11 字符串和字符串函数

内容提要

- 函数
  - gets() gets_s() fgets() puts() fputs() strcat() strncat() strcmp() strncmp() strcpy() strncpy() sprintf() strchr()
- 创建和使用字符串
- 利用 C 库里的字符串和字符串函数创建你自己的字符串函数
- 使用命令行参数

字符串表示和字符串I/O

1 字符串表示和字符串I/O

- 字符串（character string)是以空字符（\0）结尾的 char 数组
- 11.1 strings.c
- puts()函数只显示字符串，而且自动在显示的字符串末尾加上换行符
- 11.2 quotes.c

```
1.  //  strings1.c
2.  #include <stdio.h>
3.  #define MSG "I am a symbolic string constant."
4.  #define MAXLENGTH 81
5.  int main(void)
6.  {
7.      char words[MAXLENGTH] = “A string in an array.";
8.      const char * pt1 = "Something is pointing at me.";
9.      puts("Here are some strings:");
10.     puts(MSG);
11.     puts(words);
12.     puts(pt1);
13.     words[8] = 'p';
14.     puts(words);

15.     return 0;
16. }
```

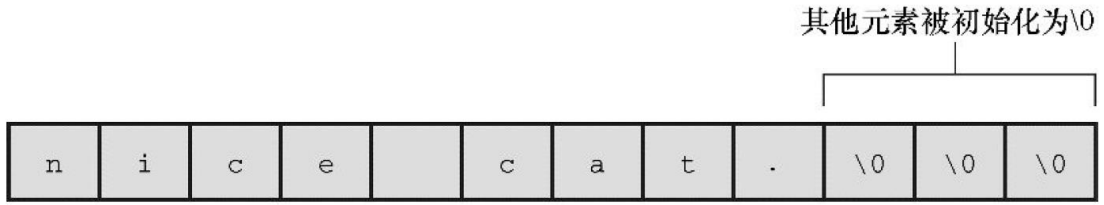
1.1 在程序中定义字符串

- 用双引号括起来的内容称为字符串字面量（string literal），也叫作字符串常量（string constant）
  - 双引号中的字符和编译器自动加入末尾的\0字符，都作为字符串存储在内存中
- char greeting[50] = "Hello, and"" how are" " you“ " today!";
- char greeting[50] = "Hello, and how are you today!";
- 字符串常量属于静态存储类别（static storage class）
  - 这说明如果在函数中使用字符串常量，该字符串只会被存储一次，在整个程序的生命期内存在，即使函数被调用多次。用双引号括起来的内容被视为指向该字符串存储位置的指针
- 程序清单11.2 strptr.c

```
1. /* strptr.c -- strings as pointers */
2. #include <stdio.h>
3. int main(void)
4. {
5.     printf("%s, %p, %c\n", "We", "are", *"space farers");
6.
7.     return 0;
8. }
```

字符串数组和初始化

- 定义字符串数组时，必须让编译器知道需要多少空间.
  - `const char m1[40] = "Limit yourself to one line's worth.";`
- 在指定数组大小时，要确保数组的元素个数至少比字符串长度多1（为了容纳空字符）。所有未被使用的元素都被自动初始化为0



`const char pets[12] = "nice cat.";`

- 省略数组初始化声明中的大小，编译器会自动计算数组的大小：
  - `const char m2[] = "If you can't think of anything, fake it.";`
- 字符数组名和其他数组名一样，是该数组首元素的地址

数组与指针

- const char * pt1 = "Something is pointing at me.";
- const char ar1[] = "Something is pointing at me.";
- 数组形式（ar1[]）在计算机的内存中分配为一个内含29个元素的数组
  - 字符串作为可执行文件的一部分存储在数据段中。当把程序载入内存时，也载入了程序中的字符串。字符串存储在静态存储区（static memory）中
  - 程序在开始运行时才会为该数组分配内存。此时将字符串拷贝到数组中。此时字符串有两个副本。一个是在静态内存中的字符串字面量，另一个是存储在ar1数组中的字符串
- 指针形式（*pt1）使得编译器为字符串在静态存储区预留29个元素的空间
  - 另外，一旦开始执行程序，它会为指针变量pt1留出一个存储位置，并把字符串的地址存储在指针变量中
- 程序清单11.3 addresses.c

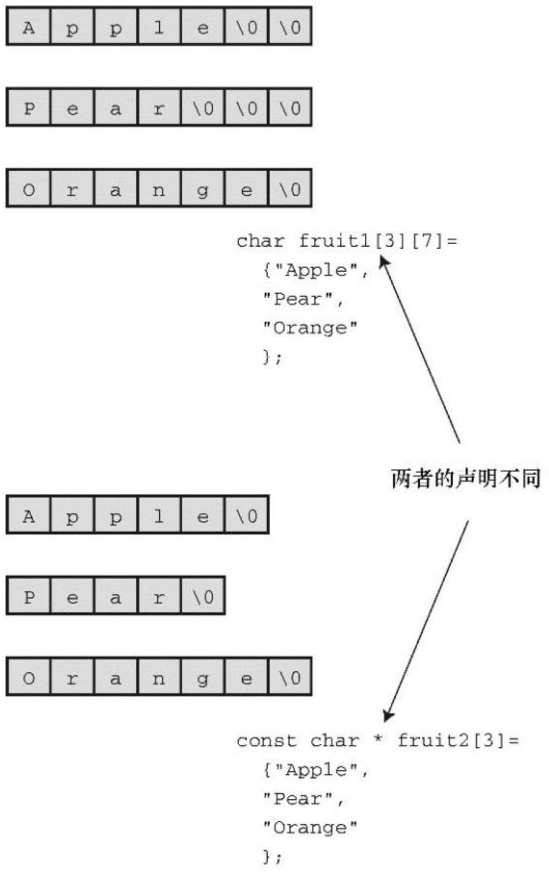
```
1.  //  addresses.c  -- addresses of strings
2.  #define MSG "I'm special."

3.  #include <stdio.h>
4.  int main()
5.  {
6.      char ar[] = MSG;
7.      const char *pt = MSG;
8.      printf("address of \"I'm \": %p \n", "I'm
    special");
9.      printf("             address ar: %p\n", ar);
10.     printf("             address pt: %p\n", pt);
11.     printf("          address of MSG: %p\n", MSG);
12.     printf("address of \"I'm \": %p \n", "I'm
    special");
13.     return 0;
14. }
```

数组和指针的差别

- `char heart[ ] = "I love tillie!.";`
- `const char *hear = "I love Millie!.";`
- 两者主要的区别：数组名heart是常量，而指针名head是变量
  - 首先，两者都可以使用数组表示法
  - 其次，两者都能进行指针加法操作
  - 但是，只有指针表示法可以进行递增操【数组名是指针常量】
- 数组的元素是变量（除非数组被声明为const），但是数组名不是变量

字符串数组

- 字符串数组(是数组，元素为字符串)
  - 通过数组下标访问多个不同的字符串
- 程序清单11.4 arrchar.c
- mytalents：5个指针
- Ourtalents：5个数组




```
1.  #define SLEN 40
2.  #define LIM 2
3.  int main(void){
4.      const char *mytalents[LIM] = {
5.          "Adding numbers swiftly",
6.          "Multiplying accurately"};
7.      char yourtalents[LIM][SLEN] = {
8.          "Walking in a straight line",
9.          "Sleeping"};
10.     for (int i = 0; i < LIM; i++)
11.         printf("%-36s %-25s\n", mytalents[i],
    yourtalents[i]);
12.     printf("\nsizeof mytalents: %zd, sizeof
    yourtalents: %zd\n", sizeof(mytalents),
    sizeof(yourtalents));
13.
14.     return 0;
15. }
```

1.2 指针和字符串

- 11.3 p_and_s.c
- 字符串的绝大多数操作都是通过指针完成

```c
/* p_and_s.c -- pointers and strings */
#include <stdio.h>
int main(void)
{
    const char * mesg = "Don't be a fool!";
    const char * copy;

    copy = mesg;
    printf("%s\n", copy);
    printf("mesg = %s; &mesg = %p; value = %p\n",
           mesg, &mesg, mesg);
    printf("copy = %s; &copy = %p; value = %p\n",
           copy, &copy, copy);

    return 0;
}
```

字符串输入

2 字符串输入

- 如果想把一个字符串读到程序中
  - 必须首先预留存储字符串的空间
  - 然后使用输入函数来获取这个字符串

2.1 分配空间

- 不要指望计算机读的时候会先计算字符串的长度，然后为字符串分配空间
- 最简单的方法，在声明时显式指明数组的大小
- 为字符串分配内存后，便可读入字符串
- C库提供了许多读取字符串的函数
  - scanf()、gets()和fgets()

2.2 gets()函数

- 程序清单11.6 getsputs.c
- gets()
  - 读取整行输入，直至遇到换行符，然后丢弃换行符，存储其余字符，并在这些字符的末尾添加一个空字符使其成为一个C字符串
- puts()
  - 显示字符串，并在末尾添加换行符
- 如果输入的字符串过长，会导致缓冲区溢出（buffer overflow），即多余的字符超出了指定的目标空间

```
1.  /*  getsputs.c  -- using gets() and puts() */
2.  #include <stdio.h>
3.  #define STLEN 81
4.  int main(void)
5.  {
6.      char words[STLEN];
7.
8.      puts("Enter a string, please.");
9.      gets(words);  // typical use
10.     printf("Your string twice:\n");
11.     printf("%s\n", words);
12.     puts(words);
13.     puts("Done.");
14.
15.     return 0;
16. }
```

2.3 fgets()函数

- 程序清单11.7 fgets1.c
- ptr = fgets(name, n, stdin);
  - 第2个参数指明了读入字符的最大数量
    - fgets()读入n-1个字符，或者读到遇到的第一个换行符为止
  - 如果读到一个换行符，会把它存储在字符串中
    - 与fgets()不同，gets()会丢弃换行符
  - 第3个参数指明要读入的文件
    - 从键盘输入数据，则以stdin（标准输入）,定义在stdio.h中
- 程序清单11.8 fgets2.c
- 程序清单11.9 fgets3.c
- 空字符（或'\0'），用于标记C字符串末尾的字符，对应字符编码是0。
- 空指针（或NULL）不会与任何数据的有效地址对应
  - 通常函数使用NULL作为返回值表示某些特殊情况发生。如遇到文件结尾或未能按预期执行

```
1.  #include <stdio.h>
2.  #define STLEN 14
3.  int main(void)
4.  {
5.      char words[STLEN];
6.      puts("Enter a string, please.");
7.      fgets(words, STLEN, stdin);
8.      printf(" (puts(), then fputs()):\n");
9.      puts(words);
10.     fputs(words, stdout);
11.     puts("Enter another string, please.");
12.     fgets(words, STLEN, stdin);
13.     printf("twice (puts(), then fputs()):\n");
14.     puts(words);
15.     fputs(words, stdout);
16.     puts("Done.");
17.     return 0;
18. }
```

gets_s()函数

- C11新增gets_s(words, STLEN)
- gets_s()只从标准输入中读取数据，所以不需要第3个参数
- 如果gets_s()读到换行符，会丢弃它而不是存储它
- 如果gets_s()读到最大字符数都没有读到换行符，会执行以下几步
  - 首先把目标数组中的首字符设置为空字符，读取并丢弃随后的输入直至读到换行符或文件结尾，然后返回空指针
  - 接着，调用依赖实现的“处理函数”（或你选择的其他函数），可能会中止或退出程序

2.4 scanf()

- 11.7 scan_str.c
- scanf()和gets()或fgets()的区别在于它们如何确定字符串的末尾
  - scanf()更像是“获取单词”，而不是“获取字符串”
  - 如果预留的存储区能装下输入行，gets()和fgets()会读取第1个换行符之前所有的字符
  - scanf()函数有两种方法确定输入结束。无论哪种方法，都从第1个非空白字符作为字符串的开始

```c
/* scan_str.c -- using scanf() */
#include <stdio.h>
int main(void)
{
    char name1[11], name2[11];
    int count;

    printf("Please enter 2 names.\n");
    count = scanf("%5s %10s",name1, name2);
    printf("I read the %d names %s and %s.\n",
           count, name1, name2);

    return 0;
}
```

字符串输出

3 字符串输出

- puts()
- fputs()
- printf()

3.1 puts()函数

- 11.18 put_out.c
- puts()函数，字符串的地址作为参数
  - puts()显示字符串时自动在其后添加换行符
- 程序清单11.13 nono.c
  - 不要效仿这个程序

```c
/* put_out.c -- using puts() */
#include <stdio.h>
#define DEF "I am a #defined string."
int main(void)
{
    char str1[80] = "An array was initialized to me.";
    const char * str2 = "A pointer was initialized.";

    puts("I'm an argument to puts().");
    puts(DEF);
    puts(str1);
    puts(str2);
    puts(&str1[5]);
    puts(str2+4);

    return 0;
}
```

3.2 fputs()函数

- fputs(line, stdout);
- fputs()函数是puts()针对文件定制的版本

3.3 printf()函数

- printf()函数可以格式化不同的数据类型

自定义字符串输入/输出函数

4 自定义字符串输入/输出函数

- 在getchar()和 putchar()的基础上建立自己的函数
- 11.10　　put1.c
- 11.11　　put2.c
- 程序清单11.16　put_put.c

```
1.  void put1(const char * string){
2.      while (*string)  /* same as *string != '\0' */
3.          putchar(*string++);
4.  }
5.  int put2(const char * string){
6.      int count = 0;
7.      while (*string){
8.          putchar(*string++);
9.          count++;
10.     }
11.     return(count);
12. }
13. int main(void){
14.     put1("If I'd as much money");
15.     put1(" as I could spend,\n");
16.     printf("I count %d characters.\n", put2("I never
    would cry old."));
17.     return 0;
18. }
```

字符串函数

5 字符串函数

- C库提供了多个处理字符串的函数，ANSI C把这些函数的原型放在string.h头文件中
  - 其中最常用的函数有strlen()、strcat()、strcmp()、strncmp()、strcpy()和strncpy()
- 另外，还有sprintf()函数，其原型在stdio.h头文件中

字符串函数

- strlen()函数
  - 用于统计字符串的长度
  - 程序清单11.17　test_fit.c
- strcat()
  - 用于拼接字符串
  - 程序清单11.18　str_cat.c
- strncat()函数
  - 程序清单11.19　join_chk.c
- strcmp()函数
  - 字符串作比较
  - 程序清单11.20　nogo.c
  - 程序清单11.21　compare.c
  - 按机器排序序列（machine collating sequence）进行比较，即根据字符的数值进行比较（通常使用ASCII值）
- strncmp()函数
  - 可以比较到字符不同的地方，也可以只比较第3个参数指定的字符数
  - 程序清单11.24　starsrch.c
- strcpy()和strncpy()
  - 拷贝整个字符串
  - 程序清单11.25　copy1.c
  - 程序清单11.26　copy2.c
  - 程序清单11.27　copy3.c
- sprintf()函数
  - 和printf()类似，但是它是把数据写入字符串，而不是打印在显示器上
  - 程序清单11.28　format.c

字符串例子：　字符串排序

6 字符串例子： 字符串排序

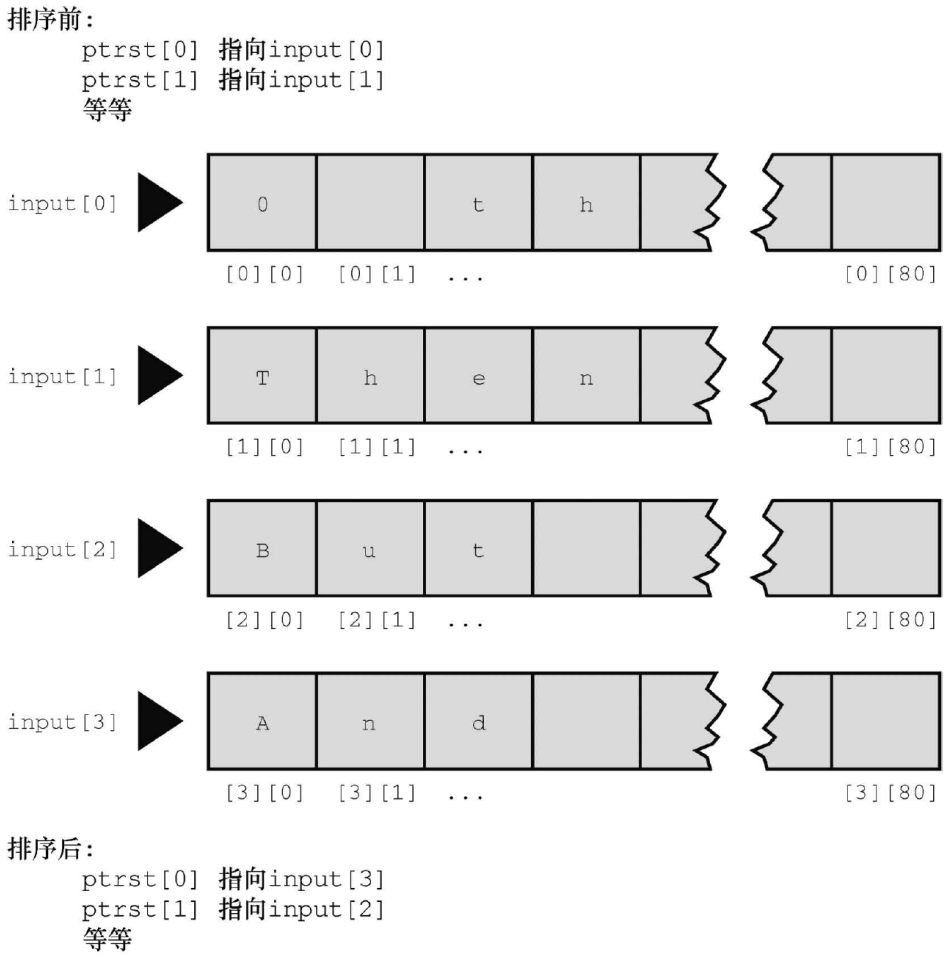
11.25 sort_str.c

```
1.  #define SIZE 81 //string length limit, including \0
2.  #define LIM 20 //maximum number of lines to be read
3.  #define HALT "" //null string to stop input
4.  void strsort(char *strings[], int num);//sort
5.  char * s_gets(char * st, int n);

6.  int main(void){
7.      char input[LIM][SIZE]; // array to store input
8.      char *ptstr[LIM]; // array of pointer variables
9.      int ct = 0;       // input count
10.     printf("Up to %d lines.\n", LIM);
11.     printf("Enter key at a line's start to stop.\n");
12.     while (ct < LIM && s_gets(input[ct], SIZE) != NULL
    && input[ct][0] != '\0'){
13.         ptstr[ct] = input[ct];  //ptrs to strings
14.         ct++;
15.     }
16.     strsort(ptstr, ct); /* string sorter */
17.     for (int k = 0; k < ct; k++) puts(ptstr[k]);
18.     return 0;
19. }
```

```
20. /* string-pointer-sorting function */
21. void strsort(char *strings[], int num){
22.     for (int top = 0; top < num-1; top++)
23.         for (int seek = top + 1; seek < num; seek++)
24.             if (strcmp(strings[top],strings[seek]) > 0){
25.                 char *temp = strings[top];
26.                 strings[top] = strings[seek];
27.                 strings[seek] = temp;
28.             }
29. }
30. char * s_gets(char * st, int n){
31.     int i = 0;
32.     char * ret_val = fgets(st, n, stdin);
33.     if (ret_val) {
34.         while (st[i] != '\n' && st[i] != '\0') i++;
35.         if (st[i] == '\n') st[i] = '\0';
36.         else // must have words[i] == '\0'
37.             while (getchar() != '\n')  continue;
38.     }
39.     return ret_val;
40. }
```

6.1 排序指针而不是字符串

- 程序技巧部分在于：并不是重新安排字符串本身，而仅仅重新安排指向字符串的指针



6.2 选择排序算法

- 使用了选择排序（selection sort）算法来进行指针排序.
  - 其思想是使用一个 for 循环把每个元素轮流与第一个元素比较. 如果被比较元素在顺序上先于当前第一个元素，程序就交换这二者
- for n = 首元素至 n = 倒数第2个元素，
  - 找出剩余元素中的最大值，并将其放在第n个元素中

ctype.h字符函数和字符串

7 ctype.h字符函数和字符串

- 11.26 mod_str.c
- ctype.h与字符相关的函数，处理字符串中的字符
- ctype.h中的函数通常作为宏（macro）来实现

```c
void ToUpper(char * str){
    while (*str){
        *str = toupper(*str);
        str++;
    }
}
int PunctCount(const char * str){
    int ct = 0;
    while (*str){
        if (ispunct(*str))
            ct++;
        str++;
    }
    return ct;
}
```

```c
#include <stdio.h>
#include <string.h>
#include <ctype.h>
#define LIMIT 81
void ToUpper(char *);
int PunctCount(const char *);
int main(void){
    char line[LIMIT];
    puts("Please enter a line:");
    fgets(line, LIMIT, stdin);
    char * find = strchr(line, '\n'); // look for newline
    if (find)           // if the address is not NULL,
        *find = '\0';   // place a null character there
    ToUpper(line);
    puts(line);
    printf("That line has %d ….\n", PunctCount(line));

    return 0;
}
```

命令行参数

8 命令行参数

- 命令行（command line）是一个命令行环境下，用户输入的用于运行程序的行
  - 命令行参数（command-line argument）是同一行的附加项
- 11.27 repeat.c
- C编译器允许main()没有参数或者有两个参数（一些实现允许main()有更多参数，属于对标准的扩展）
- main()有两个参数时
  - 第1个参数是命令行中的字符串数量
  - 过去这个int类型的参数被称为argc（表示参数计数(argument count)）

```c
/* repeat.c -- main() with arguments */
#include <stdio.h>
int main(int argc, char *argv[])
{
    int count;

    printf("%d arguments:\n", argc - 1);
    for (count = 1; count < argc; count++)
        printf("%d: %s\n", count, argv[count]);
    printf("\n");

    return 0;
}
```

把字符串转换为数字

9 把字符串转换为数字

- 11.28 hello.c
- 11.29 strcnvt.c

- itoa() 和 ftoa()
- strtol
- 用atoi()函数（用于把字母数字转换成整数）
- 许多实现使用itoa()和ftoa()函数分别把整数和浮点数转换成字符串

关键概念

10 关键概念

- 字符串，无论是由字符数组、指针还是字符串常量标识，都存储为包含字符编码的一系列字节，并以空字符串结尾
- C提供库函数处理字符串，查找字符串并分析它们
- 使用strcmp()来代替关系运算符，当比较字符串时，应该使用strcpy()或strncpy()代替赋值运算符把字符串赋给字符数组

11 总结