Character Array and Strings

Lecture 11

Strings and Pointers

- In C, a string is a one-dimensional array of type char.
  - A character in a string can be accessed either:
    - As an element of an array, or
    - By making use of a pointer to char.
  - The type pointer to char is conceptually a string.
  - A unique aspect of a string is the use of the character value '\0' to terminate the string.

The End-of-String Sentinel \0

- In C, a string is terminated by the end-of-string sentinel \0 (the null character).
  - A constant string such as "abc" is stored in memory as four characters, the last one being \0.
  - The length of the string "abc" is 3, but size of the array that holds the string must be at least 4.

Declaring a String

- To declare an array of characters that will be used to store a string, you must allocate enough storage to:
  - Hold the maximum number of characters that will be stored in the string.
  - Plus one byte (character) to hold the null character.

- Example: Our university name Jahangirnagar University, has 24 characters (including the space), but storing the name in as a string requires an array of 25 characters.
  - So I would declare

```
char universityName[25];
```

Inputting a String from the Keyboard

- How scanf() works with a string:

```
#define MAX_SIZE 100
 int main(void)
 {
     char w[MAX_SIZE];
     . . .
     scanf("%s", w);
}
```

  - First scanf() positions the input stream to an initial nonwhite space character.
  - Second, nonwhite space characters are read in and placed in memory beginning at the base address of w.
  - The process stops when a white space character or EOF is encountered.
    - At that point a null character is placed in memory (by the C system) to end the string.

Example: Inputting a String Using scanf()

- Suppose you want a user to input their first and last names on a single line:

```
char fname[20], lname[20];
. . .
printf("Input your first and last names:");
scanf("%s%s", fname, lname);
```

  - When the user types their first and last name, then presses ENTER, scanf() will:
    - skip any spaces the user input before their first name
    - start reading characters into fname
    - stop when the space between names is reached
    - place a \0 into the fname array
    - skip the space(s) and start reading characters into lname
    - stop reading characters into lname when \n is reached
    - place a \0 into lname
  - Note that the spaces and \n are removed from the input buffer but not stored in the strings.

Initialization of Strings

- Arrays, including character arrays, can be initialized.
- However, the compiler allows a special syntax for initialization of strings:

```
char s[] = "abc";
```

is equivalent to

```
char s[] = {'a', 'b', 'c', '\0'};
```

When the string-specific first form is used, the size of the string is one more than the string length in order to terminate the string with the \0 null character.

Initializing a Pointer-to-Character with a String

- A pointer-to-character can be initialized with a constant string, but the interpretation is different.

  char *p = "abc";

  - When a character pointer is initialized with a constant string, the pointer simply stores the address where the constant string is stored in memory.
  - When an array is initialized using a constant string, the characters themselves (including the null character) are stored in the array.
    - The string constant itself remains in another location in memory.

Character Handling Library

- Character handling library
  - Includes functions to perform useful tests and manipulations of character data
  - Each function receives a character (an `int`) or `EOF` as an argument
- The following slides contain a table of all the functions in `<ctype.h>`

Prototype	Function description
`int isdigit( int c );`	Returns a true value if `c` is a digit and 0 (false) otherwise.
`int isalpha( int c );`	Returns a true value if `c` is a letter and 0 otherwise.
`int isalnum( int c );`	Returns a true value if `c` is a digit or a letter and 0 otherwise.
`int isxdigit( int c );`	Returns a true value if `c` is a hexadecimal digit character and 0 otherwise. (See Appendix E, Number Systems, for a detailed explanation of binary numbers, octal numbers, decimal numbers and hexadecimal numbers.)
`int islower( int c );`	Returns a true value if `c` is a lowercase letter and 0 otherwise.
`int isupper( int c );`	Returns a true value if `c` is an uppercase letter and 0 otherwise.
`int tolower( int c );`	If `c` is an uppercase letter, `tolower` returns `c` as a lowercase letter. Otherwise, `tolower` returns the argument unchanged.

Fig. 8.1 | Character-handling library functions. (Part 1 of 2.)

Prototype	Function description
`int toupper( int c );`	If `c` is a lowercase letter, `toupper` returns `c` as an uppercase letter. Otherwise, `toupper` returns the argument unchanged.
`int isspace( int c );`	Returns a true value if `c` is a white-space character—newline (`'\n'`), space (`' '`), form feed (`'\f'`), carriage return (`'\r'`), horizontal tab (`'\t'`) or vertical tab (`'\v'`)—and 0 otherwise.
`int iscntrl( int c );`	Returns a true value if `c` is a control character and 0 otherwise.
`int ispunct( int c );`	Returns a true value if `c` is a printing character other than a space, a digit, or a letter and returns 0 otherwise.
`int isprint( int c );`	Returns a true value if `c` is a printing character including a space (`' '`) and returns 0 otherwise.
`int isgraph( int c );`	Returns a true value if `c` is a printing character other than a space (`' '`) and returns 0 otherwise.

Fig. 8.1 | Character-handling library functions. (Part 2 of 2.)

Outline

fig08_02.c

(1 of 3)

```c
/* Fig. 8.2: fig08_02.c
   Using functions isdigit, isalpha, isalnum, and isxdigit */
#include <stdio.h>
#include <ctype.h>

int main( void )
{
   printf( "%s\n%s%s\n%s%s\n\n", "According to isdigit: ",
      isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
      isdigit( '#' ) ? "# is a " : "# is not a ", "digit" );

   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
      "According to isalpha:",
      isalpha( 'A' ) ? "A is a " : "A is not a ", "letter",
      isalpha( 'b' ) ? "b is a " : "b is not a ", "letter",
      isalpha( '&' ) ? "& is a " : "& is not a ", "letter",
      isalpha( '4' ) ? "4 is a " : "4 is not a ", "letter" );

```

`isdigit` tests if a character is a decimal digit

`isalpha` tests if a character is a letter

```
19    printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
20        "According to isalnum:",
21        isalnum( 'A' ) ? "A is a " : "A is not a ",
22        "digit or a letter",
23        isalnum( '8' ) ? "8 is a " : "8 is not a ",
24        "digit or a letter",
25        isalnum( '#' ) ? "# is a " : "# is not a ",
26        "digit or a letter" );
27
28    printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
29        "According to isxdigit:",
30        isxdigit( 'F' ) ? "F is a " : "F is not a ",
31        "hexadecimal digit",
32        isxdigit( 'J' ) ? "J is a " : "J is not a ",
33        "hexadecimal digit",
34        isxdigit( '7' ) ? "7 is a " : "7 is not a ",
35        "hexadecimal digit",
36        isxdigit( '$' ) ? "$ is a " : "$ is not a ",
```


fig08_02.c

(2 of 3)

isdigit tests if a character is a decimal digit or a letter

isxdigit tests if a character is a hexadecimal digit

```
37          "hexadecimal digit",
38          isxdigit( 'f' ) ? "f is a " : "f is not a ",
39          "hexadecimal digit" );
40
41       return 0; /* indicates successful termination */
42
43 } /* end main */
```

```
According to isdigit:
8 is a digit
# is not a digit

According to isalpha:
A is a letter
b is a letter
& is not a letter
4 is not a letter

According to isalnum:
A is a digit or a letter
8 is a digit or a letter
# is not a digit or a letter

According to isxdigit:
F is a hexadecimal digit
J is not a hexadecimal digit
7 is a hexadecimal digit
$ is not a hexadecimal digit
f is a hexadecimal digit
```


fig08_02.c

(3 of 3)

Outline

fig08_03.c

(1 of 2)

```c
/* Fig. 8.3: fig08_03.c
   Using functions islower, isupper, tolower, toupper */
#include <stdio.h>
#include <ctype.h>

int main( void )
{
   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
           "According to islower:",
           islower( 'p' ) ? "p is a " : "p is not a ",
           "lowercase letter",
           islower( 'P' ) ? "P is a " : "P is not a ",
           "lowercase letter",
           islower( '5' ) ? "5 is a " : "5 is not a ",
           "lowercase letter",
           islower( '!' ) ? "! is a " : "! is not a ",
           "lowercase letter" );

   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n\n",
           "According to isupper:",
           isupper( 'D' ) ? "D is an " : "D is not an ",
           "uppercase letter",
           isupper( 'd' ) ? "d is an " : "d is not an ",
           "uppercase letter",
           isupper( '8' ) ? "8 is an " : "8 is not an ",
           "uppercase letter",
           isupper( '$' ) ? "$ is an " : "$ is not an ",
           "uppercase letter" );

```

islower tests if a character is a lowercase letter

isupper tests if a character is an uppercase letter

Outline

toupper and **tolower** convert letters to upper or lower case

Fig08_03.c (2 of 2)

```
30    printf( "%s%c\n%s%c\n%s%c\n%s%c\n",
31            "u converted to uppercase is ", toupper( 'u' ),
32            "7 converted to uppercase is ", toupper( '7' ),
33            "$ converted to uppercase is ", toupper( '$' ),
34            "L converted to lowercase is ", tolower( 'L' ) );
35
36    return 0; /* indicates successful termination */
37
38 } /* end main */
```

```
According to islower:
p is a lowercase letter
P is not a lowercase letter
5 is not a lowercase letter
! is not a lowercase letter

According to isupper:
D is an uppercase letter
d is not an uppercase letter
8 is not an uppercase letter
$ is not an uppercase letter

u converted to uppercase is U
7 converted to uppercase is 7
$ converted to uppercase is $
L converted to lowercase is l
```

Outline

fig08_04.c

(1 of 3)

```c
/* Fig. 8.4: fig08_04.c
   Using functions isspace, iscntrl, ispunct, isprint, isgraph */
#include <stdio.h>
#include <ctype.h>

int main( void )
{
   printf( "%s\n%s%s%s\n%s%s%s\n%s%s\n\n",
      "According to isspace:",
      "Newline", isspace( '\n' ) ? " is a " : " is not a ",
      "whitespace character", "Horizontal tab",
      isspace( '\t' ) ? " is a " : " is not a ",
      "whitespace character",
      isspace( '%' ) ? "% is a " : "% is not a ",
      "whitespace character" );

   printf( "%s\n%s%s%s\n%s%s\n\n", "According to iscntrl:",
      "Newline", iscntrl( '\n' ) ? " is a " : " is not a ",
      "control character", iscntrl( '$' ) ? "$ is a " :
      "$ is not a ", "control character" );
```

`isspace` tests if a character is a whitespace character

`iscntrl` tests if a character is a control character

Outline

fig08_04.c

(2 of 3)

```
21
22    printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
23        "According to ispunct:",
24        ispunct( ';' ) ? "; is a " : "; is not a ",
25        "punctuation character",
26        ispunct( 'Y' ) ? "Y is a " : "Y is not a ",
27        "punctuation character",
28        ispunct( '#' ) ? "# is a " : "# is not a ",
29        "punctuation character" );
30
31    printf( "%s\n%s%s\n%s%s%s\n\n", "According to isprint:",
32        isprint( '$' ) ? "$ is a " : "$ is not a ",
33        "printing character",
34        "Alert", isprint( '\a' ) ? " is a " : " is not a ",
35        "printing character" );
36
```

`ispunct` tests if a character is a punctuation character

`isprint` tests if a character is a printing character

```
37    printf( "%s\n%s%s\n%s%s%s\n",  "According to isgraph:",
38        isgraph( 'Q' ) ? "Q is a " : "Q is not a ",
39        "printing character other than a space",
40        "Space", isgraph( ' ' ) ? " is a " : " is not a ",
41        "printing character other than a space" );
42
43    return 0; /* indicates successful termination */
44
45 } /* end main */
```

```
According to isspace:
Newline is a whitespace character
Horizontal tab is a whitespace character
% is not a whitespace character

According to iscntrl:
Newline is a control character
$ is not a control character

According to ispunct:
; is a punctuation character
Y is not a punctuation character
# is a punctuation character

According to isprint:
$ is a printing character
Alert is not a printing character

According to isgraph:
Q is a printing character other than a space
Space is not a printing character other than a space
```


`isgraph` tests if a character is a printing character that is not a space

String Handling Functions

- The standard library contains many useful string handling functions.
  - All require that strings passed as arguments be null-terminated.
  - All return either an integer value or a pointer to char.
- The prototypes for the string functions are in string.h.

Some String Handling Functions

- char *strcat(char *s1, const char *s2);
  - strcat() concatenates s2 onto the end of s1
  - You must ensure that s1 allocated enough space to hold the result.
  - The string s1 is returned.
- int strcmp(const char *s1, const char *s2);
  - An integer is returned that is less than, equal to, or greater than zero, depending on whether s1 is lexicographically less than, equal to, or greater than s2.

Two More String Handling Functions

- strcpy(char *s1, const char *s2);
  - The string s2 is copied into s1 until the \0 is moved.
  - Whatever exists in s1 is overwritten.
  - s1 must have enough space to hold the result.
  - s1 is returned.
- unsigned strlen(const char *s);
  - A count of the number of characters before \0 is returned.

Examples Using String Handling Functions

Declarations and Initializations

```
char s1[] = "beautiful big sky country",
     s2[] = "how now brown cow";
```

Expression	Value
strlen(s1)	25
strlen(s2 + 8)	9
strcmp(s1, s2)	negative integer

Statements	What is Printed
printf("%s", s1 + 10);	big sky country
strcpy(s1 + 10, s2 + 8);	
strcat(s1, "s!");	
printf("%s", s1);	beautiful brown cows!

String-Conversion Functions

- Conversion functions
  - In `<stdlib.h>` (general utilities library)
- Convert strings of digits to integer and floating-point values

Function prototype	Function description
`double atof( const char *nPtr );`	Converts the string `nPtr` to `double`.
`int atoi( const char *nPtr );`	Converts the string `nPtr` to `int`.
`long atol( const char *nPtr );`	Converts the string `nPtr` to `long int`.
`double strtod( const char *nPtr, char **endPtr );`	Converts the string `nPtr` to `double`.
`long strtol( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `long`.
`unsigned long strtoul( const char *nPtr, char **endPtr, int base );`	Converts the string `nPtr` to `unsigned long`.

Fig. 8.5 | String-conversion functions of the general utilities library.

Outline

fig08_06.c

```c
/* Fig. 8.6: fig08_06.c
   Using atof */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   double d; /* variable to hold converted string */

   d = atof( "99.0" );

   printf( "%s%.3f\n%s%.3f\n",
           "The string \"99.0\" converted to double is ", d,
           "The converted value divided by 2 is ",
           d / 2.0 );

   return 0; /* indicates successful termination */

} /* end main */
```

`atof` converts a string to a **`double`**

```
The string "99.0" converted to double is 99.000
The converted value divided by 2 is 49.500
```

Outline

fig08_07.c

```c
/* Fig. 8.7: fig08_07.c
   Using atoi */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   int i; /* variable to hold converted string */

   i = atoi( "2593" );

   printf( "%s%d\n%s%d\n",
           "The string \"2593\" converted to int is ", i,
           "The converted value minus 593 is ", i - 593 );

   return 0; /* indicates successful termination */

} /* end main */
```

`atoi` converts a string to an **`int`**

```
The string "2593" converted to int is 2593
The converted value minus 593 is 2000
```

Outline

fig08_08.c

```c
/* Fig. 8.8: fig08_08.c
   Using atol */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   long l; /* variable to hold converted string */

   l = atol( "1000000" );

   printf( "%s%ld\n%s%ld\n",
           "The string \"1000000\" converted to long int is ", l,
           "The converted value divided by 2 is ", l / 2 );

   return 0; /* indicates successful termination */

} /* end main */
```

`atol` converts a string to a **`long`**

```
The string "1000000" converted to long int is 1000000
The converted value divided by 2 is 500000
```

Outline

fig08_09.c

```c
/* Fig. 8.9: fig08_09.c
   Using strtod */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   /* initialize string pointer */
   const char *string = "51.2% are admitted"; /* initialize string */

   double d;        /* variable to hold converted sequence */
   char *stringPtr; /* create char pointer */

   d = strtod( string, &stringPtr );

   printf( "The string \"%s\" is converted to the\n", string );
   printf( "double value %.2f and the string \"%s\"\n", d, stringPtr );

   return 0; /* indicates successful termination */

} /* end main */
```

```
The string "51.2% are admitted" is converted to the
double value 51.20 and the string "% are admitted"
```

strtod converts a piece of a string to a **double**

Outline

fig08_10.c

```c
/* Fig. 8.10: fig08_10.c
   Using strtol */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   const char *string = "-1234567abc"; /* initialize string pointer */

   char *remainderPtr; /* create char pointer */
   long x;             /* variable to hold converted sequence */

   x = strtol( string, &remainderPtr, 0 );

   printf( "%s\"%s\"\n%s%ld\n%s\"%s\"\n%s%ld\n",
           "The original string is ", string,
           "The converted value is ", x,
           "The remainder of the original string is ",
           remainderPtr,
           "The converted value plus 567 is ", x + 567 );

   return 0; /* indicates successful termination */

} /* end main */
```

`strtol` converts a piece of a string to a **`long`**

```
The original string is "-1234567abc"
The converted value is -1234567
The remainder of the original string is "abc"
The converted value plus 567 is -1234000
```

Outline

fig08_11.c

```c
/* Fig. 8.11: fig08_11.c
   Using strtoul */
#include <stdio.h>
#include <stdlib.h>

int main( void )
{
   const char *string = "1234567abc"; /* initialize string pointer */
   unsigned long x;    /* variable to hold converted sequence */
   char *remainderPtr; /* create char pointer */

   x = strtoul( string, &remainderPtr, 0 );

   printf( "%s\"%s\"\n%s%lu\n%s\"%s\"\n%s%lu\n",
           "The original string is ", string,
           "The converted value is ", x,
           "The remainder of the original string is ",
           remainderPtr,
           "The converted value minus 567 is ", x - 567 );

   return 0; /* indicates successful termination */

} /* end main */
```

strtoul converts a piece of a string to an **unsigned long**

```
The original string is "1234567abc"
The converted value is 1234567
The remainder of the original string is "abc"
The converted value minus 567 is 1234000
```

Standard Input/Output Library Functions

- Functions in `<stdio.h>`
- Used to manipulate character and string data

Function prototype	Function description
`int getchar( void );`	Inputs the next character from the standard input and returns it as an integer.
`char *gets( char *s );`	Inputs characters from the standard input into the array `s` until a newline or end-of-file character is encountered. A terminating null character is appended to the array. Returns the string inputted into `s`. Note that an error will occur if `s` is not large enough to hold the string.
`int putchar( int c );`	Prints the character stored in `c` and returns it as an integer.
`int puts( const char *s );`	Prints the string `s` followed by a newline character. Returns a non-zero integer if successful, or `EOF` if an error occurs.
`int sprintf( char *s, const char *format, ... );`	Equivalent to `printf`, except the output is stored in the array `s` instead of printed on the screen. Returns the number of characters written to `s`, or `EOF` if an error occurs.
`int sscanf( char *s, const char *format, ... );`	Equivalent to `scanf`, except the input is read from the array `s` rather than from the keyboard. Returns the number of items successfully read by the function, or `EOF` if an error occurs.

Fig. 8.12 | Standard input/output library character and string functions.

Outline

fig08_14.c

(1 of 2)

```c
/* Fig. 8.14: fig08_14.c
   Using getchar and puts */
#include <stdio.h>

int main( void )
{
   char c;                /* variable to hold character input by user */
   char sentence[ 80 ]; /* create char array */
   int i = 0;             /* initialize counter i */

   gets( sentence ); /* take inputs until e
   puts( "Enter a line of text:" );

   /* use getchar to read each
   while ( ( c = getchar() ) !=
      sentence[ i++ ] = c;
   } /* end while */

   sentence[ i ] = '\0'; /* terminate string */

```

puts prints a line of text on the screen

getchar reads a single character from the user

```
21    /* use puts to display sentence */
22    puts( "\nThe line entered was:" );
23    puts( sentence );
24
25    return 0; /* indicates successful termination */
26
27 } /* end main */
```

```
Enter a line of text:
This is a test.

The line entered was:
This is a test.
```

Outline

fig08_14.c

(2 of 2)

Outline

fig08_15.c

```c
/* Fig. 8.15: fig08_15.c
   Using sprintf */
#include <stdio.h>

int main( void )
{
   char s[ 80 ]; /* create char array */
   int x;        /* x value to be input */
   double y;     /* y value to be input */

   printf( "Enter an integer and a double:\n" );
   scanf( "%d%lf", &x, &y );

   sprintf( s, "integer:%6d\ndouble:%8.2f", x, y );

   printf( "%s\n%s\n",
           "The formatted output stored in array s is:", s );

   return 0; /* indicates successful termination */

} /* end main */
```

```
Enter an integer and a double:
298 87.375
The formatted output stored in array s is:
integer:   298
double:   87.38
```

sprintf prints a line of text into an array like **printf** prints text on the screen

Outline

fig08_28.c

```c
/* Fig. 8.28: fig08_28.c
   Using strstr */
#include <stdio.h>
#include <string.h>

int main( void )
{
   const char *string1 = "abcdefabcdef"; /* string to search */
   const char *string2 = "def"; /* string to search for */

   printf( "%s%s\n%s%s\n\n%s\n%s%s\n",
      "string1 = ", string1, "string2 = ", string2,
      "The remainder of string1 beginning with the",
      "first occurrence of string2 is: ",
      strstr( string1, string2 ) );

   return 0; /* indicates successful termination */

} /* end main */
```

`strstr` returns the remainder of **`string1`** following the last occurrence of **`string2`**

```
string1 = abcdefabcdef
string2 = def

The remainder of string1 beginning with the
first occurrence of string2 is: defabcdef
```

Outline

fig08_29.c

(1 of 2)

```c
/* Fig. 8.29: fig08_29.c
   Using strtok */
#include <stdio.h>
#include <string.h>

int main( void )
{
   /* initialize array string */
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr; /* create char pointer */

   printf( "%s\n%s\n\n%s\n",
      "The string to be tokenized is:", string,
      "The tokens are:" );

   tokenPtr = strtok( string, " " ); /* begin tokenizing sentence */

   /* continue tokenizing sentence until tokenPtr becomes NULL */
   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " ); /* get next token */
   } /* end while */
```

strtok "tokenizes" **string** by breaking it into tokens at each space

Calling **strtok** again and passing it **NULL** continues the tokenizing of the previous string

```
23
24    return 0; /* indicates successful termination */
25
26 } /* end main */
```

```
The string to be tokenized is:
This is a sentence with 7 tokens

The tokens are:
This
is
a
sentence
with
7
tokens
```


fig08_29.c

(2 of 2)

Two dimensional array

- Example

```
#include<stdio.h>

int main()
{
    char days[][50]= {"Saturday","Sunday","Monday",
                      "Tuesday","Wednesday","Thursday","Friday"};
    int i;

    for(i=0; i<strlen(days); i++)
    {
        printf("%s\n",days[i]);
    }
    return 0;
}
```

PROGRAM 11-10 Print Days of the Week

```
1  /* Demonstrates an array of pointers to strings.
2        Written by:
3        Date written:
4  */
5  #include <stdio.h>
6
7  int main (void)
8  {
9  // Local Declarations
10    char*  pDays[7];
11    char** pLast;
12
13 // Statements
14    pDays[0] = "Sunday";
15    pDays[1] = "Monday";
16    pDays[2] = "Tuesday";
17    pDays[3] = "Wednesday";
18    pDays[4] = "Thursday";
```

PROGRAM 11-10 Print Days of the Week

```
19      pDays[5] = "Friday";
20      pDays[6] = "Saturday";
21
22      printf("The days of the week\n");
23      pLast = pDays + 6;
24      for (char** pWalker = pDays;
25                  pWalker <= pLast;
26                  pWalker++)
27           printf("%s\n", *pWalker);
28      return 0;
29  } // main
```

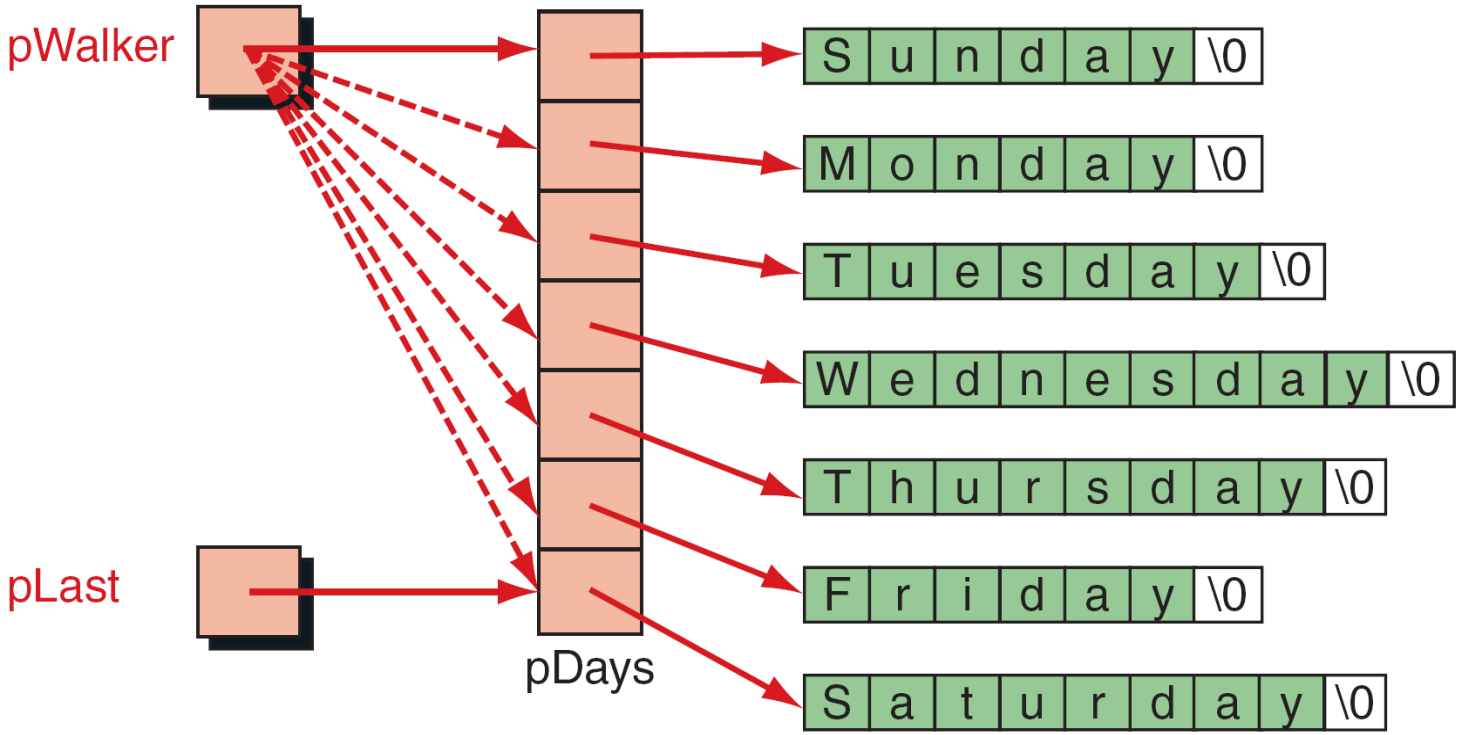

FIGURE 11-13 Pointers to Strings

Passing Arguments to main()

- We need to use arrays of pointers to char to write programs that use command line arguments.
  - Two arguments, conventionally called argc and argv can be used with main() to communicate with the operating system.
- Instead of

  int main(void)

  we use

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

The Meaning of argc and argv

- In the header of main:

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

  - the *variable* argc provides a count of command line arguments
  - the *array* argv is an array of pointers to char
    - argv can be thought of as an array of strings.

Common Programming Errors

- Overrunning the bounds of a string.
  - It is the programmer's responsibility to make sure enough space is allocated for a string.
  - Don't forget to allow memory space for the null character.
  - Can easily occur when strcat() is used.
- Using 'a' when "a" should be used.
  - "a" is stored as two characters 'a' and '\0'.
- If str is a string variable, then use scanf("%s", str); not scanf("%s", &str) to input a string.

Some exercise on string

- Write a recursive function to
  - Print the string in reverse order.
  - Determine the length of a string.
  - Count number of vowels.
  - Count number of occurrence of a particular character.
  - Parse the string and print the words using a special set of delimiters.
  - Check the string is palindrome or not.