

第6章 函数

《C语言程序设计新编教程》

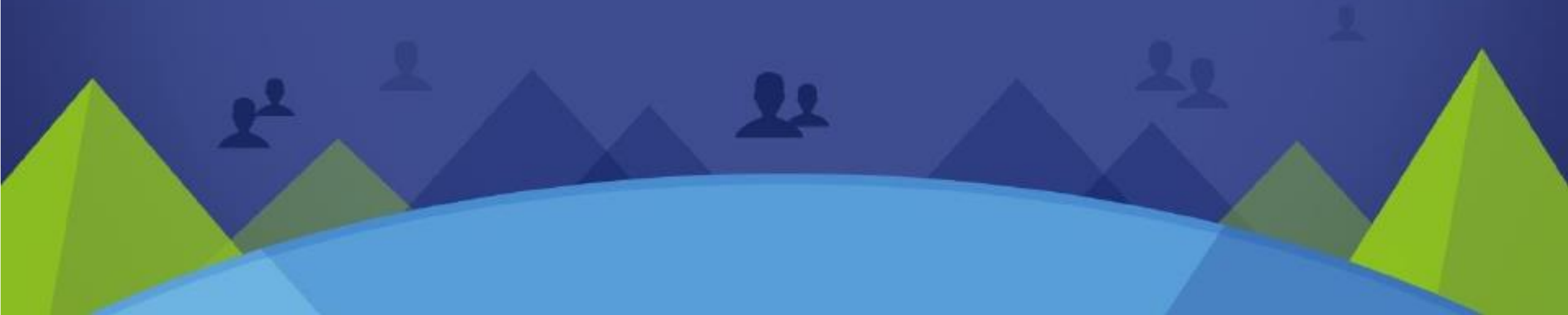


表 7-1 标 准 函 数

函 数 类 别	头 文 件	说 明	举 例
输入/输出函数	stdio.h	用于输入或输出	printf()、fprintf()
数学函数	math.h	用于数学函数计算	abs()、sqrt()、sin()
字符函数	ctype.h	用于对字符进行分类和判断	isdigit()、isupper()
字符串函数	string.h	用于字符串操作和处理	strcpy()、strlen()
转换函数	stdlib.h/ ctype.h	用于字符量和数字量、大小写转换	atoi()、tolower()
日期/时间函数	time.h	用于日期/时间转换操作	time()、ctime()
内存管理函数	stdlib.h	用于动态存储分配	malloc()、free()

使用方法：在源文件开头加入对应的头文件编译预处理命令，在程序中直接调用

(1) 数值计算函数

数值计算函数在头文件 `math.h` 中。

表 B-4 数值计算函数

函 数 名	函 数 原 型	功 能
abs	int abs(int x);	求整数 x 的绝对值
fabs	double fabs(double x);	求 x 的绝对值
exp	double exp(double x);	计算 e^x 的值
pow	double pow(double x,double y);	计算 x^y 的值
sqrt	double sqrt(double x);	计算 x 的平方根, $x \geq 0$
fmod	double fmod(double x,double y);	求 x/y 的余数 (双精度表示)
rand	int rand(void)	生成 0~32767 间的随机整数
floor	double floor(double x);	求不大于 x 的最大整数 (双精度表示)
frexp	double frexp(double val,int*eptr);	把 val 分解尾数 x 和以 2 为底的指数 n , 即 $val=x \times 2^n$, n 存放在 *eptr 中, 返回尾数 x , $0.5 \leq x < 1$
modf	double modf(double val,double *iptr);	把 val 分解成整数和小数部分, 整数部分存放在 *iptr 中, 返回小数部分
log	double log(double x);	求 $\log_e x$, 即 $\ln x$
log10	double log10(double x);	求 $\log_{10} x$, 即 $\lg x$
sin	double sin(double x);	计算 $\sin(x)$ 的值, x 为弧度
cos	double cos(double x);	计算 $\cos(x)$ 的值, x 为弧度
tan	double tan(double x);	计算 $\tan(x)$ 的值, x 为弧度
asin	double asin (double x);	计算 $\arcsin(x)$ 的值, x 在 $-1 \sim 1$ 之间
acos	double acos (double x);	计算 $\arccos(x)$ 的值, x 在 $-1 \sim 1$ 之间
atan	double atan (double x);	计算 $\arctan(x)$ 的值

例子:

```
#include<stdio.h>
#include<math.h>
int main()
{
    int x,y;
    x=10;
    y=log(x);
    printf("%d",y);
}
```

(2) 字符函数

字符函数在头文件 `ctype.h` 中。除了 `tolower` 和 `toupper` 函数外，其他函数如果判断为真，则返回 1；否则返回 0。

表 B-5 字 符 函 数

函 数 名	函 数 原 型	功 能
<code>isalpha</code>	<code>int isalpha(int ch);</code>	判断 <code>ch</code> 是否为字母
<code>isdigit</code>	<code>int isdigit(int ch);</code>	判断 <code>ch</code> 是否为数字
<code>isalnum</code>	<code>int isalnum(int ch);</code>	判断 <code>ch</code> 是否为字母或数字
<code>isctrl</code>	<code>int isctrl(int ch);</code>	判断 <code>ch</code> 是否为控制字符
<code>isspace</code>	<code>int isspace(int ch);</code>	判断 <code>ch</code> 是否为空格、制表或换行字符
<code>islower</code>	<code>int islower(int ch);</code>	判断 <code>ch</code> 是否为小写字母
<code>isupper</code>	<code>int isupper(int ch);</code>	判断 <code>ch</code> 是否为大写字母
<code>isgraph</code>	<code>int isgraph(int ch);</code>	判断 <code>ch</code> 是否为可打印字符（ASCII 值 0x21~0x7e 之间）
<code>isprint</code>	<code>int isprint(int ch);</code>	判断 <code>ch</code> 是否为可打印字符（ASCII 值 0x20~0x7e 之间）
<code>ispunct</code>	<code>int ispunct(int ch);</code>	判断 <code>ch</code> 是否为标点字符（包括空格），即除字母、数字和空格以外的所有可打印字符
<code>tolower</code>	<code>int tolower(int ch);</code>	把 <code>ch</code> 中的字母转换成小写字母，返回小写字母
<code>toupper</code>	<code>int toupper(int ch);</code>	把 <code>ch</code> 中的字母转换成大写字母，返回大写字母

(3) 字符串函数

字符串函数在头文件 `string.h` 中。

表 B-6 字符串函数

函 数 名	函 数 原 型	功 能
<code>strlen</code>	<code>unsigned strlen(char *s);</code>	求字符串 <code>s</code> 长度, 不计最后的 <code>'\0'</code>
<code>strcat</code>	<code>char *strcat(char *s1, char *s2);</code>	把字符串 <code>s2</code> 连接到 <code>s1</code> 后面, 返回 <code>s1</code> 地址
<code>strcpy</code>	<code>char *strcpy(char *s1, char *s2);</code>	把字符串 <code>s2</code> 复制到 <code>s1</code> 中, 返回 <code>s1</code> 地址
<code>strncpy</code>	<code>char *strncpy(char *dest, char *src, unsigned n);</code>	复制字符串 <code>s2</code> 的前 <code>n</code> 个字符
<code>strcmp</code>	<code>char *strcmp(char *s1, char *s2);</code>	对 <code>s1</code> 和 <code>s2</code> 所指字符串进行比较。 <code>s1 < s2</code> , 则返回 -1, <code>s1 = s2</code> , 则返回 0, <code>s1 > s2</code> , 则返回 1
<code>strcasecmp</code>	<code>int strcasecmp(char *s1, char *s2);</code>	字符串进行比较时, 自动忽略大小写的差异
<code>strstr</code>	<code>char *strstr(char *s1, char *s2);</code>	在字符串 <code>s1</code> 中, 查找 <code>s2</code> 第一次出现的位置, 返回找到的地址, 找不到返回 <code>NULL</code>
<code>strchr</code>	<code>char *strchr(char *s, int ch);</code>	在字符串 <code>s</code> 中, 查找字符 <code>ch</code> 第一次出现的位置, 返回找到的地址, 找不到返回 <code>NULL</code>



能力要求

CAPACITY



掌握函数的定义及调用



掌握函数的嵌套调用及递归调用



掌握数组作为函数参数的使用



掌握变量的存储方式



内容导航

CONTENTS

函数



函数的分类



局部变量和全局变量



函数定义的一般形式



变量的存储类别



函数的参数和函数的值



函数的调用



函数的嵌套调用



函数的递归调用

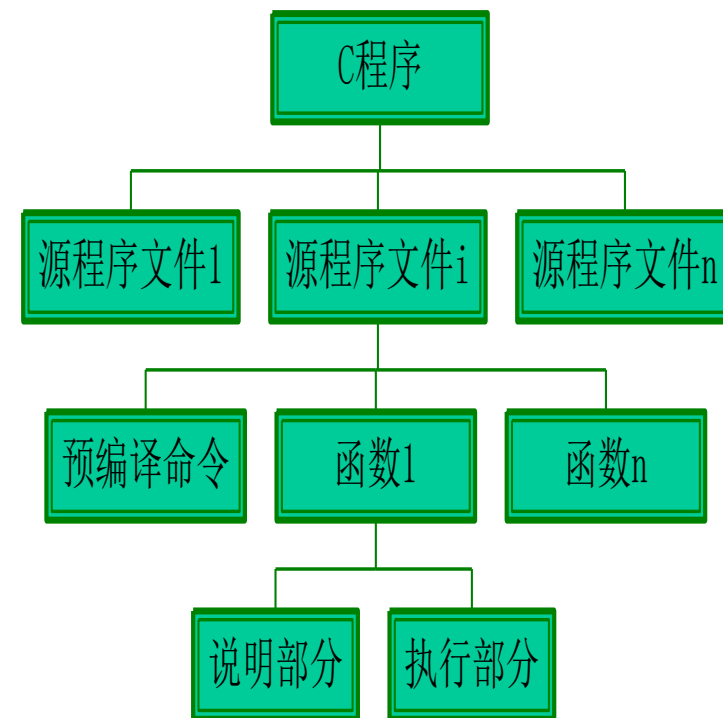


数组作为函数参数

📖 C是函数式语言（哪里都是函数）




📖 必须有且只能有一个名为main的主函数从main函数开始，在main中结束

📖 主函数再调用其他函数



C程序
结构



-  1.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。
-  2.从是否返回值的角度看，又可把函数分为有返回值函数和无返回值函数两种。
-  3.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。



库函数:

函 数 类 别	头 文 件
输入/输出函数	stdio.h
数学函数	math.h
字符函数	ctype.h
字符串函数	string.h
转换函数	stdlib.h/ ctype.h
日期/时间函数	time.h
内存管理函数	stdlib.h

自定义函数:

lize()

有返回值的函数:

数值计算函数, 返回
计算结果:

sin()
tan()等

无返回值的函数:

1. printf 函数: 这是 C 语言中最常用的函数之一, 它用于将文本输出到控制台。它没有返回值, 因为它只是将文本输出到屏幕上, 而不需要返回任何值。
2. scanf 函数: 这个函数用于从控制台读取输入。它也没有返回值, 因为它只是将输入读取到变量中, 而不需要返回任何值。
3. getchar 函数: 这个函数用于从控制台读取单个字符。它没有返回值, 因为它只是将字符读取到变量中, 而不需要返回任何值。
4. putchar 函数: 这个函数用于将单个字符输出到控制台。它没有返回值, 因为它只是将字符输出到屏幕上, 而不需要返回任何值。

有参函数:

数值计算函数, 返回
计算结果:

sin()
tan()等

无参函数:

在C语言中, 下列函数不需要参数的是 ()。

A.getchar()

B.putchar()

C.printf()

D.scan()

```
#include<stdio.h>
int main()
{
    char a;
    getchar();
    printf("%c",a);
}
```



内容导航

CONTENTS

函数



函数的分类



函数定义的一般形式



函数的参数和函数的值



函数的调用



函数的嵌套调用



函数的递归调用



数组作为函数参数



局部变量和全局变量



变量的存储类别

函数在使用前，是需要定义的！！！！

库函数需要使用者定义吗？在哪里定义？

能否自己定义一个函数？

能否自己写一个函数的头文件？



C语言中的库函数定义在什么地方啊？

比如说我想知道"string.h"中的strcmp()函数具体是怎么实现的，那我应该到哪儿找啊？？

头文件中只有函数的声明而没有函数的定义。这个头文件是给程序员看的，好让你知道有这些函数。

那真正的函数定义在哪呢？

库文件中的函数早就已经编译好了存在.obj或者是.lib的文件里面。这种是静态的连接模式。在你的程序便已完成以后连接器再将你的程序的.obj文件和库文件进行连接最后生成.exe文件。

其实大部分的库文件都是能够自己编写的。像strcmp()这种微软的库文件是更不可能放出原码的



能否自己定义一个函数？

能



能否自己写一个函数的头文件？

能



• 1.无参函数的定义

- 类型名 函数名()
- {声明部分
- 语句
- }

定义函数时要用类型名指定函数值得类型，其中类型名和函数名称为函数头。该类型名与前面介绍的各种数据类型相同。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。

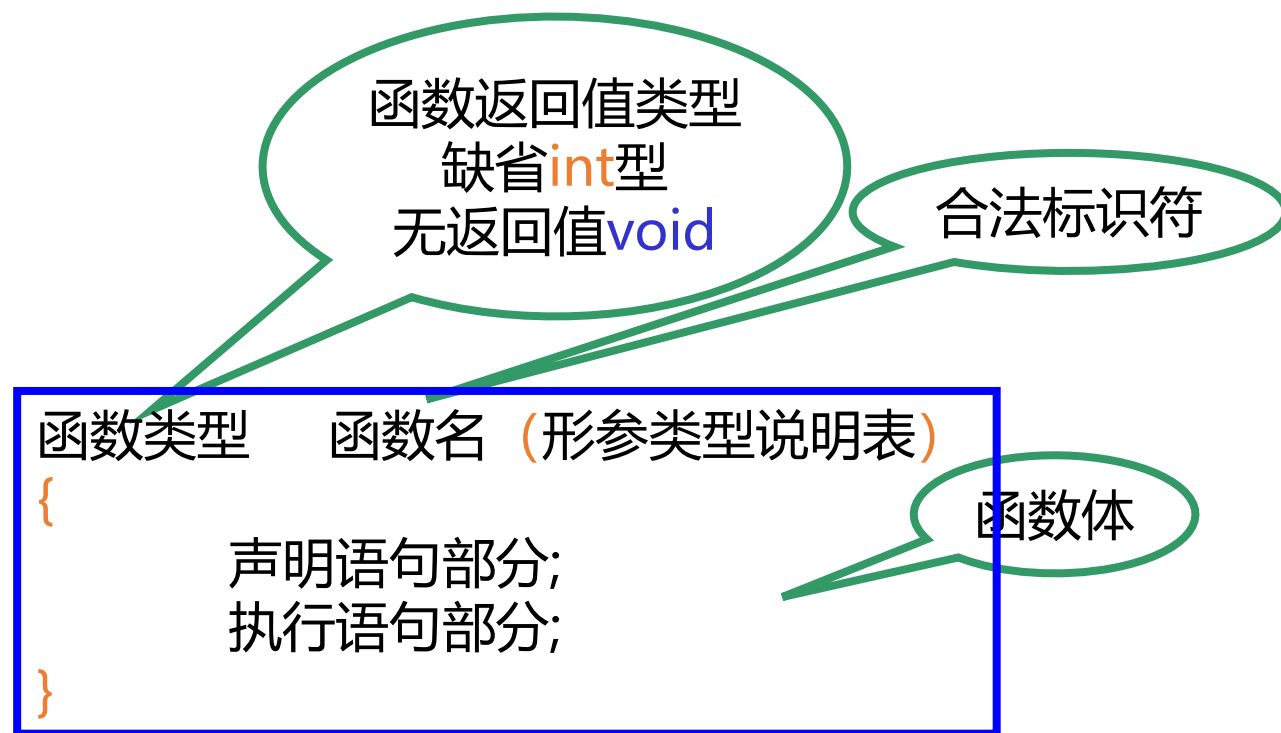
• 2.有参函数的定义

- 类型名 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。



- 任何函数（包括主函数main()）都是由函数说明和函数体两部分组成。
- 一般格式



有参函数比无参函数多了一个参数表。调用有参函数时，调用函数将赋予这些参数实际的值。

为了与调用函数提供的实际参数区别开，将函数定义中的参数表称为形式参数表，简称形参表。

例 有参函数（现代风格）

```
int max(int x,int y)
{   int z;
    z=x>y?x:y;
    return(z);
}
```

例 有参函数（现代风格）

```
int max(int x, y)
{   int z;
    z=x>y?x:y;
    return(z);
}
```



[案例6.1] 定义一个函数，用于求两个数中的大数。

```
#include <stdio.h>
int max(int x,int y)
{
    int result;
    result =x>y?x:y;
    return(result);
}
int main()
{
    int a,b,c;
    printf("Input two integers:");
    scanf("%d %d",&a,&b);
    c=max(a,b);
    printf("Max is:%d.\n",c);
    return 0;
}
```



- 2. 说明

- (1) **函数定义不允许嵌套。**

- 在C语言中，**所有函数（包括主函数main()）都是平行的。**

- 但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。

- (2) 空函数——既无参数、函数体又为空的函数。其一般形式为：

- [函数类型] 函数名(void)

- {}



- (3) 在老版本C语言中，参数类型说明允许放在函数说明部分的第2行单独指定。



内容导航

CONTENTS

函数

- 函数的分类
 - 函数定义的一般形式
 - 函数的参数和函数的值
 - 函数的调用
 - 函数的嵌套调用
 - 函数的递归调用
 - 数组作为函数参数
- 局部变量和全局变量
 - 变量的存储类别

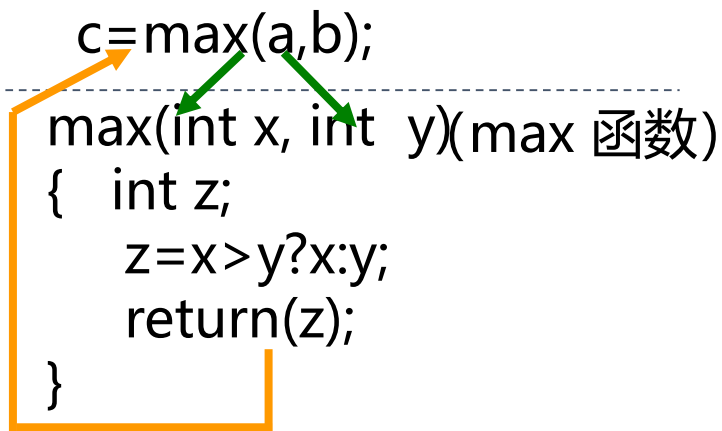
• 形参与实参

- 形式参数：定义函数时函数名后面括号中的变量名
- 实际参数：调用函数时函数名后面括号中的表达式

例 比较两个数并输出大者

```

c=max(a,b);
max(int x, int y)(max 函数)
{
    int z;
    z=x>y?x:y;
    return(z);
}
    
```



```

main()
{
    int a,b,c;
    scanf("%d,%d",&a,&b);
    c=max(a,b);
    printf("Max is %d",c);
}
    
```

实参

```

max(int x, int y)
{
    int z;
    z=x>y?x:y;
    return(z);
}
    
```

形参



函数的**形参**和**实参**具有以下特点：

- 1、**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。
- 2、**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，输入等方法使实参获得确定值。
- 3、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。
- 4、**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。



```
int s(int n)
{ int i;
  printf("n_x=%d\n",n);      /*输出改变前形参的值*/
  for(i=n-1; i>=1; i--) n=n+i; /*改变形参的值*/
  printf("n_x=%d\n",n);      /*输出改变后形参的值*/
}
```

```
main()
{ void s(int n);             /*说明函数*/
  int n=100;                 /*定义实参n, 并初始化*/
  s(n);                     /*调用函数*/
  printf("n_s=%d\n",n);      /*输出调用后实参的值, 便于进行比较*/
  getch();
}
```



- 说明:
- (1) 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, 使实参获得确定的值。
- (2) 形参变量只有在被调用时, 才分配内存单元; 调用结束时, 即刻释放所分配的内存单元。
因此, 形参只有在该函数内有效。调用结束, 返回调用函数后, 则不能再使用该形参变量。
- (3) 实参对形参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。
- (4) 实参和形参占用不同的内存单元, 即使同名也互不影响。
- (5) 形参必须指定类型
- (6) 形参与实参类型一致, 个数相同
- (7) 若形参与实参类型不一致, 自动按形参类型转换——函数调用转换

- 函数的返回值与函数类型

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，又可将函数分为有返回值函数和无返回值函数两种。

1. 函数返回值与return语句

有参函数的返回值，是通过函数中的return语句来获得的。

形式: `return(表达式);`
 或 `return 表达式;`
 或 `return;`

功能: 使程序控制从被调用函数返回到调用函数中，
 同时把返回值带给调用函数



- 说明:

- 函数中可有多个return语句
- 若无return语句，遇}时，自动返回调用函数
- 若函数类型与return语句中表达式值的类型不一致，按前者为准，自动转换-----函数调用转换
- void型函数

注意: 调用函数中无return语句，并不是不返回一个值，而是一个不确定的值。
为了明确表示不返回值，可以用“void”定义成“无（空）类型”。

```
例 无返回值函数
void swap(int x,int y )
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
```



2. 函数类型

在定义函数时，对函数类型的说明，应与return语句中、返回值表达式的类型一致。

如果不一致，则以函数类型为准。如果缺省函数类型，则系统一律按整型处理。



- 1、通过自定义函数的方法，定义一个函数实现“输入数值的5次方计算”功能，并在程序中正确调用。
- 2、通过自定义函数的方法，定义一个函数实现“输入数值的n次方计算”功能，并在程序中正确调用。
- 3、通过自定义函数的方法，定义一个函数实现“两个数数值交换”功能，并在程序中正确调用。
- 4、编写函数将给定的一个二维数组（4×4）转置，即行列互换
- 5、通过自定义函数的方法，定义一个函数实现“输入数值阶乘求和”功能，并在程序中正确调用。

即

$$1! + 2! + 3! + \dots + N!$$

1、通过自定义函数的方法，定义一个函数实现“输入数值的5次方计算”功能，并在程序中正确调用。

```
#include<stdio.h>
int five(int a)
{
    int y;
    y=a*a*a*a*a;
    return (y);
}

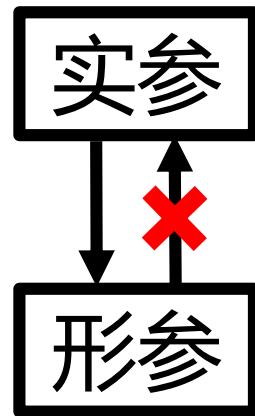
int main()
{
    int x,y;
    scanf("%d",&x);
    printf("x=%d\n",x);
    y=five(x);
    printf("x的五次方为%d",y);
}
```


2、通过自定义函数的方法，定义一个函数实现“两个数数值交换”功能，并在程序中正确调用。

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。



2、通过自定义函数的方法，定义一个函数实现“两个数数值交换”功能，并在程序中正确调用。

```
#include<stdio.h>
int swap(int a,int b)
{
    int temp;
    temp=a;
    a=b;
    b=temp;
}
int main()
{
    int x,y;
    scanf("%d%d",&x,&y);
    printf("交换前\n");
    printf("x=%d,y=%d\n",x,y);
    swap(x,y);
    printf("交换后\n");
    printf("x=%d,y=%d\n",x,y);
}
```

这个代码能实现你想设计的功能吗？



内容导航

CONTENTS

函数

- 函数的分类
 - 函数定义的一般形式
 - 函数的参数和函数的值
 - 函数的调用
 - 函数的嵌套调用
 - 函数的递归调用
 - 数组作为函数参数
- 局部变量和全局变量
 - 变量的存储类别

- 对被调用函数要求:
 - 必须是已存在的函数
 - 库函数: `#include <*.h>`
 - 用户自定义函数: 函数定义



- 在程序中，是通过对函数的调用来执行函数体的，其过程与其它语言的子程序调用相似。
- C语言中，函数调用的一般形式为：**函数名([实际参数表])**
- **切记：实参的个数、类型和顺序，应该与被调用函数所要求的参数个数、类型和顺序一致，才能正确地进行数据传递。**
- 在C语言中，可以用以下几种方式调用函数：
 - (1) **函数表达式**。函数作为表达式的一项，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。`z=max(x,y)`
 - (2) **函数语句**。C语言中的函数可以只进行某些操作而不返回函数值，这时的函数调用可作为一条独立的语句。`printf("%d" ,a)`
 - (3) **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。`printf("%d" , z=max(x,y))`

函数的调用方式

- 函数语句:

例 `printstar();`
`printf("Hello,World!\n");`

- 函数表达式:

例 `m=max(a,b)*2;`

- 函数参数:

例 `printf("%d" ,max(a,b));`
`m=max(a,max(b,c));`



说明:

- (1) 调用函数时，函数名称必须与具有该功能的自定义函数名称完全一致。
- (2) **实参在类型上按顺序与形参，必须一一对应和匹配。**如果类型不匹配，C编译程序将按赋值兼容的规则进行转换。如果实参和形参的类型不赋值兼容，通常并不给出出错信息，且程序仍然继续执行，只是得不到正确的结果。
- (3) 如果实参表中包括多个参数，对实参的求值顺序随系统而异。





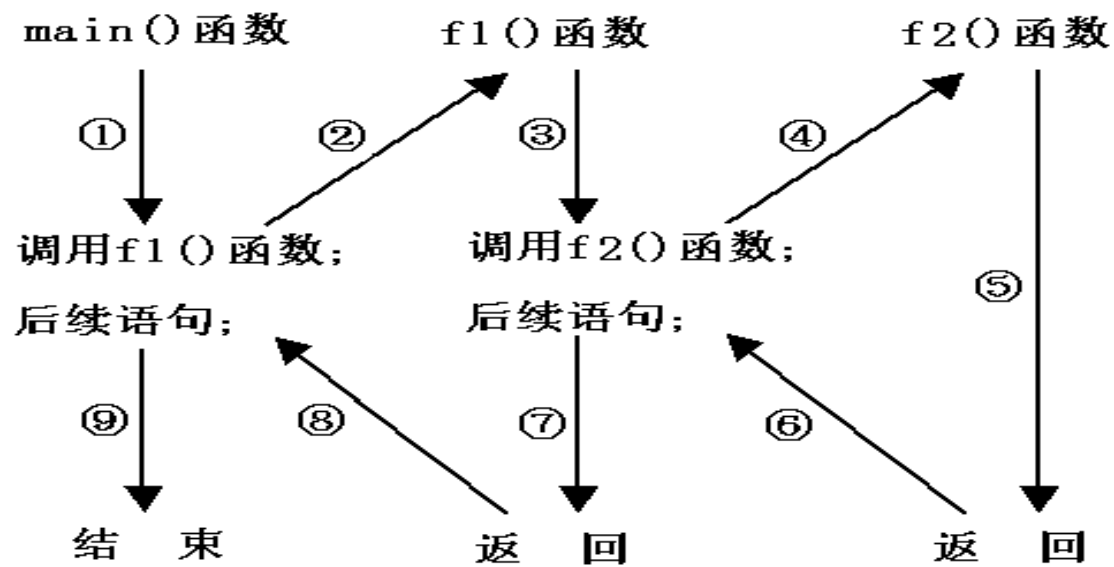
内容导航

CONTENTS

函数

- 函数的分类
 - 函数定义的一般形式
 - 函数的参数和函数的值
 - 函数的调用
 - 函数的嵌套调用
 - 函数的递归调用
 - 数组作为函数参数
- 局部变量和全局变量
 - 变量的存储类别

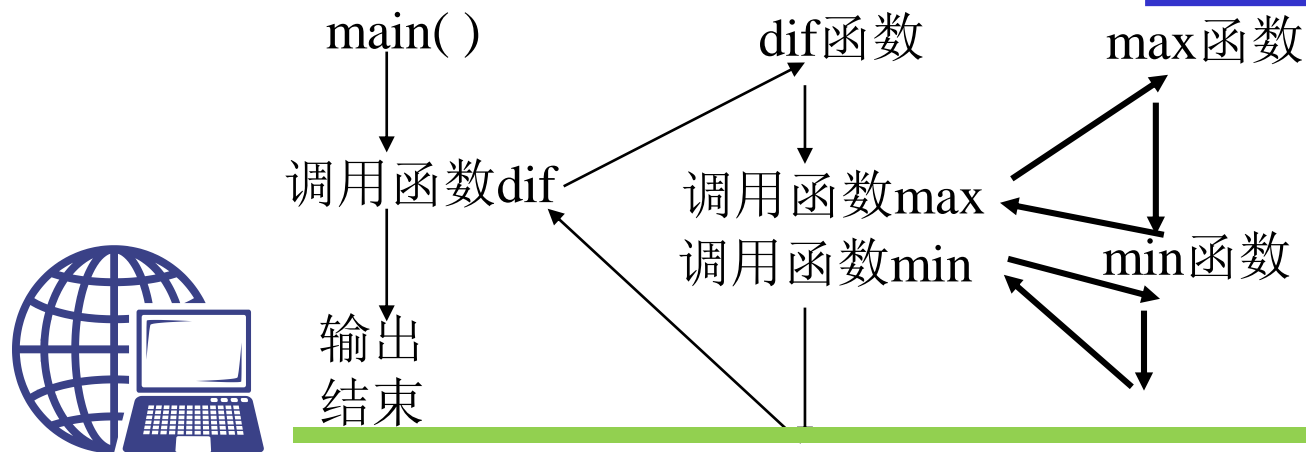
- **函数的嵌套调用**
- C规定：函数定义不可嵌套，但可以嵌套调用函数
- 函数的嵌套调用是指，在执行被调用函数时，被调用函数又调用了其它函数。这与其它语言的子程序嵌套调用的情形是类似的，其关系可表示如图7-1。



```
#include <stdio.h>
int dif(int x,int y,int z);
int max(int x,int y,int z);
int min(int x,int y,int z);
void main()
{ int a,b,c,d;
  scanf("%d%d%d",&a,&b,&c);
  d=dif(a,b,c);
  printf("Max-Min=%d\n",d);
}
```

```
int dif(int x,int y,int z)
{ return max(x,y,z)-min(x,y,z); }
int max(int x,int y,int z)
{ int r;
  r=x>y?x:y;
  return(r>z?r:z);
}
int min(int x,int y,int z)
{ int r;
  r=x<y?x:y;
  return(r<z?r:z);
}
```

例 求三个数中最大数和最小数的差值





内容导航

CONTENTS

函数

- 函数的分类
 - 函数定义的一般形式
 - 函数的参数和函数的值
 - 函数的调用
 - 函数的嵌套调用
 - 函数的递归调用
 - 数组作为函数参数
- 局部变量和全局变量
 - 变量的存储类别

- 函数的递归调用
- 函数的递归调用是指，一个函数在它的函数体内，直接或间接地调用它自身。
- C语言允许函数的递归调用。在递归调用中，调用函数又是被调用函数，执行递归函数将反复调用其自身。每调用一次就进入新的一层。
- 为了防止递归调用无终止地进行，必须在函数内有终止递归调用的手段。常用的办法是加条件判断，满足某种条件后就不再作递归调用，然后逐层返回。

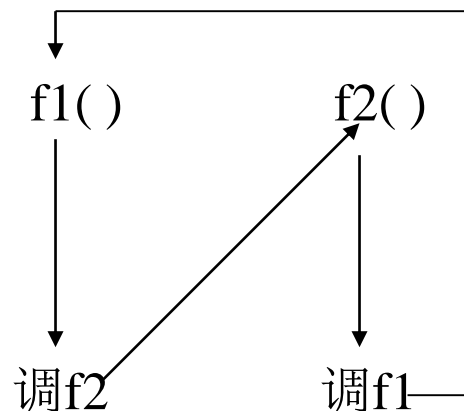
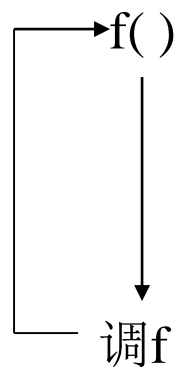


哪个是递归调用?
那个是嵌套调用?

```
int f(int x)
{
    int y,z;
    .....
    z=f(y);
    .....
    return(2*z);
}
```

```
int f1(int x)
{
    int y,z;
    .....
    z=f2(y);
    .....
    return(2*z);
}
```

```
int f2(int t)
{
    int a,c;
    .....
    c=f1(a);
    .....
    return(3+c);
}
```



说明

C编译系统对递归函数的自调用次数没有限制

每调用函数一次，在内存堆栈区分配空间，用于存放函数变量、返回值等信息，所以递归次数过多，可能引起堆栈溢出

