第六章 函　数

【教学目的】

理解“函数”概念，学会构造程序模块
掌握函数的定义和调用
掌握函数之间的信息传递
掌握函数的递归调用
熟悉 C 标准库中常见的库函数

【教学内容】

正确定义函数及其声明
函数的调用的方式及其返回值
函数的参数，两种重要的传递方式——传值调用和传址调用
命令行参数
函数的递归调用

【教学重点和难点】

掌握函数的定义，在实际问题中如何构造函数这样的程序模块。

掌握函数的传值调用和传址调用，特别是传址调用。注意两种方式的区别，什么情况下用哪一种方式更高效，更方便。

主函数带参数的情况，参数如何传递，程序如何执行。

掌握递归算法，包括递归的基本概念，递归函数的动态执行过程，递归程序的编写方法。

【问题的提出】

在前五章中所出现的程序都只有一个 main 函数，当开发和维护大型程序时，程序往往很大，用一个 main 函数编写的程序会很长，不利于多人合作开发大型程序（或软件），也不利于程序的阅读和调试。在这种情况下提出了一种好的办法，就是用更容易管理的小程序块（即模块）来建立程序，这种程序设计技术称为“细化”(divide and conquer)。

从组成上看，各个功能模块彼此有一定的联系，功能上各自独立，从开发过程上看，可能不同的模块由不同的程序员开发，怎样将不同的功能模块连接在一起，成为一个程序，怎样保证不同的开发者的工作既不重复，又能彼此衔接，这就是模块化设计。在 C 语言中，用函数实现功能模块的定义，一个文件中可以包含多个函数。

C 语言的程序通常是用程序员编写的新函数和 C 标准库中的函数写成的。C 标准库中提供了丰富的函数集，这些函数能够完成常用的数学计算、字符串操作、字符操作、输入／输出以及其它许多有用的操作。这些函数给程序员提供了很多必要的功能，减少了程序员的工作量，节省了开发时间，使程序具有更好的可移植性。标准库函数存放在不同的头文件中，使用时只要把头文件包含在用户程序中，就可以直接调用相应的库函数了。即在程序开始部分用如下形式：　　#include <头文件名> 或：#include "头文件名"。

用户可根据自己的需要，编写完成指定任务的函数，这些函数称为“自定义函数”。函数的使用是通过函数调用实现的，函数调用指定了被调用函数的名字和调用函数所需的信息(参数)。

【教学要点】

1. 函数定义、声明、调用及返回值

【例 6.1】定义一个函数，用于求两个数中较大的数。

分析： 根据题意，需要自定义一个函数实现求大数的功能；而在 main( )中实现两数的输入、调用自定义函数及输出功能。

源程序如下：

```
main()
{ float max(float n1, float n2) ;   /* 在 main( )中对被调用函数 max( )进行声明 */
   float num1,num2,m ;
   printf("input two numbers:\n") ;
   scanf("%f%f", &num1, &num2) ;   /* 输入两个数 */
   m=max(num1,num2);          /* 调用 max( )函数，返回后的较大值存于 m 中 */
   printf("max=%f\n",m) ;     /* 输出较大数 */
   getch();
}
float max(float n1, float n2)   /* 定义一个函数 max( )求较大数，其返回值为 float 型*/
{ return (n1>n2?n1:n2) ;     /* 返回表达式的值 */
}
```

说明：

1. 任何一个函数都是由函数说明和函数体两部分组成。根据函数是否需要参数，可将函数分为无参函数和有参函数两种。在本例中定义一个函数 max( )是有参函数，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。本例中将实参 num1、num2 的值分别传递给形参 n1、n2。
2. 在 C 语言中，所有函数（包括主函数 main( ) ）都是平行的。一个函数的定义，可以放在程序中的任意位置，主函数 main( ) 之前或之后。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( ) 函数可以调用其它函数，不允许被其它函数调用。
3. max( )函数名前面的类型标识符为 float 型， C 语言规定，函数的默认类型为 int 型。若非默认类型被调用函数的定义在调用函数之后出现，就必须在调用函数中对被调用函数加以说明。
4. 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。 为了使程序具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（void）。

2. 函数参数——传值调用

【例 6.2】输入两个整数，编写一个函数，其功能是：交换其值后输出。

分析：在 main( )中实现两数的输入、调用自定义函数 swap( )实现交换。因为用 return 语句时，只能返回一个值，交换后的两数不能都通过 return 返回，可考虑在 swap( )中输出。

源程序如下：

```
void swap(int a,int b)     /* 定义在 main( )之前，空类型，无返回值 */
```

```
{   int t;
  t=a; a=b; b=t;        /* 两数交换 */
 printf("1、a=%d,b=%d\n",a,b);    /* 输出交换的结果 */
 }
main( )
 { int a,b;
    a=10;b=20;
    swap(a, b);      /* 调用 swap( )函数 */
    printf("2、a=%d,b=%d\n",a,b);        /* 输出 a、b 的值 */
 }
```

运行结果： 1、a=20,b=10
2、a=10,b=20

说明：

1. 本例中函数的调用是传值调用（它是一种单向传递方式，只能由调用函数向被调用函数传递值），将实参 a、b 的值复制 1 份分别传递给形参 a、b，从而实现调用函数向被调用函数的数据传送。调用函数时，实参在类型上按顺序与形参，必须一一对应和匹配。
2. 从结果中可以看出 main()中 a、b 的值保持不变，因为实参 a、b 为局部变量，即在函数内部定义的变量，只在本函数范围内有效。在 swap()函数中的形参 a、b 也为局部变量，只在 swap()函数函数范围内有效。不会出现同名混淆的情况，实参和形参各用各的内存单元，这时形参的变化不会影响实参。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。 返回调用函数后，则不能再使用该形参变量。
3. 因为用 return 语句时，只能返回一个值。思考：若要将交换后的结果返回 main()，该如何实现？即要想返回两个以上的值该如何实现？

3. 利用全局变量达到返回两个以上的值的目的

【例 6.3】将**【例 6.2】**改编，利用全局变量将交换后的值返回主函数。

分析：全局变量即在函数外部定义的变量，也称为外部变量，它的有效范围是从定义变量的位置到本源文件结束。

源程序如下：

```
int a,b;      /* 定义 a,b 为全局变量，有效范围是从这里到本源文件结束。*/
main( )
{ void swap( );      /* 定义在 main( )之后，需声明 */
  a=10;b=20;
  swap( );          /* 调用 swap( )函数* /
  printf("2、a=%d,b=%d\n",a,b);        /* 输出 a、b 的值 */
 }
void swap( )        /* 定义在 main( )之后，空类型，无返回值 */
{   int t;
    t=a; a=b; b=t;        /* 两数交换 */
  printf("1、a=%d,b=%d\n",a,b);    /* 输出交换的结果 */
}
```

运行结果： 1、a=20,b=10
2、a=20,b=10

说明：

1. 本例中定义 a、b 为全局变量，在 main( )函数和 swap( )函数中的 a、b 是同一个变量，应此在 swap( )函数中进行交换的 a、b 实际上就是 main( )函数中的 a、b。
2. 采用全局变量虽然达到了目的，但在编写大程序并不是一种好的编程方法。因为定义了全局变量，在调用 swap( )函数时，要想知道 a、b 的值，就必须阅读所有与全局变量相关的函数才能得知，而不能只阅读 swap( )函数。那么，更好实现方法是什么呢？这就是函数的传址调用。

4. 函数参数——传址调用

【例 6.4】将变量的地址值或指针变量作为函数的实参，指针变量作为函数的形参，编写函数实现两数的交换。

源程序如下：

```
void swap(int *a1,int *b1)     /*  定义在 main( )之前，空类型，无返回值  */
   { int t;
      t=*a1; *a1=*b1; *b1=t;         /*  两数交换  */
      printf("1、a=%d,b=%d\n",*a1,*b1);     /*  输出交换后的结果  */
   }
 main( )
   { int a,b;
      a=10;b=20;
     swap(&a, &b);       /*  调用  swap( )函数  */
      printf("2、a=%d,b=%d\n",a,b);         /*  输出 a、b 的值  */
   }
```

运行结果： 1、a=20,b=10
2、a=20,b=10

说明：本例中函数的调用是传地址调用，由于实参分别是变量 a 和 b 的地址值，形参是指针变量，参数的传递是传地址。实参和相对应的形参占用的是同一存储空间，形参变量*a1 和*b1 的交换实际上就是相应的实参的交换，从而得到了上面的运行结果。

【例 6.5】将数组名或指针变量作为函数的形参和实参，编写函数 aver( )实现已知某个学生 5 门课程的成绩，求平均成绩，在 main( ) 函数中完成输入输出功能。

分析：在 main( ) 函数输入 5 门成绩，用一维数组存放，在 aver( )中要获得这 5 门成绩，可定义一个形参数组（或指针变量），进行地址传递。此时形参数组和实参数组共享同一存储空间。

源程序如下：

```
float aver(float a[ ])     /*  求平均值函数，形参为一数组  */
{ int i;
   float average,s= 0;
   for(i=0; i<5; i++) s += a[i];     /*  求成绩之和  */
   average=s/5;
   return average;       /*  返回平均值  */
}
main( )
{ float score[5],av;
   int i;
   printf("\n input 5 scores:\n");
```

```
    for(i=0; i<5; i++) scanf("%f",&score[i]);   /* 输入第 1- 5 个成绩 */
    av=aver(score);        /* 调用函数，实参为一数组名 */
    printf("average score is % 5.2f \n",av);
    getch();
}
```

说明 ：

1. 用数组名作函数实参数，可在被调用函数中定义形参数组，且两者数据类型必须一致，否则结果将出错。除此之外，被调用函数中的形参还可定义为指针变量。例如在本例中， float aver(float a[ ]) 可改为： float aver(float *a) 。
2. C 编译系统对形参数组大小不作检查，所以形参数组可以不指定大小，也可指定大小，指定时长度大于等于实参数组的长度。例如在本例中， float aver(float a[ ]) 可改为：float aver(float a[5])。

【例 6.6】编写函数 fun( )实现：把字符串 s 中出现的每一个字符，紧随其后重复出现一次，形成一个新串放到 t 中，且在 t 中把相邻字符的位置进行交换。在 main( ) 函数中完成输入输出功能。例如：当字符串 s 为：“12345”时，t 为“2211443355”。

分析：主函数中输入串 s 后，调用 fun( )函数，两者之间进行地址传递，实参用一维字符数组名，形参用指针变量来实现。

源程序如下：

```
#include <stdio.h>
#include <string.h>
void fun (char *s, char *t)     /* 指针变量作形参 */
{ int i,j,sl;
   sl = strlen(s);                  /* 求串 s 的长度 */
   for (i=0, j=0; i<sl; i+=2)   /* 每循环一次，就依次将相邻的两数存入 t 中 */
      { if (i+1 < sl)             /* 判断后一个数是否越界 */
          {   t[j] = s[i+1];   t[j+1] = s[i+1];   /* 先将后一个数存入 */
              j=j+2;   }
        t[j] = s[i];   t[j+1] = s[i];          /* 再存入前一个数 */
        j=j+2;
      }
      t[2*sl] = '\0';          /* 在串的末尾加上\0 */
}
main()
{   char s[100], t[100];
    printf("\n Please enter string s:");
    scanf("%s", s);
    fun(s,t);            /* 函数调用，数组名作实参 */
    printf("The result is: %s\n", t);
}
```

【例 6.7】用指向函数的指针实现函数的调用，编程求 a、b 两数的大者。

分析：C 程序是由函数组成的。每个函数的原代码在内存中占一连续的存储单元。函数名代表该区域的首地址，称为函数的入口地址。 指针变量既然可以指向整型变量、字符串、数组、也可以指向函数，即可以存放函数的入口地址，这种指针变量称为函数指针。前面所学的都是用函数名调用，如何用指向函数的指针实现函数的调用呢？只需用函数指针代替函数名即可。

源程序如下：

```
main( )
{ int (*p)( );        /*  定义指针 p 是一个指向函数的指针  */
   int max( ),a,b,c;
   scanf("%d%d",&a,&b);
   p=max;     /*  指针变量 p 指向函数 max，只需给出函数名，不必给出参数    */
   c=(*p)(a,b);     /*  用函数指针变量调用函数时，只需用(*p)代替函数名即可    */
   printf("max=%d\n",c);
}
max(int x,int y)       /*    max( )函数实现求两数大者  */
{ int z;
 z=x>y?x:y;
 return(z);
}
```

5. 命令行参数——带形参的 main( )

【例 6.8】从命令行输入一个整数，编写一个函数判断该整数是否为素数。

分析：在前面所学的章节中，main( )函数的括号中都是不带参数的。实际上，main( )函数既可以是无参函数，也可以是有参函数。由于 C 程序是从 main( )函数开始执行的，其它任何函数都不能调用 main( )函数，所以也就无法向 main( )函数传递信息，只能通过程序之外给它传递信息，因此 main( )函数中形参相对应的实参是由操作的命令行提供，所以通常称这些形参为命令行参数。

源程序如下：（源程序名为 L6-8.c）

```
#include <math.h>
# include <string.h>
main(int argc,char *argv[] )     /*  带两个形参的 main( )    */
{ int a;
   if (argc!=2) exit(0);/*  命令行字符串个数不等于 2，输入有错，exit()终止程序执行  */
   a=atoi(argv[1]);   /* atoi( )将字符型数据转换为整型数据  */
   flag=isprime(a);   /*  调用函数 isprime( )，返回值赋予标志变量 flag */
   if(flag==1)          /*  判断标志变量 flag 是否为 1，是，输出 a 是素数  */
         printf("\n %d is prime.",a);
   else printf("\n %d is not prime.",a);     /*  否，输出 a 不是素数  */
}
 int isprime(int a)     /* isprime( )函数，实现判断 a 是否为素数  */
{ int i;
   for(i=2;i<sqrt(a);i++)
      if(a%i==0) return 0;     /*  不是素数，返回 0 值  */
   return (1);       /*  是素数，返回 1 值  */
}
```

运行过程:源程序名为 L6-8.c，经过编译、连接成功后，得到一个可执行文件 L6-8.exe，然后在操作系统提示符下，输入运行命令的为：L6-8.exe 56 后回车，屏幕上将输出 56 is prime.。其中 L6-8.exe 为命令，它的.exe 可以省略，56 为参数，两者用空格分开。

说明 ：

1. 带两个形参的 main( )，第一个形参是整型变量，argc 记录了命令行中命令与参数的个数，共 2 个；第二个形参是一个字符型的指针数组，指针数组 argv 的大小由参数

argc 的值决定，数组元素中依次存放的是命令行中各字符串的首地址。即为 char *argv[2]，其中 argv[0]指向 L6-8.exe，argv[1]指向 56。

2. 命令行中的每一部分都被处理为字符串，因此字符串 56 要先由 atoi( )将它转换为整型数据。
3. 带形参的 main( )不能用”Run”命令来运行。

6. 函数的递归调用

【例 6.9】输入一个串，使用指针编写递归函数，实现串的反向输出。

分析：设计递归算法，可以描述如下：

1. 定义两个字符指针分别指向字符串的首字符和最后一个字符('\0'前的字符)；
2. 将指针所指的两个字符进行交换；
3. 使中间部分构成"新的"字符串，递归并对其进行串反向操作。

源程序如下：

```
void    fx( char * p1 )      /*   指针 p1 获得字符串的首地址   */
{ char *p2=p1, t;
   while (*p2) p2++;       /*   确定串结束标记'\0'的位置    */
   p2--;                   /*    p2 指向'\0'之前的最后一个字符    */
   if (p1<p2)              /*  递归调用结束的条件，若条件不成立，返回到上一层  */
     {   t=*p1;            /*   将串首字符存到中间变量 t 中   */
         *p1=*p2;          /*   将串尾字符存到串的首位置   */
         *p2='\0';         /*  形成一个"新"的字符串   */
         fx(p1+1);         /*   对"新串"（起始地址为 p1+1）进行递归调用   */
         *p2=t;            /*   将串首字符存到串的最后面的位置   */
     }
}
main( )
{ char s[50];
   printf("input a string(<50): ");
   gets(s);        /*  输入串  */
   fx( s );        /*  函数调用，其参数为数组名，进行地址传递  */
   printf(“result is :”);
   puts(s);        /*  输出串  */
   getch();
}
```

说明：

1. 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。 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。
2. 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。
3. 思考：用非递归算法该如何求解？

【例 6.10】用递归法和非递归法实现，输入任意一个整数，在各数位间插入空格后输出。如输入 1234，输出 1 2 3 4 。

方法一：用递归法

分析：定义变量 n，利用算术运算做 n/10，得到下一次的 n 值，递归调用层层回推，如

1234—>123—>12—>1，直到最后一次的 n <10，再利用算术运算求 n%10，层层递推返回，如余数为 1—>2—>3—>4，逐层输出。

源程序如下：

```
main()
{ void fun(long m);        /* 函数声明 */
   long int n;             /* 定义 n 为长整型 */
   printf("input a inteager number:\n");
   scanf("%ld",&n);
   fun(n);                 /* 函数调用 */
   getch();
}
void fun(long m)
{ if (m>=10)
  fun(m/10);       /* 递归调用 */
   printf("%d   ",m%10);
}
```

方法二：用非递归法

分析：定义变量 n，利用算术运算求 n%10，可定义一个数组来存放每一次的余数；利用算术运算做 n/10，得到下一次的 n 值，重复这个过程，直到 n <=0。

源程序如下：

```
main()
{ long int n; int a[10],i,k;
   printf("input a inteager number:\n");
   scanf("%ld",&n);
   k=fun(n,a);              /* 函数调用,返回 n 的位数个数 */
   printf("result is : ");
   for(i=k-1;i>=0;i--)     /* 利用循环输出各位数，并用空格分开 */
      printf("%2d",a[i]);
   getch();
}
int   fun(long m,int a[])
{ int i=0;
   while (m>0)
      {a[i++]=m%10;     /* 将余数依次放入数组中 */
       m=m/10;     /* 求下一次 m 的值 */
      }
   return(i);
}
```

说明：在本例中对同一问题而言，递归算法与非递归算法是求解一个问题的两种不同方法，是从两个不同的角度看待相同的问题。在学习递归程序设计的过程中，应当将递归算法与同一问题的非递归算法进行比较，分析问题的数学模型、程序实现过程、程序运行结果、程序实现难度与长度等，从中找出递归算法与非递归算法之间的联系与区别，体会同一问题使用不同算法求解的难度与思路。

【小结】

1、 本章的基本内容是 C 语言关于函数的使用，包括：函数的一般定义方法、函数说明规定、函数返回、函数的返回值和函数的调用。

2、重点介绍函数之间参数的传递，包括：在函数调用时形式参数与实际参数的对应关系，参数传递的方式（值传递和地址传递）。

3、 简介变量的存储类型，包括：局部变量和全局变量的说明方式、特点和适用的范围，不同存储类型变量在使用时的区别。

4、 本章中的难点之一是递归算法，递归作为一种常用的程序设计方法，可以很方便地解决不少特定的问题。有些问题只能用递归算法实现。