值： 15

a 的值： 16

a 的值： 17

a 的值： 18

a 的值： 19

for 循环

for 循环允许您编写一个执行指定次数的循环控制结构。

语法

C 语言中 for 循环的语法：

for ( init; condition; increment )

{

statement(s);

}

下面是 for 循环的控制流：

1.init 会首先被执行，且只会执行一次。这一步允许您声明并初始化任何循环控制变量。您也可以不在这里写任何语句，只要有一个分号出现即可。

2.接下来，会判断 condition。如果为真，则执行循环主体。如果为假，则不执行循环主体，且控制流会跳转到紧接着 for 循环的下一条语句。

3.在执行完 for 循环主体后，控制流会跳回上面的 increment 语句。该语句允许您更新循环控制变量。该语句可以留空，只要在条件后有一个分号出现即可。

4.条件再次被判断。如果为真，则执行循环，这个过程会不断重复（循环主体，然后增加步值，再然后重新判断条件）。在条件变为假时，for 循环终止。

实例

#include <stdio.h>

int main ()

{

/\* for 循环执行 \*/

for( int a = 10; a < 20; a = a + 1 )

{

printf("a 的值： %d\n", a);

}

return 0;

}

不像 for 和 while 循环，它们是在循环头部测试循环条件。在 C 语言中，do...while 循环是在循环的尾部检查它的条件。

do...while 循环与 while 循环类似，但是 do...while 循环会确保至少执行一次循环。

语法

C 语言中 do...while 循环的语法：

do

{

statement(s);

}while( condition );

请注意，条件表达式出现在循环的尾部，所以循环中的 statement(s) 会在条件被测试之前至少执行一次。

如果条件为真，控制流会跳转回上面的 do，然后重新执行循环中的 statement(s)。这个过程会不断重复，直到给定条件变为假为止。

实例

#include <stdio.h>

int main ()

{

/\* 局部变量定义 \*/

int a = 10;

/\* do 循环执行 \*/

do

{

printf("a 的值： %d\n", a);

a = a + 1;

}while( a < 20 );

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

a 的值： 10

a 的值： 11

a 的值： 12

a 的值： 13

a 的值： 14

a 的值： 15

a 的值： 16

a 的值： 17

a 的值： 18

a 的值： 19

嵌套循环

C 语言允许在一个循环内使用另一个循环，下面演示几个实例来说明这个概念。

语法

C 语言中 嵌套 for 循环 语句的语法：

for ( init; condition; increment )

{

for ( init; condition; increment )

{

statement(s);

}

statement(s);

}

C 语言中 嵌套 while 循环 语句的语法：

while(condition)

{

while(condition)

{

statement(s);

}

statement(s);

}

C 语言中 嵌套 do...while 循环 语句的语法：

do

{

statement(s);

do

{

statement(s);

}while( condition );

}while( condition );

关于嵌套循环有一点值得注意，您可以在任何类型的循环内嵌套其他任何类型的循环。比如，一个 for 循环可以嵌套在一个 while 循环内，反之亦然。

循环控制语句

循环控制语句改变你代码的执行顺序。通过它你可以实现代码的跳转。

C 提供了下列的循环控制语句。点击链接查看每个语句的细节。

控制语句 描述

break 语句 终止循环或 switch 语句，程序流将继续执行紧接着循环或 switch 的下一条语句。

continue 语句 告诉一个循环体立刻停止本次循环迭代，重新开始下次循环迭代。

goto 语句 将控制转移到被标记的语句。但是不建议在程序中使用 goto 语句。

break 语句

C 语言中 break 语句有以下两种用法：

1.当 break 语句出现在一个循环内时，循环会立即终止，且程序流将继续执行紧接着循环的下一条语句。

2.它可用于终止 switch 语句中的一个 case。

如果您使用的是嵌套循环（即一个循环内嵌套另一个循环），break 语句会停止执行最内层的循环，然后开始执行该块之后的下一行代码。

语法

C 语言中 break 语句的语法：

break;

实例

#include <stdio.h>

int main ()

{

/\* 局部变量定义 \*/

int a = 10;

/\* while 循环执行 \*/

while( a < 20 )

{

printf("a 的值： %d\n", a);

a++;

if( a > 15)

{

/\* 使用 break 语句终止循环 \*/

break;

}

}

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

a 的值： 10

a 的值： 11

a 的值： 12

a 的值： 13

a 的值： 14

a 的值： 15

continue 语句

C 语言中的 continue 语句有点像 break 语句。但它不是强迫终止，continue 会跳过当前循环中的代码，强迫开始下一次循环。

对于 for 循环，continue 语句执行后自增语句仍然会执行。对于 while 和 do...while 循环，continue 语句重新执行条件判断语句。

语法

C 语言中 continue 语句的语法：

continue;

实例

#include <stdio.h>

int main ()

{

/\* 局部变量定义 \*/

int a = 10;

/\* do 循环执行 \*/

do

{

if( a == 15)

{

/\* 跳过迭代 \*/

a = a + 1;

continue;

}

printf("a 的值： %d\n", a);

a++;

}while( a < 20 );

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

a 的值： 10

a 的值： 11

a 的值： 12

a 的值： 13

a 的值： 14

a 的值： 16

a 的值： 17

a 的值： 18

a 的值： 19

goto 语句

C 语言中的 goto 语句允许把控制无条件转移到同一函数内的被标记的语句。

注意：在任何编程语言中，都不建议使用 goto 语句。因为它使得程序的控制流难以跟踪，使程序难以理解和难以修改。任何使用 goto 语句的程序可以改写成不需要使用 goto 语句的写法。

语法

C 语言中 goto 语句的语法：

goto label;

..

.

label: statement;

在这里，label 可以是任何除 C 关键字以外的纯文本，它可以设置在 C 程序中 goto 语句的前面或者后面。

实例

#include <stdio.h>

int main ()

{

/\* 局部变量定义 \*/

int a = 10;

/\* do 循环执行 \*/

LOOP:do

{

if( a == 15)

{

/\* 跳过迭代 \*/

a = a + 1;

goto LOOP;

}

printf("a 的值： %d\n", a);

a++;

}while( a < 20 );

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

a 的值： 10

a 的值： 11

a 的值： 12

a 的值： 13

a 的值： 14

a 的值： 16

a 的值： 17

a 的值： 18

a 的值： 19

无限循环

如果条件永远不为假，则循环将变成无限循环。for 循环在传统意义上可用于实现无限循环。由于构成循环的三个表达式中任何一个都不是必需的，您可以将某些条件表达式留空来构成一个无限循环。

#include <stdio.h>

int main ()

{

for( ; ; )

{

printf("This loop will run forever.\n");

}

return 0;

}

当条件表达式不存在时，它被假设为真。您也可以设置一个初始值和增量表达式，但是一般情况下，C 程序员偏向于使用 for(;;) 结构来表示一个无限循环。

注意：您可以按 Ctrl + C 键终止一个无限循环。

# C函数

函数是一组一起执行一个任务的语句。每个 C 程序都至少有一个函数，即主函数 main() ，所有简单的程序都可以定义其他额外的函数。

您可以把代码划分到不同的函数中。如何划分代码到不同的函数中是由您来决定的，但在逻辑上，划分通常是根据每个函数执行一个特定的任务来进行的。

函数声明告诉编译器函数的名称、返回类型和参数。函数定义提供了函数的实际主体。

C 标准库提供了大量的程序可以调用的内置函数。例如，函数 strcat() 用来连接两个字符串，函数 memcpy() 用来复制内存到另一个位置。

函数还有很多叫法，比如方法、子例程或程序，等等。

定义函数

C 语言中的函数定义的一般形式如下：

return\_type function\_name( parameter list )

{

body of the function

}

在 C 语言中，函数由一个函数头和一个函数主体组成。下面列出一个函数的所有组成部分：

返回类型：一个函数可以返回一个值。return\_type 是函数返回的值的数据类型。有些函数执行所需的操作而不返回值，在这种情况下，return\_type 是关键字 void。

函数名称：这是函数的实际名称。函数名和参数列表一起构成了函数签名。

参数：参数就像是占位符。当函数被调用时，您向参数传递一个值，这个值被称为实际参数。参数列表包括函数参数的类型、顺序、数量。参数是可选的，也就是说，函数可能不包含参数。

函数主体：函数主体包含一组定义函数执行任务的语句。

实例

以下是 max() 函数的源代码。该函数有两个参数 num1 和 num2，会返回这两个数中较大的那个数：

/\* 函数返回两个数中较大的那个数 \*/

int max(int num1, int num2)

{

/\* 局部变量声明 \*/

int result;

if (num1 > num2)

result = num1;

else

result = num2;

return result;

}

函数声明

函数声明会告诉编译器函数名称及如何调用函数。函数的实际主体可以单独定义。

函数声明包括以下几个部分：

return\_type function\_name( parameter list );

针对上面定义的函数 max()，以下是函数声明：

int max(int num1, int num2);

在函数声明中，参数的名称并不重要，只有参数的类型是必需的，因此下面也是有效的声明：

int max(int, int);

当您在一个源文件中定义函数且在另一个文件中调用函数时，函数声明是必需的。在这种情况下，您应该在调用函数的文件顶部声明函数。

调用函数

创建 C 函数时，会定义函数做什么，然后通过调用函数来完成已定义的任务。

当程序调用函数时，程序控制权会转移给被调用的函数。被调用的函数执行已定义的任务，当函数的返回语句被执行时，或到达函数的结束括号时，会把程序控制权交还给主程序。

调用函数时，传递所需参数，如果函数返回一个值，则可以存储返回值。例如：

#include <stdio.h>

/\* 函数声明 \*/

int max(int num1, int num2);

int main ()

{

/\* 局部变量定义 \*/

int a = 100;

int b = 200;

int ret;

/\* 调用函数来获取最大值 \*/

ret = max(a, b);

printf( "Max value is : %d\n", ret );

return 0;

}

/\* 函数返回两个数中较大的那个数 \*/

int max(int num1, int num2)

{

/\* 局部变量声明 \*/

int result;

if (num1 > num2)

result = num1;

else

result = num2;

return result;

}

把 max() 函数和 main() 函数放一块，编译源代码。当运行最后的可执行文件时，会产生下列结果：

Max value is : 200

函数参数

如果函数要使用参数，则必须声明接受参数值的变量。这些变量称为函数的形式参数。

形式参数就像函数内的其他局部变量，在进入函数时被创建，退出函数时被销毁。

当调用函数时，有两种向函数传递参数的方式：

调用类型 描述

传值调用 该方法把参数的实际值复制给函数的形式参数。在这种情况下，修改函数内的形式参数不会影响实际参数。

引用调用 该方法把参数的地址复制给形式参数。在函数内，该地址用于访问调用中要用到的实际参数。这意味着，修改形式参数会影响实际参数。

指针方式 通过指针传递方式，形参为指向实参地址的指针，当对形参的指向操作时，就相当于对实参本身进行的操作。

默认情况下，C 使用传值调用来传递参数。一般来说，这意味着函数内的代码不能改变用于调用函数的实际参数。

# C作用域规则

任何一种编程中，作用域是程序中定义的变量所存在的区域，超过该区域变量就不能被访问。C 语言中有三个地方可以声明变量：

1.在函数或块内部的局部变量

2.在所有函数外部的全局变量

3.在形式参数的函数参数定义中

局部变量

在某个函数或块的内部声明的变量称为局部变量。它们只能被该函数或该代码块内部的语句使用。局部变量在函数外部是不可知的。下面是使用局部变量的实例。在这里，所有的变量 a、b 和 c 是 main() 函数的局部变量。

#include <stdio.h>

int main ()

{

/\* 局部变量声明 \*/

int a, b;

int c;

/\* 实际初始化 \*/

a = 10;

b = 20;

c = a + b;

printf ("value of a = %d, b = %d and c = %d\n", a, b, c);

return 0;

}

全局变量

全局变量是定义在函数外部，通常是在程序的顶部。全局变量在整个程序生命周期内都是有效的，在任意的函数内部能访问全局变量。

全局变量可以被任何函数访问。也就是说，全局变量在声明后整个程序中都是可用的。下面是使用全局变量和局部变量的实例：

#include <stdio.h>

/\* 全局变量声明 \*/

int g;

int main ()

{

/\* 局部变量声明 \*/

int a, b;

/\* 实际初始化 \*/

a = 10;

b = 20;

g = a + b;

printf ("value of a = %d, b = %d and g = %d\n", a, b, g);

return 0;

}

在程序中，局部变量和全局变量的名称可以相同，但是在函数内，局部变量的值会覆盖全局变量的值。下面是一个实例：

#include <stdio.h>

/\* 全局变量声明 \*/

int g = 20;

int main ()

{

/\* 局部变量声明 \*/

int g = 10;

printf ("value of g = %d\n", g);

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

value of g = 10

初始化局部变量和全局变量

当局部变量被定义时，系统不会对其初始化，您必须自行对其初始化。定义全局变量时，系统会自动对其初始化，如下所示：

数据类型 初始化默认值

int 0

char '\0'

float 0

double 0

pointer NULL

正确地初始化变量是一个良好的编程习惯，否则有时候程序可能会产生意想不到的结果，因为未初始化的变量会导致一些在内存位置中已经可用的垃圾值。

# C数组

C 语言支持数组数据结构，它可以存储一个固定大小的相同类型元素的顺序集合。数组是用来存储一系列数据，但它往往被认为是一系列相同类型的变量。

数组的声明并不是声明一个个单独的变量，比如 number0、number1、...、number99，而是声明一个数组变量，比如 numbers，然后使用 numbers[0]、numbers[1]、...、numbers[99] 来代表一个个单独的变量。数组中的特定元素可以通过索引访问。

所有的数组都是由连续的内存位置组成。最低的地址对应第一个元素，最高的地址对应最后一个元素。



声明数组

在 C 中要声明一个数组，需要指定元素的类型和元素的数量，如下所示：

type arrayName [ arraySize ];

这叫做一维数组。arraySize 必须是一个大于零的整数常量，type 可以是任意有效的 C 数据类型。例如，要声明一个类型为 double 的包含 10 个元素的数组 balance，声明语句如下：

double balance[10];

现在 balance 是一个可用的数组，可以容纳 10 个类型为 double 的数字。

初始化数组

在 C 中，您可以逐个初始化数组，也可以使用一个初始化语句，如下所示：

double balance[5] = {1000.0, 2.0, 3.4, 7.0, 50.0};

大括号 { } 之间的值的数目不能大于我们在数组声明时在方括号 [ ] 中指定的元素数目。

如果您省略掉了数组的大小，数组的大小则为初始化时元素的个数。因此，如果：

double balance[] = {1000.0, 2.0, 3.4, 7.0, 50.0};

您将创建一个数组，它与前一个实例中所创建的数组是完全相同的。下面是一个为数组中某个元素赋值的实例：

balance[4] = 50.0;

上述的语句把数组中第五个元素的值赋为 50.0。所有的数组都是以 0 作为它们第一个元素的索引，也被称为基索引，数组的最后一个索引是数组的总大小减去 1。以下是上面所讨论的数组的的图形表示：



访问数组元素

数组元素可以通过数组名称加索引进行访问。元素的索引是放在方括号内，跟在数组名称的后边。例如：

double salary = balance[9];

上面的语句将把数组中第 10 个元素的值赋给 salary 变量。下面的实例使用了上述的三个概念，即，声明数组、数组赋值、访问数组：

#include <stdio.h>

int main ()

{

int n[ 10 ]; /\* n 是一个包含 10 个整数的数组 \*/

int i,j;

/\* 初始化数组元素 \*/

for ( i = 0; i < 10; i++ )

{

n[ i ] = i + 100; /\* 设置元素 i 为 i + 100 \*/

}

/\* 输出数组中每个元素的值 \*/

for (j = 0; j < 10; j++ )

{

printf("Element[%d] = %d\n", j, n[j] );

}

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

Element[0] = 100

Element[1] = 101

Element[2] = 102

Element[3] = 103

Element[4] = 104

Element[5] = 105

Element[6] = 106

Element[7] = 107

Element[8] = 108

Element[9] = 109

C 中数组详解

在 C 中，数组是非常重要的，我们需要了解更多有关数组的细节。下面列出了 C 程序员必须清楚的一些与数组相关的重要概念：

概念 描述

多维数组 C 支持多维数组。多维数组最简单的形式是二维数组。

传递数组给函数 您可以通过指定不带索引的数组名称来给函数传递一个指向数组的指针。

从函数返回数组 C 允许从函数返回数组。

指向数组的指针 您可以通过指定不带索引的数组名称来生成一个指向数组中第一个元素的指针。

多维数组

C 语言支持多维数组。多维数组声明的一般形式如下：

type name[size1][size2]...[sizeN];

例如，下面的声明创建了一个三维 5 . 10 . 4 整型数组：

int threedim[5][10][4];

二维数组

多维数组最简单的形式是二维数组。一个二维数组，在本质上，是一个一维数组的列表。声明一个 x 行 y 列的二维整型数组，形式如下：

type arrayName [ x ][ y ];

其中，type 可以是任意有效的 C 数据类型，arrayName 是一个有效的 C 标识符。一个二维数组可以被认为是一个带有 x 行和 y 列的表格。下面是一个二维数组，包含 3 行和 4 列：



因此，数组中的每个元素是使用形式为 a[ i , j ] 的元素名称来标识的，其中 a 是数组名称，i 和 j 是唯一标识 a 中每个元素的下标。

初始化二维数组

多维数组可以通过在括号内为每行指定值来进行初始化。下面是一个带有 3 行 4 列的数组。

int a[3][4] = {

{0, 1, 2, 3} , /\* 初始化索引号为 0 的行 \*/

{4, 5, 6, 7} , /\* 初始化索引号为 1 的行 \*/

{8, 9, 10, 11} /\* 初始化索引号为 2 的行 \*/

};

内部嵌套的括号是可选的，下面的初始化与上面是等同的：

int a[3][4] = {0,1,2,3,4,5,6,7,8,9,10,11};

访问二维数组元素

二维数组中的元素是通过使用下标（即数组的行索引和列索引）来访问的。例如：

int val = a[2][3];

上面的语句将获取数组中第 3 行第 4 个元素。您可以通过上面的示意图来进行验证。让我们来看看下面的程序，我们将使用嵌套循环来处理二维数组：

#include <stdio.h>

int main ()

{

/\* 一个带有 5 行 2 列的数组 \*/

int a[5][2] = { {0,0}, {1,2}, {2,4}, {3,6},{4,8}};

int i, j;

/\* 输出数组中每个元素的值 \*/

for ( i = 0; i < 5; i++ )

{

for ( j = 0; j < 2; j++ )

{

printf("a[%d][%d] = %d\n", i,j, a[i][j] );

}

}

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

a[0][0]: 0

a[0][1]: 0

a[1][0]: 1

a[1][1]: 2

a[2][0]: 2

a[2][1]: 4

a[3][0]: 3

a[3][1]: 6

a[4][0]: 4

a[4][1]: 8

如上所述，您可以创建任意维度的数组，但是一般情况下，我们创建的数组是一维数组和二维数组。

# C指针

学习 C 语言的指针既简单又有趣。通过指针，可以简化一些 C 编程任务的执行，还有一些任务，如动态内存分配，没有指针是无法执行的。所以，想要成为一名优秀的 C 程序员，学习指针是很有必要的。

正如您所知道的，每一个变量都有一个内存位置，每一个内存位置都定义了可使用连字号（&）运算符访问的地址，它表示了在内存中的一个地址。请看下面的实例，它将输出定义的变量地址：

#include <stdio.h>

int main ()

{

int var1;

char var2[10];

printf("var1 变量的地址： %x\n", &var1 );

printf("var2 变量的地址： %x\n", &var2 );

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

var1 变量的地址： bff5a400

var2 变量的地址： bff5a3f6

通过上面的实例，我们了解了什么是内存地址以及如何访问它。接下来让我们看看什么是指针。

什么是指针？

指针是一个变量，其值为另一个变量的地址，即，内存位置的直接地址。就像其他变量或常量一样，您必须在使用指针存储其他变量地址之前，对其进行声明。指针变量声明的一般形式为：

type \*var-name;

在这里，type 是指针的基类型，它必须是一个有效的 C 数据类型，var-name 是指针变量的名称。用来声明指针的星号 \* 与乘法中使用的星号是相同的。但是，在这个语句中，星号是用来指定一个变量是指针。以下是有效的指针声明：

int \*ip; /\* 一个整型的指针 \*/

double \*dp; /\* 一个 double 型的指针 \*/

float \*fp; /\* 一个浮点型的指针 \*/

char \*ch /\* 一个字符型的指针 \*/

所有指针的值的实际数据类型，不管是整型、浮点型、字符型，还是其他的数据类型，都是一样的，都是一个代表内存地址的长的十六进制数。不同数据类型的指针之间唯一的不同是，指针所指向的变量或常量的数据类型不同。

如何使用指针？

使用指针时会频繁进行以下几个操作：定义一个指针变量、把变量地址赋值给指针、访问指针变量中可用地址的值。这些是通过使用一元运算符 \* 来返回位于操作数所指定地址的变量的值。下面的实例涉及到了这些操作：

#include <stdio.h>

int main ()

{

int var = 20; /\* 实际变量的声明 \*/

int \*ip; /\* 指针变量的声明 \*/

ip = &var; /\* 在指针变量中存储 var 的地址 \*/

printf("Address of var variable: %x\n", &var );

/\* 在指针变量中存储的地址 \*/

printf("Address stored in ip variable: %x\n", ip );

/\* 使用指针访问值 \*/

printf("Value of \*ip variable: %d\n", \*ip );

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

Address of var variable: bffd8b3c

Address stored in ip variable: bffd8b3c

Value of \*ip variable: 20

C 中的 NULL 指针

在变量声明的时候，如果没有确切的地址可以赋值，为指针变量赋一个 NULL 值是一个良好的编程习惯。赋为 NULL 值的指针被称为空指针。

NULL 指针是一个定义在标准库中的值为零的常量。请看下面的程序：

#include <stdio.h>

int main ()

{

int \*ptr = NULL;

printf("ptr 的值是 %x\n", ptr );

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

ptr 的值是 0

在大多数的操作系统上，程序不允许访问地址为 0 的内存，因为该内存是操作系统保留的。然而，内存地址 0 有特别重要的意义，它表明该指针不指向一个可访问的内存位置。但按照惯例，如果指针包含空值（零值），则假定它不指向任何东西。

如需检查一个空指针，您可以使用 if 语句，如下所示：

if(ptr) /\* 如果 p 非空，则完成 \*/

if(!ptr) /\* 如果 p 为空，则完成 \*/

C 指针详解

在 C 中，有很多指针相关的概念，这些概念都很简单，但是都很重要。下面列出了 C 程序员必须清楚的一些与指针相关的重要概念：

概念 描述

指针的算术运算 可以对指针进行四种算术运算：++、--、+、-

指针数组 可以定义用来存储指针的数组。

指向指针的指针 C 允许指向指针的指针。

传递指针给函数 通过引用或地址传递参数，使传递的参数在调用函数中被改变。

从函数返回指针 C 允许函数返回指针到局部变量、静态变量和动态内存分配。

# C字符串

C 语言中，字符串实际上是使用 null 字符 '\0' 终止的一维字符数组。因此，一个以 null 结尾的字符串，包含了组成字符串的字符。

下面的声明和初始化创建了一个 "Hello" 字符串。由于在数组的末尾存储了空字符，所以字符数组的大小比单词 "Hello" 的字符数多一个。

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

依据数组初始化规则，您可以把上面的语句写成以下语句：

char greeting[] = "Hello";

以下是 C/C++ 中定义的字符串的内存表示：



其实，您不需要把 null 字符放在字符串常量的末尾。C 编译器会在初始化数组时，自动把 '\0' 放在字符串的末尾。让我们尝试输出上面的字符串：

#include <stdio.h>

int main ()

{

char greeting[6] = {'H', 'e', 'l', 'l', 'o', '\0'};

printf("Greeting message: %s\n", greeting );

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

Greeting message: Hello

C 中有大量操作字符串的函数：

序号 函数 & 目的

1 strcpy(s1, s2);

复制字符串 s2 到字符串 s1。

2 strcat(s1, s2);

连接字符串 s2 到字符串 s1 的末尾。

3 strlen(s1);

返回字符串 s1 的长度。

4 strcmp(s1, s2);

如果 s1 和 s2 是相同的，则返回 0；如果 s1<s2 则返回小于 0；如果 s1>s2 则返回大于 0。

5 strchr(s1, ch);

返回一个指针，指向字符串 s1 中字符 ch 的第一次出现的位置。

6 strstr(s1, s2);

返回一个指针，指向字符串 s1 中字符串 s2 的第一次出现的位置。

下面的实例使用了上述的一些函数：

#include <stdio.h>

#include <string.h>

int main ()

{

char str1[12] = "Hello";

char str2[12] = "World";

char str3[12];

int len ;

/\* 复制 str1 到 str3 \*/

strcpy(str3, str1);

printf("strcpy( str3, str1) : %s\n", str3 );

/\* 连接 str1 和 str2 \*/

strcat( str1, str2);

printf("strcat( str1, str2): %s\n", str1 );

/\* 连接后，str1 的总长度 \*/

len = strlen(str1);

printf("strlen(str1) : %d\n", len );

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

strcpy( str3, str1) : Hello

strcat( str1, str2): HelloWorld

strlen(str1) : 10

您可以在 C 标准库中找到更多字符串相关的函数。

# C结构体

C 数组允许定义可存储相同类型数据项的变量，结构是 C 编程中另一种用户自定义的可用的数据类型，它允许您存储不同类型的数据项。

结构用于表示一条记录，假设您想要跟踪图书馆中书本的动态，您可能需要跟踪每本书的下列属性：

Title

Author

Subject

Book ID

定义结构

为了定义结构，您必须使用 struct 语句。struct 语句定义了一个包含多个成员的新的数据类型，struct 语句的格式如下：

struct [structure tag]

{

member definition;

member definition;

...

member definition;

} [one or more structure variables];

structure tag 是可选的，每个 member definition 是标准的变量定义，比如 int i; 或者 float f; 或者其他有效的变量定义。在结构定义的末尾，最后一个分号之前，您可以指定一个或多个结构变量，这是可选的。下面是声明 Book 结构的方式：

struct Books

{

char title[50];

char author[50];

char subject[100];

int book\_id;

} book;

# C共用体

共用体是一种特殊的数据类型，允许您在相同的内存位置存储不同的数据类型。您可以定义一个带有多成员的共用体，但是任何时候只能有一个成员带有值。共用体提供了一种使用相同的内存位置的有效方式。

定义共用体

为了定义共用体，您必须使用 union 语句，方式与定义结构类似。union 语句定义了一个新的数据类型，带有多个成员。union 语句的格式如下：

union [union tag]

{

member definition;

member definition;

...

member definition;

} [one or more union variables];

union tag 是可选的，每个 member definition 是标准的变量定义，比如 int i; 或者 float f; 或者其他有效的变量定义。在共用体定义的末尾，最后一个分号之前，您可以指定一个或多个共用体变量，这是可选的。下面定义一个名为 Data 的共用体类型，有三个成员 i、f 和 str：

union Data

{

int i;

float f;

char str[20];

} data;

现在，Data 类型的变量可以存储一个整数、一个浮点数，或者一个字符串。这意味着一个变量（相同的内存位置）可以存储多个多种类型的数据。您可以根据需要在一个共用体内使用任何内置的或者用户自定义的数据类型。

共用体占用的内存应足够存储共用体中最大的成员。例如，在上面的实例中，Data 将占用 20 个字节的内存空间，因为在各个成员中，字符串所占用的空间是最大的。

# C输入&输出

当我们提到输入时，这意味着要向程序填充一些数据。输入可以是以文件的形式或从命令行中进行。C 语言提供了一系列内置的函数来读取给定的输入，并根据需要填充到程序中。

当我们提到输出时，这意味着要在屏幕上、打印机上或任意文件中显示一些数据。C 语言提供了一系列内置的函数来输出数据到计算机屏幕上和保存数据到文本文件或二进制文件中。

标准文件

C 语言把所有的设备都当作文件。所以设备（比如显示器）被处理的方式与文件相同。以下三个文件会在程序执行时自动打开，以便访问键盘和屏幕。

标准文件 文件指针 设备

标准输入 stdin 键盘

标准输出 stdout 屏幕

标准错误 stderr 您的屏幕

文件指针是访问文件的方式，本节将讲解如何从屏幕读取值以及如何把结果输出到屏幕上。

getchar() & putchar() 函数

int getchar(void) 函数从屏幕读取下一个可用的字符，并把它返回为一个整数。这个函数在同一个时间内只会读取一个单一的字符。您可以在循环内使用这个方法，以便从屏幕上读取多个字符。

int putchar(int c) 函数把字符输出到屏幕上，并返回相同的字符。这个函数在同一个时间内只会输出一个单一的字符。您可以在循环内使用这个方法，以便在屏幕上输出多个字符。

请看下面的实例：

#include <stdio.h>

int main( )

{

int c;

printf( "Enter a value :");

c = getchar( );

printf( "\nYou entered: ");

putchar( c );

return 0;

}

当上面的代码被编译和执行时，它会等待您输入一些文本，当您输入一个文本并按下回车键时，程序会继续并只会读取一个单一的字符，显示如下：

$./a.out

<b>Enter a value :</b> this is test

<b>You entered:</b> t

gets() & puts() 函数

char \*gets(char \*s) 函数从 stdin 读取一行到 s 所指向的缓冲区，直到一个终止符或 EOF。

int puts(const char \*s) 函数把字符串 s 和一个尾随的换行符写入到 stdout。

#include <stdio.h>

int main( )

{

char str[100];

printf( "Enter a value :");

gets( str );

printf( "\nYou entered: ");

puts( str );

return 0;

}

当上面的代码被编译和执行时，它会等待您输入一些文本，当您输入一个文本并按下回车键时，程序会继续并读取一整行直到该行结束，显示如下：

$./a.out

<b>Enter a value :</b> this is test

<b>You entered:</b> This is test

scanf() 和 printf() 函数

int scanf(const char \*format, ...) 函数从标准输入流 stdin 读取输入，并根据提供的 format 来浏览输入。

int printf(const char \*format, ...) 函数把输出写入到标准输出流 stdout ，并根据提供的格式产生输出。

format 可以是一个简单的常量字符串，但是您可以分别指定 %s、%d、%c、%f 等来输出或读取字符串、整数、字符或浮点数。还有许多其他可用的格式选项，可以根据需要使用。如需了解完整的细节，可以查看这些函数的参考手册。

# C文件读写

一个文件，无论它是文本文件还是二进制文件，都是代表了一系列的字节。C 语言不仅提供了访问顶层的函数，也提供了底层（OS）调用来处理存储设备上的文件。本章将讲解文件管理的重要调用。

打开文件

您可以使用 fopen( ) 函数来创建一个新的文件或者打开一个已有的文件，这个调用会初始化类型 FILE 的一个对象，类型 FILE 包含了所有用来控制流的必要的信息。下面是这个函数调用的原型：

FILE \*fopen( const char \* filename, const char \* mode );

在这里，filename 是字符串，用来命名文件，访问模式 mode 的值可以是下列值中的一个：

模式 描述

r 打开一个已有的文本文件，允许读取文件。

w 打开一个文本文件，允许写入文件。如果文件不存在，则会创建一个新文件。在这里，您的程序会从文件的开头写入内容。

a 打开一个文本文件，以追加模式写入文件。如果文件不存在，则会创建一个新文件。在这里，您的程序会在已有的文件内容中追加内容。

r+ 打开一个文本文件，允许读写文件。

w+ 打开一个文本文件，允许读写文件。如果文件已存在，则文件会被截断为零长度，如果文件不存在，则会创建一个新文件。

a+ 打开一个文本文件，允许读写文件。如果文件不存在，则会创建一个新文件。读取会从文件的开头开始，写入则只能是追加模式。

如果处理的是二进制文件，则需使用下面的访问模式来取代上面的访问模式：

"rb", "wb", "ab", "rb+", "r+b", "wb+", "w+b", "ab+", "a+b"

关闭文件

；

为了关闭文件，请使用 fclose( ) 函数。函数的原型如下：

int fclose( FILE \*fp );

如果成功关闭文件，fclose( ) 函数返回零，如果关闭文件时发生错误，函数返回 EOF。这个函数实际上，会清空缓冲区中的数据，关闭文件，并释放用于该文件的所有内存。EOF 是一个定义在头文件 stdio.h 中的常量。

C 标准库提供了各种函数来按字符或者以固定长度字符串的形式读写文件。

写入文件

下面是把字符写入到流中的最简单的函数：

int fputc( int c, FILE \*fp );

函数 fputc() 把参数 c 的字符值写入到 fp 所指向的输出流中。如果写入成功，它会返回写入的字符，如果发生错误，则会返回 EOF。您可以使用下面的函数来把一个以 null 结尾的字符串写入到流中：

int fputs( const char \*s, FILE \*fp );

函数 fputs() 把字符串 s 写入到 fp 所指向的输出流中。如果写入成功，它会返回一个非负值，如果发生错误，则会返回 EOF。您也可以使用 int fprintf(FILE \*fp,const char \*format, ...) 函数来写把一个字符串写入到文件中。尝试下面的实例：

注意：请确保您有可用的 /tmp 目录，如果不存在该目录，则需要在您的计算机上先创建该目录。

#include <stdio.h>

main()

{

FILE \*fp;

fp = fopen("/tmp/test.txt", "w+");

fprintf(fp, "This is testing for fprintf...\n");

fputs("This is testing for fputs...\n", fp);

fclose(fp);

}

读取文件

下面是从文件读取单个字符的最简单的函数：

int fgetc( FILE \* fp );

fgetc() 函数从 fp 所指向的输入文件中读取一个字符。返回值是读取的字符，如果发生错误则返回 EOF。下面的函数允许您从流中读取一个字符串：

char \*fgets( char \*buf, int n, FILE \*fp );

函数 fgets() 从 fp 所指向的输入流中读取 n - 1 个字符。它会把读取的字符串复制到缓冲区 buf，并在最后追加一个 null 字符来终止字符串。

如果这个函数在读取最后一个字符之前就遇到一个换行符 '\n' 或文件的末尾 EOF，则只会返回读取到的字符，包括换行符。您也可以使用 int fscanf(FILE \*fp, const char \*format, ...) 函数来从文件中读取字符串，但是在遇到第一个空格字符时，它会停止读取。

# C预处理器

C 预处理器不是编译器的组成部分，但是它是编译过程中一个单独的步骤。简言之，C 预处理器只不过是一个文本替换工具而已，它们会指示编译器在实际编译之前完成所需的预处理。我们将把 C 预处理器（C Preprocessor）简写为 CPP。

所有的预处理器命令都是以井号（#）开头。它必须是第一个非空字符，为了增强可读性，预处理器指令应从第一列开始。下面列出了所有重要的预处理器指令：

指令 描述

#define 定义宏

#include 包含一个源代码文件

#undef 取消已定义的宏

#ifdef 如果宏已经定义，则返回真

#ifndef 如果宏没有定义，则返回真

#if 如果给定条件为真，则编译下面代码

#else #if 的替代方案

#elif 如果前面的 #if 给定条件不为真，当前条件为真，则编译下面代码

#endif 结束一个 #if……#else 条件编译块

#error 当遇到标准错误时，输出错误消息

#pragma 使用标准化方法，向编译器发布特殊的命令到编译器中

# C头文件

头文件是扩展名为 .h 的文件，包含了 C 函数声明和宏定义，被多个源文件中引用共享。有两种类型的头文件：程序员编写的头文件和编译器自带的头文件。

在程序中要使用头文件，需要使用 C 预处理指令 #include 来引用它。前面我们已经看过 stdio.h 头文件，它是编译器自带的头文件。

引用头文件相当于复制头文件的内容，但是我们不会直接在源文件中复制头文件的内容，因为这么做很容易出错，特别在程序是由多个源文件组成的时候。

A simple practice in C 或 C++ 程序中，建议把所有的常量、宏、系统全局变量和函数原型写在头文件中，在需要的时候随时引用这些头文件。

引用头文件的语法

使用预处理指令 #include 可以引用用户和系统头文件。它的形式有以下两种：

#include <file>

这种形式用于引用系统头文件。它在系统目录的标准列表中搜索名为 file 的文件。在编译源代码时，您可以通过 -I 选项把目录前置在该列表前。

#include "file"

这种形式用于引用用户头文件。它在包含当前文件的目录中搜索名为 file 的文件。在编译源代码时，您可以通过 -I 选项把目录前置在该列表前。

# C递归

递归指的是在函数的定义中使用函数自身的方法。

语法格式如下：

void recursion()

{

recursion(); /\* 函数调用自身 \*/

}

int main()

{

recursion();

}

C 语言支持递归，即一个函数可以调用其自身。但在使用递归时，程序员需要注意定义一个从函数退出的条件，否则会进入死循环。

递归函数在解决许多数学问题上起了至关重要的作用，比如计算一个数的阶乘、生成斐波那契数列，等等。

数的阶乘

下面的实例使用递归函数计算一个给定的数的阶乘：

#include <stdio.h>

double factorial(unsigned int i)

{

if(i <= 1)

{

return 1;

}

return i \* factorial(i - 1);

}

int main()

{

int i = 15;

printf("%d 的阶乘为 %f\n", i, factorial(i));

return 0;

}

当上面的代码被编译和执行时，它会产生下列结果：

15 的阶乘为 1307674368000.000000

# 调用函数

有两种方法：

1 直接将需要代用的\*.c和待编译的\*.c放在相同目录下。

如，将被调用的22.c和待编译的11.c放在相同目录下，

22.c，

#include <stdio.h>

/\* 函数返回两个数中较大的那个数 \*/

int max(int num1, int num2)

{

/\* 局部变量声明 \*/

int result;

if (num1 > num2)

result = num1;

else

result = num2;

return result;

}

11.c，

#include <stdio.h>

#include "22.c"

/\* 函数声明 \*/

int max(int num1, int num2);

int main ()

{

/\* 局部变量定义 \*/

int a = 100;

int b = 200;

int ret;

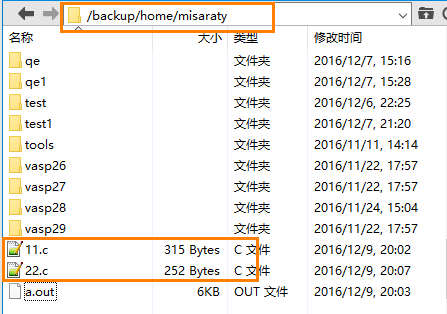
/\* 调用函数来获取最大值 \*/

ret = max(a, b);

printf( "Max value is : %d\n", ret );

return 0;

}





2 11.c和22.c放于不同文件，被调用的22.c放于~/22，11.c放于~，且11.c改（可以不改），

#include <stdio.h>

#include <22.c>



注意：#include <file>，在系统目录的标准列表中搜索名为 file 的文件。#include "file"，在包含当前文件的目录中搜索名为 file 的文件。

# C 语言经典例子

题目：有1、2、3、4个数字，能组成多少个互不相同且无重复数字的三位数？都是多少？

程序分析：可填在百位、十位、个位的数字都是1、2、3、4。组成所有的排列后再去 掉不满足条件的排列。

程序源代码：

#include<stdio.h>

int main()

{

int i,j,k;

printf("\n");

for(i=1;i<5;i++) { // ÒÔÏÂÎªÈýÖØÑ­»·

for(j=1;j<5;j++) {

for (k=1;k<5;k++) { // È·±£i¡¢j¡¢kÈýÎ»»¥²»ÏàÍ¬

if (i!=k&&i!=j&&j!=k) {

printf("%d,%d,%d\n",i,j,k);

}

}

}

}

}

以上实例输出结果为：

1,2,3

1,2,4

1,3,2

1,3,4

1,4,2

1,4,3

2,1,3

2,1,4

2,3,1

2,3,4

2,4,1

2,4,3

3,1,2

3,1,4

3,2,1

3,2,4

3,4,1

3,4,2

4,1,2

4,1,3

4,2,1

4,2,3

4,3,1

4,3,2

题目：企业发放的奖金根据利润提成。利润(I)低于或等于10万元时，奖金可提10%；利润高于10万元，低于20万元时，低于10万元的部分按10%提成，高于10万元的部分，可可提成7.5%；20万到40万之间时，高于20万元的部分，可提成5%；40万到60万之间时高于40万元的部分，可提成3%；60万到100万之间时，高于60万元的部分，可提成1.5%，高于100万元时，超过100万元的部分按1%提成，从键盘输入当月利润I，求应发放奖金总数？

程序分析：请利用数轴来分界，定位。注意定义时需把奖金定义成长整型。

程序源代码：

#include<stdio.h>

int main()

{

double i;

double bonus1,bonus2,bonus4,bonus6,bonus10,bonus;

printf("ÄãµÄ¾»ÀûÈóÊÇ£º\n");

scanf("%lf",&i);

bonus1=100000\*0.1;

bonus2=bonus1+100000\*0.075;

bonus4=bonus2+200000\*0.05;

bonus6=bonus4+200000\*0.03;

bonus10=bonus6+400000\*0.015;

if(i<=100000) {

bonus=i\*0.1;

} else if(i<=200000) {

bonus=bonus1+(i-100000)\*0.075;

} else if(i<=400000) {

bonus=bonus2+(i-200000)\*0.05;

} else if(i<=600000) {

bonus=bonus4+(i-400000)\*0.03;

} else if(i<=1000000) {

bonus=bonus6+(i-600000)\*0.015;

} else if(i>1000000) {

bonus=bonus10+(i-1000000)\*0.01;

}

printf("Ìá³ÉÎª£ºbonus=%lf",bonus);

printf("\n");

}

以上实例输出结果为：

你的净利润是：

120000

提成为：bonus=11500.000000

题目：一个正整数，它加上100后是一个完全平方数，再加上168又是一个完全平方数，请问该数是多少？

程序分析：在10万以内判断，先将该数加上100后再开方，再将该数加上268后再开方，如果开方后的结果满足如下条件，即是结果。

程序源代码：

#include <stdio.h>

#include "math.h"

int main() {

long int i,x,y;

for (i=1;i<100000;i++) {

x=sqrt(i+100); // x为加上100后开方后的结果

y=sqrt(i+268); // y为再加上168后开方后的结果

if(x\*x==i+100 && y\*y==i+268){ //如果一个数的平方根的平方等于该数，这说明此数是完全平方数

printf("\n%ld\n",i);

}

}

}

以上实例输出结果为：

21

261

1581

题目：输入某年某月某日，判断这一天是这一年的第几天？

程序分析：以3月5日为例，应该先把前两个月的加起来，然后再加上5天即本年的第几天，特殊情况，闰年且输入月份大于3时需考虑多加一天。

程序源代码：

#include <stdio.h>

int main()

{

int day,month,year,sum,leap;

printf("\n请输入年、月、日，格式为：年,月,日（2015,12,10）\n");

scanf("%d,%d,%d",&year,&month,&day); // 格式为：2015,12,10

switch(month) // 先计算某月以前月份的总天数

{

case 1:sum=0;break;

case 2:sum=31;break;

case 3:sum=59;break;

case 4:sum=90;break;

case 5:sum=120;break;

case 6:sum=151;break;

case 7:sum=181;break;

case 8:sum=212;break;

case 9:sum=243;break;

case 10:sum=273;break;

case 11:sum=304;break;

case 12:sum=334;break;

default:printf("data error");break;

}

sum=sum+day; // 再加上某天的天数

if(year%400==0||(year%4==0&&year%100!=0)) {// 判断是不是闰年

leap=1;

} else {

leap=0;

}

if(leap==1&&month>2) { // \*如果是闰年且月份大于2,总天数应该加一天

sum++;

}

printf("这是这一年的第 %d 天。",sum);

printf("\n");

}

以上实例输出结果为：

请输入年、月、日，格式为：年,月,日（2015,12,10）

2015,10,1

这是这一年的第 274 天。

题目：输入三个整数x,y,z，请把这三个数由小到大输出。

程序分析：我们想办法把最小的数放到x上，先将x与y进行比较，如果x>y则将x与y的值进行交换，然后再用x与z进行比较，如果x>z则将x与z的值进行交换，这样能使x最小。

程序源代码：

// Created by www.runoob.com on 15/11/9.

// Copyright © 2015年 菜鸟教程. All rights reserved.

//

#include <stdio.h>

int main()

{

int x,y,z,t;

printf("\n请输入三个数字:\n");

scanf("%d%d%d",&x,&y,&z);

if (x>y) { /\*交换x,y的值\*/

t=x;x=y;y=t;

}

if(x>z) { /\*交换x,z的值\*/

t=z;z=x;x=t;

}

if(y>z) { /\*交换z,y的值\*/

t=y;y=z;z=t;

}

printf("从小到大排序: %d %d %d\n",x,y,z);

}

以上实例输出结果为：

请输入三个数字:

1

3

2

从小到大排序: 1 2 3

题目：输出9\*9口诀。

程序分析：分行与列考虑，共9行9列，i控制行，j控制列。

程序源代码：

// Created by www.runoob.com on 15/11/9.

// Copyright © 2015年 菜鸟教程. All rights reserved.

//

#include<stdio.h>

int main()

{

int i,j,result;

printf("\n");

for (i=1;i<10;i++)

{

for(j=1;j<10;j++)

{

result=i\*j;

printf("%d\*%d=%-3d",i,j,result); /\*-3d表示左对齐，占3位\*/

}

printf("\n"); /\*每一行后换行\*/

}

}

以上实例输出结果为：

1\*1=1 1\*2=2 1\*3=3 1\*4=4 1\*5=5 1\*6=6 1\*7=7 1\*8=8 1\*9=9

2\*1=2 2\*2=4 2\*3=6 2\*4=8 2\*5=10 2\*6=12 2\*7=14 2\*8=16 2\*9=18

3\*1=3 3\*2=6 3\*3=9 3\*4=12 3\*5=15 3\*6=18 3\*7=21 3\*8=24 3\*9=27

4\*1=4 4\*2=8 4\*3=12 4\*4=16 4\*5=20 4\*6=24 4\*7=28 4\*8=32 4\*9=36

5\*1=5 5\*2=10 5\*3=15 5\*4=20 5\*5=25 5\*6=30 5\*7=35 5\*8=40 5\*9=45

6\*1=6 6\*2=12 6\*3=18 6\*4=24 6\*5=30 6\*6=36 6\*7=42 6\*8=48 6\*9=54

7\*1=7 7\*2=14 7\*3=21 7\*4=28 7\*5=35 7\*6=42 7\*7=49 7\*8=56 7\*9=63

8\*1=8 8\*2=16 8\*3=24 8\*4=32 8\*5=40 8\*6=48 8\*7=56 8\*8=64 8\*9=72

9\*1=9 9\*2=18 9\*3=27 9\*4=36 9\*5=45 9\*6=54 9\*7=63 9\*8=72 9\*9=81

# 参考

* 《C Primer Plus》
* [C语言教程](http://www.runoob.com/cprogramming/c-tutorial.html)
* 《C程序设计案例与提高教程 第二版》 清华大学出版社 王金鹏等 2015