**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()** 。

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：

返回类型 函数名称(参数列表)

{

函数主体;

}

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：

返回类型 函数名称(参数列表) ;

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：

**实例**

#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：

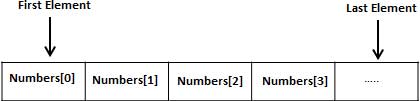
Max value is : 200

# 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：

数据类型 数组名称 [ 数组大小 ];

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。

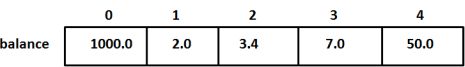
如果您省略掉了数组的大小，数组的大小则为初始化时元素的个数。因此，如果：

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：

## 实例

#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 enum(枚举)

枚举是 C 语言中的一种基本数据类型，它可以让数据更简洁，更易读。

枚举语法定义格式为：

enum　枚举名　{枚举元素1,枚举元素2,……};

接下来我们举个例子，比如：一星期有 7 天，如果不用枚举，我们需要使用 #define 来为每个整数定义一个别名：

#define MON 1 #define TUE 2 #define WED 3 #define THU 4 #define FRI 5 #define SAT 6 #define SUN 7

这个看起来代码量就比较多，接下来我们看看使用枚举的方式：

enum DAY

{

MON=1, TUE, WED, THU, FRI, SAT, SUN

};

这样看起来是不是更简洁了。

**注意：**第一个枚举成员的默认值为整型的 0，后续枚举成员的值在前一个成员上加 1。我们在这个实例中把第一个枚举成员的值定义为 1，第二个就为 2，以此类推。

*可以在定义枚举类型时改变枚举元素的值：*

enum season {spring, summer=3, autumn, winter};

*没有指定值的枚举元素，其值为前一元素加 1。也就说 spring 的值为 0，summer 的值为 3，autumn 的值为 4，winter 的值为 5*

### 枚举变量的定义

前面我们只是声明了枚举类型，接下来我们看看如何定义枚举变量。

我们可以通过以下三种方式来定义枚举变量

**1、先定义枚举类型，再定义枚举变量**

enum DAY

{

MON=1, TUE, WED, THU, FRI, SAT, SUN

};

enum DAY day;

**2、定义枚举类型的同时定义枚举变量**

enum DAY

{

MON=1, TUE, WED, THU, FRI, SAT, SUN

} day;

**3、省略枚举名称，直接定义枚举变量**

enum

{

MON=1, TUE, WED, THU, FRI, SAT, SUN

} day;

## 实例

#include<stdio.h>

enum DAY

{

MON=1,

TUE,

WED,

THU,

FRI,

SAT,

SUN

};

int main()

{

enum DAY day;

day = WED;

printf("%d",day);

return 0;

}

**C 结构体**

C 数组允许定义可存储相同类型数据项的变量，**结构体**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。

结构体用于表示一条记录，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 tag

{

member-list;

member-list;

member-list

} variable-list ;

**tag** 是结构体标签。

**member-list** 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。

**variable-list** 结构变量，定义在结构的末尾，最后一个分号之前，您可以指定一个或多个结构变量。

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;

在一般情况下，**tag、member-list、variable-list** 这 3 部分至少要出现 2 个。

以下为实例：

//此声明声明了拥有3个成员的结构体，分别为整型的a，字符型的b和双精度的c //同时又声明了结构体变量s1 //这个结构体并没有标明其标签

struct

{

int a;

char b;

double c;

} s1;

//此声明声明了拥有3个成员的结构体，分别为整型的a，字符型的b和双精度的c //结构体的标签被命名为SIMPLE,没有声明变量

struct SIMPLE

{

int a;

char b;

double c;

};

//用SIMPLE标签的结构体，另外声明了变量t1、t2、t3

struct SIMPLE t1, t2[20], \*t3;

//也可以用typedef创建新类型

typedef struct

{

int a;

char b;

double c;

} Simple2;

//现在可以用Simple2作为类型声明新的结构体变量

Simple2 u1, u2[20], \*u3;

在上面的声明中，第一个和第二声明被编译器当作两个完全不同的类型，即使他们的成员列表是一样的，如果令 t3=&s1，则是非法的。

结构体的成员可以包含其他结构体，也可以包含指向自己结构体类型的指针，而通常这种指针的应用是为了实现一些更高级的数据结构如链表和树等。

**结构体变量的初始化**

和其它类型变量一样，对结构体变量可以在定义时指定初始值。

**实例**

#include <stdio.h>

struct Books

{

char title[50];

char author[50];

char subject[100];

int book\_id;

} book = {"C 语言", "RUNOOB", "编程语言", 123456};

int main()

{

printf("title : %s\nauthor: %s\nsubject: %s\nbook\_id: %d\n", book.title, book.author, book.subject, book.book\_id);

}

**访问结构成员**

为了访问结构的成员，我们使用**成员访问运算符（.）**。

成员访问运算符是结构变量名称和我们要访问的结构成员之间的一个句号。

您可以使用 **struct** 关键字来定义结构类型的变量。下面的实例演示了结构的用法：

**实例**

#include <stdio.h>

#include <string.h>

struct Books

{

char title[50];

char author[50];

char subject[100];

int book\_id;

};

int main( )

{

struct Books Book1; /\* 声明 Book1，类型为 Books \*/

struct Books Book2; /\* 声明 Book2，类型为 Books \*/

/\* Book1 详述 \*/

strcpy( Book1.title, "C Programming");

strcpy( Book1.author, "Nuha Ali");

strcpy( Book1.subject, "C Programming Tutorial");

Book1.book\_id = 6495407;

/\* Book2 详述 \*/

strcpy( Book2.title, "Telecom Billing");

strcpy( Book2.author, "Zara Ali");

strcpy( Book2.subject, "Telecom Billing Tutorial");

Book2.book\_id = 6495700;

/\* 输出 Book1 信息 \*/

printf( "Book 1 title : %s\n", Book1.title);

printf( "Book 1 author : %s\n", Book1.author);

printf( "Book 1 subject : %s\n", Book1.subject);

printf( "Book 1 book\_id : %d\n", Book1.book\_id);

/\* 输出 Book2 信息 \*/ printf( "Book 2 title : %s\n", Book2.title);

printf( "Book 2 author : %s\n", Book2.author);

printf( "Book 2 subject : %s\n", Book2.subject);

printf( "Book 2 book\_id : %d\n", Book2.book\_id); return 0; }

# C typedef

C 语言提供了 **typedef** 关键字，您可以使用它来为类型取一个新的名字。下面的实例为单字节数字定义了一个术语 **u8**：

typedef unsigned char u8;

在这个类型定义之后，标识符 u8 可作为类型 **unsigned char** 的缩写，例如：

u8 b1, b2;

# C 预处理器

**C 预处理器** 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。

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 |

## 预处理器实例

分析下面的实例来理解不同的指令。

#define MAX\_ARRAY\_LENGTH 20

这个指令告诉 CPP 把所有的 MAX\_ARRAY\_LENGTH 替换为 20。使用 *#define* 定义常量来增强可读性。

#include <stdio.h>

#include "myheader.h"

这些指令告诉 CPP 从**系统库**中获取 stdio.h，并添加文本到当前的源文件中。下一行告诉 CPP 从本地目录中获取 **myheader.h**，并添加内容到当前的源文件中。

#undef FILE\_SIZE

#define FILE\_SIZE 42

这个指令告诉 CPP 取消已定义的 FILE\_SIZE，并定义它为 42。

#ifndef MESSAGE

#define MESSAGE "You wish!"

#endif

## 参数化的宏

CPP 一个强大的功能是可以使用参数化的宏来模拟函数。例如，下面的代码是计算一个数的平方：

int square(int x) {

return x \* x;

}

我们可以使用宏重写上面的代码，如下：

#define square(x) ((x) \* (x))

# 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。

引用头文件相当于复制头文件的内容。

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。

#include <stdio.h> //系统自带的头文件，保存在系统文件夹中

#include “my.h” //用户编写的头文件，保存在用户文件夹中

## 只引用一次头文件

如果一个头文件被引用两次，编译器会处理两次头文件的内容，这将产生错误。为了防止这种情况，标准的做法是把文件的整个内容放在条件编译语句中，如下：

#ifndef \_MY

#define \_MY

头文件内容

#endif

这种结构就是通常所说的包装器 **#ifndef**。当再次引用头文件时，条件为假，因为 HEADER\_FILE 已定义。此时，预处理器会跳过文件的整个内容，编译器会忽略它。