第 10 章

常用库函数的应用

本章内容提要

C 语言标准库中为用户提供了功能强大而且丰富的内置函数，使用这些函数可以在很大程度上减少代码的开发量，降低代码开发的难度。如使用的函数 printf()和函数 scanf_s()，就是由标准输入输出库提供的。本章就来介绍常用库函数的应用，包括主函数 main()的应用、数学函数的应用、字符串函数的应用、时间和日期函数的应用等。


微视频

10.1 认识 C 语言标准库函数

C 语言标准库函数中包含有 15 种功能强大的函数，通过使用这些库函数能够减少开发人员编写程序的难度以及开发程序的工作量，15 种标准库函数的头文件名称以及说明，如表 10-1 所示。

表 10-1 C 语言 15 种标准库函数的头文件名称以及说明

头文件名称	说　明
<assert.h>	包含断言宏，被用来在程序的调试版本中帮助检测逻辑错误以及其他类型的 bug
<ctype.h>	定义了一组函数，用来根据类型来给字符分类，或者进行大小写转换，而不关心所使用的字符集
<errno.h>	错误检测
<float.h>	系统定义的浮点型界限
<limits.h>	系统定义的整数界限
<locale.h>	定义 C 语言本地化函数
<math.h>	定义 C 语言数学函数
<stjump.h>	定义了宏 setjmp 和 longjmp，在非局部跳转的时候使用
<signal.h>	定义 C 语言信号处理函数
<stdarg.h>	可变长度参数处理
<stddef.h>	系统常量
<stdio.h>	输入输出
<stdlib.h>	多种公用
<string.h>	定义 C 语言字符串处理函数
<time.h>	时间和日期

10.2　主函数的应用


微视频

C 语言提供了一个很特殊的函数，这个函数就是主函数，之所以说它特殊，是因为它的作用是“统领”其他自定义函数的，其他函数都必须在它的控制下才能使用。

10.2.1　主函数的作用

当需要用 C 语言完成一个复杂的功能时，先将这个复杂的功能划分成许多小的功能，然后使用函数声明定义实现这些小的功能，最后使用函数调用，将这些函数“串”起来，来完成复杂的功能。在“串”函数时，就会出现两个问题，首先谁来调用第一个函数，也就是将这一串函数的头交给谁？其次，谁来接函数的尾，也就是这一串函数的尾交给谁。

这就用到主函数了，主函数会抓起这一串函数的头，拉起这一串函数的尾，然后将其交给计算机，这样我们书写的程序就会被计算机识别。主函数就是你写的函数与计算机融合的一个函数，只要将你的函数调用放到主函数的定义中，就可以实现这一融合。当然其中肯定还是会穿插非函数调用的语句用来实现一些简单的补充功能的。

10.2.2　主函数的声明

主函数是 C 语言规定的函数，对于它的形式是有要求的，主要体现在主函数的样子上，也就是主函数的函数名、参数和返回值类型上。

C 语言中主函数的声明可以有好几种形式，分别满足不同的 C 语言标准要求，最常使用的主要有以下 4 种。

（1）无返回值无参数：

```
void main()
```

（2）无返回值有参数：

```
void main(int argc,char *argv[])
```

（3）有返回值无参数：

```
int main()
```

（4）有返回值有参数：

```
int main(int argc,char *argv[])
```

这 4 种形式的区别主要在于是不是有返回值和参数上。如果有返回值，返回值类型必须是 int。如果无返回值，返回值类型是 void。如果有参数，参数必须是一个 int 类型的变量和一个一维数组，其中保护的是字符指针，变量名要求一个是 argc，另一个是 argv。当然，也可以使用其他变量名，不过最好按照规定来，使用 argc 和 argv 这两个变量名，以免特殊情况。如果没有参数，就什么都不用写。

对于这 4 种形式，最新的标准要求为最后 2 个，即可以没有参数，但必须有返回值。现在所有的编译器基本上都支持最后两种函数 main()声明形式，以免出现问题。所以在此建议大家最好使用有返回值的函数 main()形式。

10.2.3 主函数的参数

在使用主函数时，一般是没有参数的，但是这不代表主函数是没有参数设置的，主函数有两个参数，一个是 int argc，另一个是 char *argv[]。

1. argc 参数

argc 参数是主函数的第一个参数，它是一个整型变量，这个变量是系统在运行程序时使用的。它代表的是在用命令行运行程序时，输入的命令中包含的字符串的个数。当什么参数也没有时，它的值是 1。

2. argv 参数

系统在运行程序的时候，光传入命令中有几个字符串是不够的，还需要告诉程序，命令中的字符串都是什么，不然程序就没法使用用户输入的命令，计算机系统就是使用 argv 这个一维数组来告诉程序，用户输入的命令中的字符串都是什么。

argv 是一个一维数组，其中保存的是命令中的每个字符串在内存中的地址，通过这个地址就可以知道用户输入命令中的字符串。对数组进行遍历，就可以一次访问到第二个、第三个、…、第 argc 个字符串了。例如我们运行一个控制台程序 test.exe，参数是 aaa、bbb、ccc，这里写成./test aaa bbb ccc，那么 argc 的值就是 4，argv[1]的值就是 aaa，argv[2]的值就是 bbb，argv[3]的值就是 ccc。

10.2.4 主函数的返回值

按照 C 语言中主函数的 4 种形式，主函数的返回值类型要么为空，要么为整型。对于返回值为空的主函数，一般不推荐，如果非要使用这种形式，可以不用管主函数的返回值，否则返回其他值。

主函数的完整使用形式有如下两种。

（1）不带参数形式（其中参数 void 可以省略不写）。

```
int main(void)
{
…
return 0;
}
```

（2）带参数形式。

```
int main(int argc,char *argv[])
{
…
return 0;
}
```

写程序的时候，就按照这两种方式来写主函数，如果在程序运行的时候传入数据，就使用第一种形式，否则使用第二种形式。主函数的函数体中实现对其他函数的调用，通过函数调用及其他语句来完成我们想要计算机完成的功能。


微视频

10.3 数学函数的应用

数学计算是计算机最擅长的运算方式，计算机大部分运算方法都是基于数学运算执行的。

C 语言提供了很多用于数学计算的库函数，要使用这些函数，在程序文件头中必须加入#include <math.h>语句。下面就介绍一些最常用的数学函数。

10.3.1　三角函数

三角函数常用的正弦、余弦和正切函数形式如表 10-2 所示。

表 10.2　三角函数

原　　型	功　　能
double sin(double x)	计算双精度实数 x 的正弦值
double cos(double x)	计算双精度实数 x 的余弦值
double tan(double x)	计算双精度实数 x 的正切值
double asin(double x)	计算双精度实数 x 的反正弦值
double acos(double x)	计算双精度实数 x 的反余弦值
double atan(double x)	计算双精度实数 x 的反正切值
double sinh(double x)	计算双精度实数 x 的双曲正弦值
double cosh(double x)	计算双精度实数 x 的双曲余弦值
double tanh(double x)	计算双精度实数 x 的双曲正切值

要正确使用三角函数，需要注意参数范围：

对于 sin 和 cos 函数，其参数 x 的范围是[−1,1]；

对于 asin 的 x 的定义域为[−1.0，1.0]，值域为[−π/2，+π/2]；

对于 acos 的 x 的定义域为[−1.0，1.0]，值域为[0，π]；

对于 atan 的值域为(−π/2，+π/2)。

【例 10.1】编写程序，使用数学函数中的三角函数（源代码\ch10\10.1.txt）。

```
#include <stdio.h>
/* 数学函数头文件 */
#include <math.h>
#define PI 3.14
int main()
{
  double x;
  x=PI/2;
  /* 三角函数 */
  printf("sin(PI/2)=%.2f\n",sin(x));
  x=PI/4;
  printf("cos(%.4f)=%.4f\n",x,cos(x));
  printf("tan(PI/4)=%f\n",tan(x));
  printf("sinh(%.4f)=%.4f\n",x,sinh(x));
  printf("cosh(%.4f)=%.4f\n",x,cosh(x));
  printf("tanh(%.4f)=%.4f\n",x,tanh(x));
  x=0.45;
  printf("asin(%.2f)=%.4f\n",x,asin(x));
  printf("acos(%.2f)=%.4f\n",x,acos(x));
  printf("atan(%.2f)=%.4f\n",x,atan(x));
  return 0;
}
```

程序运行结果如图 10-1 所示。

图 10-1　例 10.1 的程序运行结果

10.3.2　绝对值函数

绝对值函数 abs()用于计算整数的绝对值，使用语法如下：

```
int abs(int x);
```

表示求解整数 x 的绝对值。

绝对值函数 fabs()用于计算浮点数的绝对值，使用语法如下：

```
float fabs(float x);
```

表示求解浮点数 x 的绝对值。

【例 10.2】编写程序，求不同类型数值的绝对值（源代码\ch10\10.2.txt）。

```
#include <stdio.h> /*包含标准输入输出头文件*/
#include <math.h>  /*包含数学头文件*/
int main()
{
int x;
double y;
x=-5;
printf("|%d|=%d\n",x,abs(x));     /*调用绝对值函数*/
x=0;
printf("|%d|=%d\n",x,abs(x));     /*调用绝对值函数*/
x=+5;
printf("|%d|=%d\n",x,abs(x));     /*调用绝对值函数*/
y=2.15;
printf("|%.2f|=%.2f\n",y,fabs(y));
y=-3.34;
printf("|%.2f|=%.2f\n",y,fabs(y));
return 0;
}
```

程序运行结果如图 10-2 所示。本程序的主要功能是使用函数 abs()计算正整数、零和负整数的绝对值，使用函数 fabs()计算浮点数的绝对值。

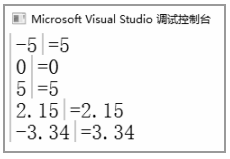

图 10-2　例 10.2 的程序运行结果

10.3.3　幂函数和平方根函数

求幂函数 pow()用于求实数的 N 次幂，使用语法如下：

```
double pow(double x, double y);
```

表示求解双精度实数 x 的 y 次幂。

开平方函数 sqrt()用于求解实数的平方根，使用语法如下：

```
double sqrt(double x);
```

表示计算双精度实数 x 的平方根。

【例 10.3】编写程序，使用数学函数中的求幂函数与开平方跟函数（源代码\ch10\10.3.txt）。

```
#include <stdio.h>
/* 数学函数头文件 */
#include <math.h>
int main()
{
  double x,y,z;
  printf("请输入一个实数: \n");
  scanf_s("%lf",&x);
  /* 求幂函数 */
  y=pow(x,2);
  /* 开平方函数 */
  z=sqrt(x);
  printf("%.2f 的 2 次幂为 %.2f\n",x,y);
  printf("对 %.2f 开 2 次平方根为 %.2f\n",x,z);
  return 0;
}
```

保存并运行程序，这里输入实数 9，然后显示输出结果如图 10-3 所示。

图 10-3　例 10.3 的程序运行结果

10.3.4　指数函数和对数函数

指数函数和对数函数互为逆函数，e 是自然对数的底，值是无理数 2.718281828……。形式如表 10-3 所示。

表 10-3　指数函数和对数函数

原　　型	功　　能
double exp(double x)	计算 e 的双精度实数 x 次幂
double log(double x)	计算以 e 为底的双精度实数 x 的对数 ln(x)
double log10(double x)	计算以 10 为底的双精度实数 x 的对数 log10(x)

指数函数 exp()用于求 e 的 N 次幂，使用语法如下：

```
double exp(double x);
```

表示计算 e 的双精度实数的 x 次幂。

对数函数 log()与 log10()分别用于计算以 e 为底和以 10 为底的实数的对数，使用语法如下：

```
double log(double x);
double log10(double x);
```

表示计算以 e 为底的实数 x 的对数 ln(x)和计算以 10 为底的实数 x 的对数 log10(x)。

【例 10.4】编写程序，使用数学函数中的指数函数与对数函数（源代码\ch10\10.4.txt）。

```
#include <stdio.h>
/* 数学函数头文件 */
#include <math.h>
```

```
#define E 2.718281828
int main()
{
  /* 指数函数 */
  printf("e=%f\n",exp(1.0));
  /* 对数函数 */
  printf("ln(e)=%f\n", log(E));
  printf("lg(5)=%f\n", log10(5.0));
  return 0;
}
```

程序运行结果如图 10-4 所示。

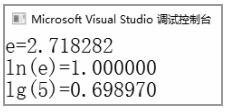

图 10-4　例 10.4 的程序运行结果

10.3.5　取整函数和取余函数

取整函数用于获取实数的整数部分，取余函数用于获取实数的余数部分，形式如表 10-4 所示。

表 10-4　取整函数和取余函数

原　　型	功　　能
double ceil(double x)	计算不小于双精度实数 x 的最小整数
double floor(doulbe x)	计算不大于双精度实数 x 的最大整数
double fmod(double x,double y)	计算双精度实数 x/y 的余数，余数使用 x 的符号
double modf(double x,double *ip)	把 x 分解成整数部分和小数部分，x 是双精度浮点数，ip 是整数部分指针，返回结果是小数部分

假设 x 的值是 74.12，则 ceil(x)的值是 75，如果 x 的值是−74.12，则 ceil(x)的值是−74。

假设 x 的值是 74.12，则 floor(x)的值是 74，如果 x 的值是−74.12，则 floor(x)的值是−75。

【例 10.5】编写程序，使用数学函数中的取整函数与取余函数（源代码\ch10\10.5.txt）。

```
#include <stdio.h>
#include <stdio.h>
/* 数学函数头文件 */
#include <math.h>
int main()
{
  float a,b;
  double x,y;
  a=3.14;
  /* 取整函数 */
  printf("ceil(%.2f)=%.0f\n",a,ceil(a));
  printf("floor(%.2f)=%.0f\n",a,floor(a));
  a=-3.14;
  printf("ceil(%.2f)=%.0f\n",a,ceil(a));
  printf("floor(%.2f)=%.0f\n",a,floor(a));
  /* 取余函数 */
  a=3.14;
  b=1.5;
  printf("3.14/1.5 的余数为 %.4f\n",fmod(a,b));
  b=-1.5;
```

```
    printf("3.14/(-1.5) 的余数为 %.4f\n",fmod(a,b));
    /*函数 modf()*/
    y=modf(-3.14,&x);
    printf("-3.14=%.0lf+(%.2f)\n",x,y);
    return 0;
}
```

程序运行结果如图 10-5 所示。

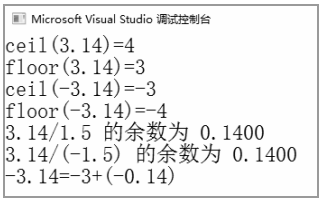

图 10-5　例 10.5 的程序运行结果

10.4　字符串处理函数的应用


微视频

C 语言中，字符串处理异常频繁，经常需要对字符串进行输入、输出、合并、修改、比较、转换等操作。为了高效统一地进行字符串处理，C 语言提供了丰富的字符串处理函数，要使用这些函数，在程序文件头中必须加入#include < string.h>语句。下面就介绍一些最常用的字符串处理函数。

10.4.1　字符串长度函数

字符串长度函数 strlen()用于返回字符串的长度，不包含结束符 NULL，它的使用语法如下：

```
int strlen(char *s);
```

其中，s 为指向字符串的指针。

【例 10.6】编写程序，使用字符串长度函数获取字符串的长度（源代码\ch10\10.6.txt）。

```
#include <stdio.h>
/* 字符串函数头文件 */
#include <string.h>
int main()
{
  char *s="Hello World";
  printf("字符串 %s 包含 %d 个字符\n",s,strlen(s));
  return 0;
}
```

程序运行结果如图 10-6 所示。

图 10-6　例 10.6 的程序运行结果

10.4.2　字符串连接函数

字符串连接函数 strcat_s()可将两个字符串连接在一起，它的使用语法如下：

```
char *strcat(char *s1,sizeof(string),char *s2);
```

表示将 s2 所指字符串添加到 s1 的结尾处（覆盖 s1 结尾处的'\0'）并添加'\0'，返回指针 s1。

另外，字符串连接函数 strncat_s()也可以将两个字符串连接在一起，它的使用语法如下：

```
char *strncat_s(char *s1,sizeof(string),char *s2,int n);
```

表示将 s2 所指字符串的前 n 个字符添加到 s1 结尾处（覆盖 s1 结尾处的'\0'）并添加'\0'，返回指针 s1。

☆**大牛提醒**☆

s1 和 s2 所指内存区域不可以重叠，并且 s1 必须有足够的空间来容纳 s2 的字符串。

【例 10.7】编写程序，使用字符串连接函数连接不同的字符串（源代码\ch10\10.7.txt）。

```
#include <stdio.h>
/* 字符串函数头文件 */
#include <string.h>
int main()
{
  char s1[15] = "Hello";
  char* s2 = "World!";
  char* s3 = "~!~";
  /* 字符串连接函数 */
  strcat_s(s1, sizeof(s1), s2);
  printf("%s\n", s1);
  strncat_s(s1, sizeof(s1), s3, 1);
  printf("%s\n", s1);
  return 0;
}
```

程序运行结果如图 10-7 所示。

图 10-7　例 10.7 的程序运行结果

10.4.3　字符串复制函数

字符串复制函数可以将一个字符串复制到另一个字符串中，它的使用语法以及说明，如表 10-5 所示。

表 10-5　字符串复制函数使用语法以及说明

字符串复制函数的使用语法	说　明
char *strcpy(char *s1, sizeof(string), char *s2)	将 s2 所指由 NULL 结束的字符串复制到 s1 所指的数组中，返回 s1
char *strncpy(char *s1, sizeof(string), char *s2,int n)	将 s2 所指由 NULL 结束的字符串的前 n 个字符复制到 s1 所指的数组中，返回 s1
void *memcpy(void *s1, void *s2, int n)	由 s2 所指内存区域复制 n 字节到 s1 所指内存区域，返回 s1
void *memmove (void *s1, void *s2, int n)	由 s2 所指内存区域复制 n 字节到 s1 所指内存区域，返回 s1

注意：

（1）使用函数 strcpy()、函数 strncpy()以及函数 memcpy()时，s1 和 s2 所指内存区域不可以重叠，并且 s1 必须有足够的空间来容纳 s2 的字符串。

（2）函数 memmove()并不关心被复制的数据类型，只是逐字节地进行复制，这给函数的使用带来了很大的灵活性，可以面向任何数据类型进行复制。

【例 10.8】编写程序，使用字符串复制函数复制字符串（源代码\ch10\10.8.txt）。

```
#include <stdio.h>
/* 字符串函数头文件 */
#include <string.h>
int main()
{
  char *s1="apple";
  char a[20];
  char b[]="abcdef";
  char c[20];
  char d[20];
  /* 字符串复制函数 */
  strcpy_s(a, sizeof(a),s1);
  strncpy_s(b, sizeof(b),s1,strlen(s1));
  /* 需在结尾添加结束标志 */
  b[3]='\0';
  memcpy(c,s1,sizeof(c));
  memmove(d,s1+3,sizeof(s1)+1);
  printf("字符串 a 为 %s\n",a);
  printf("字符串 b 为 %s\n",b);
  printf("字符串 c 为 %s\n",c);
  printf("字符串 d 为 %s\n",d);
  return 0;
}
```

程序运行结果如图 10-8 所示。

图 10-8　例 10.8 的程序运行结果

10.4.4　字符串比较函数

字符串比较函数可以对两个字符串中字符的 ASCII 码值进行比较，它的使用语法以及说明，如表 10-6 所示。

表 10-6　字符串比较函数使用语法以及说明

字符串比较函数的使用语法	说　明
int strcmp(char *s1, char *s2)	比较字符串 s1 和 s2。若 s1<s2，返回负数；若 s1==s2，返回 0；若 s1>s2，返回非负数
int strncmp(char *s1, char *s2, int n)	比较字符串 s1 和 s2 的前 n 个字符。若 s1<s2，返回负数；若 s1==s2，返回 0；若 s1>s2，返回非负数
int memcmp(void *s1, void *s2, int n)	比较内存区域 s1 和 s2 的前 n 字节。若 s1<s2，返回负数；若 s1==s2，返回 0；若 s1>s2，返回非负数

【例 10.9】编写程序，以函数 memcmp()为例演示字符串的比较（源代码\ch10\10.9.txt）。

```
#include <stdio.h>
/* 字符串函数头文件 */
#include <string.h>
int main()
{
  char *s1="I love C!";
  char *s2="I love c!";
  int s;
```

```
    printf("字符串 s1=%s\n 字符串 s2=%s\n",s1,s2);
    /* 字符串比较函数 */
    s=memcmp(s1,s2,strlen(s1));
    if(!s)
    {
      printf("两个字符串相等\n");
    }
    else if(s<0)
    {
      printf("字符串 s1 小于 s2\n");
    }
    else if(s>0)
    {
      printf("字符串 s1 大于 s2\n");
    }
    return 0;
}
```

程序运行结果如图 10-9 所示。

图 10-9　例 10.9 的程序运行结果

10.4.5　字符串查找函数

字符串查找函数能够在字符串中查找某个字符出现的位置，它的使用语法以及说明，如表 10-7 所示。

表 10-7　字符串查找函数使用语法以及说明

字符串查找函数的使用语法	说　明
char *strchr (char *s, char c)	表示返回一个指向字符串 s 中 c 第 1 次出现的指针；或者如果没有找到 c，则返回指向 NULL 的指针
char *strstr(char *s1, char *s2)	表示返回一个指向字符串 s1 中字符 s2 第 1 次出现的指针；或者如果没有找到 s2，则返回指向 NULL 的指针
void *memchr(void *s, char c, int n)	表示返回一个指向被 s 所指向的 n 个字符中 c 第 1 次出现的指针；或者如果没有找到 c，则返回指向 NULL 的指针

【例 10.10】编写程序，使用字符串查找函数查找字符串（源代码\ch10\10.10.txt）。

```
#include <stdio.h>
/* 字符串函数头文件 */
#include <string.h>
int main()
{
  char *s1="I love C!";
  char *s2="love";
  char *p;
  /* 字符串查找函数 */
  p=strchr(s1,s2);
  if(p)
  {
    printf("%s\n",p);
  }
  else
```

```
    {
      printf("未找到! \n");
    }
    p=strstr(s1,s2);
    if(p)
    {
      printf("%s\n",p);
    }
    else
    {
      printf("未找到! \n");
    }
    p=memchr(s1,'l',strlen(s1));
    if(p)
    {
      printf("%s\n",p);
    }
    else
    {
      printf("未找到! \n");
    }
    return 0;
}
```

程序运行结果如图 10-10 所示。

图 10-10　例 10.10 的程序运行结果

10.4.6　字符串填充函数

字符串填充函数用于快速赋值一个字符到一个字符串，它的使用语法以及说明，如表 10-8 所示。

表 10-8　字符串填充函数

字符串填充函数的使用语法	说　明
void *memset(void *d; char c, int n)	使用 n 个字符 c 填充 void*类型变量 d

【例 10.11】编写程序，使用字符串填充函数填充字符串（源代码\ch10\10.11.txt）。

```
#include <stdio.h>              /*包含标准输入输出头文件*/
#include <string.h>             /*包含字符串处理头文件*/
int main()
{
   char array[]="Hello World";
   char *s=array;
   memset(s,'W',5);             /*调用字符串填充函数*/
   printf("%s",s);
   return 0;
}
```

程序运行结果如图 10-11 所示。本实例是把指针所指内存存储空间的前 5 个字符使用字符“W”填充。

图 10-11　例 10.11 的程序运行结果

10.4.7　字符串大小写转换函数

字符串大写转换函数_strupr_s()用于将字符串中出现的小写字母转换为大写字母，返回指向该字符串的指针，使用语法如下：

```
char *_strupr_s(char *s, size);
```

字符串小写转换函数_strlwr_s()用于将字符串中出现的大写字母转换为小写字母，返回指向该字符串的指针，使用语法如下：

```
char *_strlwr_s(char *s,size);
```

【例 10.12】编写程序，将字符串的大小写进行转换（源代码\ch10\10.12.txt）。

```
#include <stdio.h>
/* 字符串函数头文件 */
#include <string.h>
int main()
{
  char s[] = "I love C!";
  printf("原字符串为 %s \n", s);
  /* 大小写转换函数 */
  _strupr_s(s,10);
  printf("转换为大写 %s\n", s);
  _strlwr_s(s,10);
  printf("转换为小写 %s\n", s);
  return 0;
}
```

程序运行结果如图 10-12 所示。

图 10-12　例 10.12 的程序运行结果

☆**大牛提醒**☆

使用大小写转换函数时，不能使用指向常量字符串的指针进行传递，否则会出现异常。


微视频

10.5　字符处理函数的应用

在 C 语言中，除了字符串处理函数外，还有标准 C 语言字符处理函数，如字符的大小写转换函数、字符的类型判断函数等。使用字符处理函数需要添加字符处理函数的头文件#include <ctype.h>。

10.5.1　字符类型判断函数

字符的类型判断函数能够对指定的字符进行判断，这些字符可以是字母、数字、空格等。字符的判断函数使用语法以及说明，如表 10-9 所示。

表 10-9　字符的判断函数使用语法以及说明

字符判断函数的使用语法	说　明
int isalnum(int c)	当字符 c 是文字或数字时返回非零，否则返回零
int isalpha(int c)	当字符 c 是一个字母时返回非零，否则返回零
int iscntrl(int c)	当字符 c 是一个控制符时返回非零，否则返回零
int isdigit(int c)	当字符 c 是一个数字时返回非零，否则返回零
int isgraph(int c)	当字符 c 是可打印的（除空格外）返回非零，否则返回零
int islower(int c)	当字符 c 是小写字母时返回非零，否则返回零
int isprint(int c)	当字符 c 是可打印的（含空格）返回非零，否则返回零
int ispunct(int c)	当字符 c 是可打印的（除空格、字母或数字外）返回非零，否则返回零
int isspace(int c)	当字符 c 是一个空格时返回非零，否则返回零
int isupper(int c)	当字符 c 是大写字母时返回非零，否则返回零
int isxdigit(int c)	当字符 c 是十六进制数字时返回非零，否则返回零

【例 10.13】编写程序，使用字符判断函数对输入的字符进行判断（源代码\ch10\10.13.txt）。

```
#include <stdio.h>
/* 字符函数头文件 */
#include <ctype.h>
int main()
{
  int ch;
  printf("请输入一个字符: \n");
  ch=getchar();
  if(islower(ch))
  {
    printf("该字符是小写字母");
  }
  else if(isupper(ch))
  {
    printf("该字符是大写字母");
  }
  else if(isdigit(ch))
  {
    printf("该字符是数字");
  }
  else
  {
    printf("该字符是其他字符") ;
  }
  printf("\n") ;
  return 0;
}
```

程序运行结果如图 10-13 所示。

图 10-13　例 10.13 的程序运行结果

10.5.2 字符大小写转换函数

字符大写转换函数 toupper()用于将字符转换为大写英文字母，使用语法如下：

```
int toupper(int c);
```

字符小写转换函数 tolower()用于将字符转换为小写英文字母，使用语法如下：

```
int tolower(int c);
```

☆**大牛提醒**☆

若是字符本身为大写/小写，使用字符大写/小写转换函数时，该字符不会发生变化。

【例 10.14】编写程序，通过输入端输入一个英文字母，将这个字母转换为它的大/小写形式（源代码\ch10\10.14.txt）。

```
#include <stdio.h>
/* 字符函数头文件 */
#include <ctype.h>
int main()
{
  char ch;
  printf("请输入一个英文字母: \n");
  ch=getchar();
  if(islower(ch))
  {
    printf("该字符是小写字母, 转换为大写字母为: %c\n",toupper(ch));
  }
  else if(isupper(ch))
  {
    printf("该字符是大写字母, 转换为小写字母为: %c\n",tolower(ch));
  }
  else
  {
    printf("无法转换, 该字符不为英文字母\n") ;
  }
  return 0;
}
```

程序运行结果如图 10-14 所示。

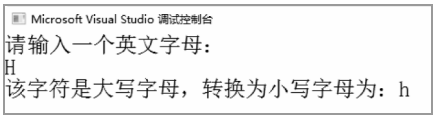

图 10-14 例 10.14 的程序运行结果

微视频

10.6 动态分配内存函数

为了实现内存的动态分配，C 语言提供了一些程序执行后才开辟和释放某些内存区的函数，包括函数 malloc()、函数 calloc()、函数 realloc()和函数 free()，通过这些函数可以实现根据程序的需要来动态分配存储空间。

10.6.1 函数 malloc()

函数 malloc()定义在头文件“stdlib.h”中，使用时需要添加此头文件，该函数的原型为：

```
void *malloc(unsigned int size);
```

它的作用是向系统申请一个确定大小（size 字节）的内存空间。若函数调用成功，则返回值为一个指向 void 类型的分配域起始地址的指针值；若函数调用失败，则返回值为空。

函数 malloc()的使用语法如下：

```
指针变量=(基类型*)malloc(内存空间字节数);
```

例如：

```
int *p;
p=(int*)malloc(sizeof(int));
```

注意：上述函数 malloc()分配的是一个整型空间，需要使用强制类型转换保证返回相对应的整型指针。

【例 10.15】编写程序，使用函数 malloc()进行动态内存分配，通过输入端输入一个数据并输出结果（源代码\ch10\10.15.txt）。

```
#include <stdio.h>
/* 添加头文件 */
#include <stdlib.h>
int main()
{
  int *p;
  int a;
  /* 调用函数动态分配内存 */
  p=(int*)malloc(sizeof(int));
  if(!p)
  {
    exit(0);
  }
  p=&a;
  scanf_s("%d",p);
  printf("a=%d\n",a);
  return 0;
}
```

程序运行结果如图 10-15 所示。本实例代码中，首先添加头文件“stdlib.h”，接着在主函数中定义一个指针变量 p，使用函数 malloc()进行动态内存的分配，并将分配的整型空间内存起始地址赋予指针变量 p，若分配成功，则通过输入端输入一个数据存储到该空间中，然后输出验证。

图 10-15　例 10.15 的程序运行结果

10.6.2　函数 calloc()

函数 calloc()定义在头文件“stdlib.h”中，使用时需要添加此头文件，该函数的原型为：

```
void *calloc(unsigned int n, unsigned int size);
```

它的作用是向系统申请 n 个 size 字节大小的连续内存空间，若是函数调用成功，则返回值为一个指向 void 类型的分配域起始地址的指针值；若是函数调用失败，则返回值为空。使用该函数能够为一维数组开辟一片连续的动态存储空间。

函数 calloc()的使用语法如下：

```
指针变量=(数组元素类型*)calloc(n, 每一个数组元素内存空间字节数);
```

例如：

```
int *p;
p=(int*)calloc(5, sizeof(int));
```

表示使用函数 calloc()动态分配 5 个大小为整型字节的连续内存空间，最后将返回的指针赋予指针变量 p，可以理解为该指针变量指向的是一个有 5 个元素的一维数组的首地址。

【例 10.16】编写程序，使用函数 calloc()动态分配一个包含有 5 个元素的一维数组的连续存

储空间，并分别为它们进行赋值，输出结果（源代码\ch10\10.16.txt）。

```
#include <stdio.h>
/* 添加头文件 */
#include <stdlib.h>
#define S 5
int main()
{
  int *p;
  int a,i;
  /* 调用函数动态分配一组内存 */
  p=(int*)calloc(S,sizeof(int));
  if(!p)
  {
    exit(0);
  }
  printf("请为%d个整型数据赋值: \n",S);
  for(i=0;i<S;i++)
  {
    scanf_s("%d",&a);
    *(p+i)=a;
  }
  printf("\n");
  for(i=0;i<S;i++)
  {
    printf("%d\t",*(p+i));
  }
  printf("\n");
  return 0;
}
```

程序运行结果如图 10-16 所示。例 10.16 代码中，首先添加头文件“stdlib.h”并定义一个符号常量 S，用于表示开辟元素的个数。接着在主函数中定义一个指针变量 p，然后调用函数 calloc()动态的分配一组拥有 5 个元素大小为整型字节的连续存储空间，并将首地址赋予指针变量 p，若是函数调用成功，则通过输入端为这 5 个元素进行赋值，最后输出赋值结果。

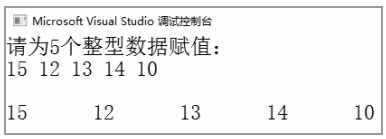

图 10-16 例 10.16 的程序运行结果

10.6.3 函数 realloc()

函数 realloc()定义在头文件“stdlib.h”中，使用时需要添加此头文件，该函数的原型为：

```
void *realloc(void *p, unsigned int size);
```

它的作用是向系统申请一个大小为 size 的内存空间，并将指针变量 p 指向的空间大小改为 size，同时将原存储空间中存放的数据传递到新的地址空间的低端，原存储空间的数据将会丢失。若是函数调用成功，则返回一个指向空类型的分配域起始地址的指针值；若是函数调用失败，则返回值为空。

函数 realloc()的使用语法如下：

```
指针变量=(基类型*)realloc(原存储空间首地址, 新的存储空间字节数);
```

例如：

```
int *p1,*p2;
p1=(int*)malloc(sizeof(int)*2);
p2=(int*)realloc(p1,sizeof(int)*4);
```

表示通过函数 realloc()对 p1 所指向的空间大小进行扩充，并将改变之后的内存空间首地址

赋予指针变量 p2。

☆**大牛提醒**☆

使用函数 realloc()设定的 size 大小为任意值，也就是说可以比原存储空间大，也可以比原存储空间小。

【例 10.17】编写程序，先使用函数 malloc()分配动态内存空间，然后使用函数 realloc()将该内存空间进行扩充，分别输出扩充前以及扩充后的动态内存的首地址（源代码\ch10\10.17.txt）。

```
#include <stdio.h>
/* 添加头文件 */
#include <stdlib.h>
int main()
{
  int *p1,*p2;
  int a;
  /* 调用函数 malloc()动态分配内存并将首地址赋予 p1 */
  p1=(int*)malloc(sizeof(int)*2);
  if(p1)
  {
    printf("内存分配在：%p1\n",p1);
  }
  else
  {
    printf("内存不足！\n");
    exit(0);
  }
  /* 调用函数 realloc()将 p1 中数据大小进行改变，并将改变后的内存首地址赋予 p2 */
  p2=(int*)realloc(p1,sizeof(int)*4);
  if(p2)
  {
    printf("内存重新分配在：%p2\n",p2);
  }
  else
  {
    printf("没有足够内存！\n");
    exit(0);
  }
  return 0;
}
```

程序运行结果如图 10-17 所示。例 10.17 代码中，首先添加头文件 stdlib.h，接着在主函数中定义两个指针变量 p1 和 p2，调用函数 malloc()动态分配内存并将首地址赋予指针变量 p1，若是调用成功则输出该内存的首地址，然后调用函数 realloc()将 p1 中内存大小进行改变，并将改变后的内存首地址赋予指针变量 p2，若是调用成功则输出该内存的首地址。

图 10-17　例 10.17 的程序运行结果

10.6.4　函数 free()

函数 free()定义在头文件 stdlib.h 中，使用时需要添加此头文件，该函数的原型为：

```
void free(void *p);
```

它的作用是释放指针变量 p 所指的内存区，将该存储空间返还给系统，使得其他变量能够

使用此存储空间。该函数没有任何返回值。

函数 free()的使用语法如下：

```
free(指针变量);
```

例如：

```
int *p;
p=(int*)malloc(sizeof(p));
free(p);
```

表示使用函数 free()将函数 malloc()分配的动态内存空间释放。

【例 10.18】编写程序，使用函数 free()将事先分配好的动态内存空间进行释放，输出释放前后该内存中存放的数据（源代码\ch10\10.18.txt）。

```
#include <stdio.h>
/* 添加头文件 */
#include <stdlib.h>
#include <string.h>
int main()
{
  /* 定义字符指针 */
  char *str;
  /* 调用函数 malloc()分配动态内存 */
  str=(char*)malloc(10);
  /* 调用函数 strcpy_s()将字符串赋给 str */
  strcpy_s(str,10,"Apple");
  printf("字符串为： %s\n", str);
  /* 调用函数 free()释放内存空间 */
  free(str);
  printf("释放后字符串为： %s\n", str);
  return 0;
}
```

程序运行结果如图 10-18 所示。本实例代码中，首先添加头文件“stdlib.h”以及“string.h”，接着在主函数中定义字符指针 str，调用函数 malloc()分配动态 10 字节大小的内存空间并将首地址赋予 str，然后调用函数 strcpy_s()将字符串存放到该内存空间内，并输出该字符串，接着使用函数 free()将此内存空间释放，再次输出该字符串为乱码，原因是内存已经被释放，其中存放的数据就不存在了。

图 10-18　例 10.18 的程序运行结果


微视频

10.7　其他常用函数的应用

除了以上介绍的数学函数、字符串函数以及字符函数之外，还有一些比较常用的函数，如随机函数、结束程序函数、快速排序函数等，本节将对这些函数进行详细讲解。

10.7.1　随机函数

随机函数 rand()用于产生从 0～32767 的随机数，它的使用语法如下：

```
int r;
r=rand();
```

表示生成一个随机数并赋予变量 r。

【例 10.19】编写程序，使用随机函数 rand()输出一个随机数（源代码\ch10\10.19.txt）。

```
#include <stdio.h>
/* 使用随机数函数添加头文件 */
#include <stdlib.h>
int main()
{
  int r;
  /* 随机函数 */
  r=rand();
  printf("%d\n", r);
  return 0;
}
```

程序运行结果如图 10-19 所示。本实例代码中，首先添加头文件“stdlib.h”，然后定义一个整型变量 r，使用随机函数 rand()产生一个随机数赋予 r，最后输出。

图 10-19　例 10.19 的程序运行结果

☆**大牛提醒**☆

通过使用随机函数 rand()，可以发现每次运行程序所产生的随机数都是一样的，这是因为随机数在 C 语言中采用的是固定序列，每次运行程序取的是同一个数。

为了每次产生不同的随机数，可以使用函数 srand()，该函数能够产生随机数的起始发生数据。

【例 10.20】编写程序，使用随机函数 srand()与 rand()相结合的形式产生不同随机数（源代码\ch10\10.20.txt）。

```
#include <stdio.h>
/* 添加相应头文件 */
#include <stdlib.h>
#include <time.h>
int main(void)
{
  int i;
  time_t t;
  /* 使用随机函数与时间函数相结合 */
  srand((unsigned) time(&t));
  printf("随机产生 0-99 的随机数: \n");
  for (i=0; i<5; i++)
  {
    printf("%d\n", rand()%100);
  }
  return 0;
}
```

程序运行结果如图 10-20 所示。

图 10-20　例 10.20 的程序运行结果

10.7.2 结束程序函数

结束程序函数 exit()可将当前运行程序结束，返回值将被忽略，其中 exit(0)表示正常退出，括号内数字不为 0 则表示异常退出。如果使用结束程序函数 exit()，需要包含<stdlib.h>头文件，语法格式如下：

```
void exit(int retval);
```

【例 10.21】编写程序，使用结束程序函数 exit()结束程序运行（源代码\ch10\10.21.txt）。

```
#include <stdio.h>      /*包含标准输入输出头文件*/
#include <stdlib.h>     /*包含转换和存储头文件*/
int main()
{
  int i;
  for(i=0;i<10;i++)
  {
    if(i==5) exit(0);
    else
    {
      printf("%d",i);
      getchar();         /*等待键入字符*/
    }
  }
  return 0;
}
```

当 i 值为 5 时，执行结束程序函数 exit()，终止程序，结束程序函数 exit()的返回值将被忽略。经过编译、连接、运行程序，输出结果如图 10-21 所示。

图 10-21 例 10.21 的程序运行结果

10.7.3 快速排序函数

快速排序函数 qsort()包含在<stdlib.h>头文件中，此函数根据给出的比较条件进行快速排序，通过指针移动实现排序。排序之后的结果仍然放在原数组中。使用快速排序函数必须自己写一个比较函数。语法格式如下：

```
void qsort ( void * base,int n, int size, int ( * fcmp ) ( const void *, const void * ) );
```

【例 10.22】编写程序，使用快速排序函数对数组进行排序（源代码\ch10\10.22.txt）。

```
#include <stdio.h>                          /*包含标准输入输出头文件*/
#include <stdlib.h>                         /*包含转换和存储头文件*/
#include <string.h>                         /*包含字符串处理头文件*/
char stringlist[5][6] = { "boy", "girl", "man", "woman", "human" };
int intlist[5]= { 3, 2,5,1,4 };
int sort_stringfun( const void *a, const void *b);
int sort_intfun( const void *a, const void *b);
int main(void)
{
  int x;
  printf("字符串排序: \n");
  qsort((void *)stringlist, 5, sizeof(stringlist[0]), sort_stringfun); /*调用快速排序函数*/
    for (x = 0; x < 5; x++)
```

```
    printf("%s\n", stringlist[x]);
  printf("整数排序: \n");
  qsort((void *)intlist, 5, sizeof(intlist[0]), sort_intfun);
  for (x = 0; x < 5; x++)
    printf("%d\n", intlist[x]);
  return 0;
}
int sort_stringfun( const void *a, const void *b)
{
  return( strcmp((const char *)a,(const char *)b) );
}
int sort_intfun( const void *a, const void *b)
{
  return *(int *)a - *(int *)b;
}
```

程序运行结果如图 10-22 所示。

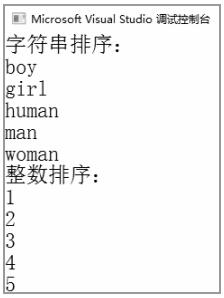

图 10-22　例 10.22 的程序运行结果

10.8　新手疑难问题解答

问题 1：主函数是否有参数，有没有参数会不会有什么影响？

解答：主函数也会有自己的参数。语法格式如下：

```
int main( int argc, char *argv[] )
```

其中，argc 和 argv 是主函数的形式参数。这两个形式参数的类型是系统规定的。如果主函数要带参数，就是这两个类型的参数；否则主函数就没有参数。

变量名称 argc 和 argv 是常规的名称，也可以换成其他名称。主函数收到参数后就会做自己的事。那么，实际参数是如何传递给主函数的 argc 和 argv 的呢？我们知道，C 程序在编译和链接后，都生成一个 exe 文件，执行该 exe 文件时，可以直接执行；也可以在命令行下带参数执行，命令行执行的形式为：可执行文件名称 参数 1 参数 2……参数 n。可执行文件名称和参数、参数之间均使用空格隔开。

如果按照这种方法执行，命令行字符串将作为实际参数传递给主函数。具体为：

（1）可执行文件名称和所有参数的个数之和传递给 argc；

（2）可执行文件名称（包括路径名称）作为一个字符串，首地址被赋给 argv[0]，参数 1 也作为一个字符串，首地址被赋给 argv[1]……以此类推。

问题 2：结束程序函数和 return 有什么区别？

解答：在最初调用的主函数中使用 return 和 exit()的效果相同。但要注意这里所说的是“最初调用”。如果主函数在一个递归程序中，exit()仍然会终止程序；但 return 将控制权移交给递

归的前一级，直到最初的那一级，此时 return 才会终止程序。return 和 exit()的另一个区别在于，即使在除主函数之外的函数中调用 exit()，它也将终止程序。


解题思路

10.9 实战训练

实战 1：使用函数输出字符所对应的进制数。

编写程序，要求输出字符 a 的十进制数、八进制数以及十六进制数，程序运行结果如图 10-23 所示。

图 10-23 实战 1 的程序运行结果

实战 2：输出三个数字组成的无重复数字三位数。

编写程序，有 1、2、3 三个数字，使用这三个数字组成无重复数字的三位数。程序运行结果如图 10-24 所示。

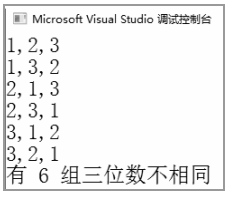

图 10-24 实战 2 的程序运行结果

实战 3：实现数字猜谜游戏。

编写程序，使用随机函数完成一个报数游戏：通过输入端输入一个 100 以内整数，并与产生的随机数进行比较，输出比较结果，直到猜中为止。程序运行结果如图 10-25 所示。

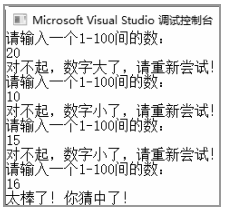

图 10-25 实战 3 的程序运行结果