Chapter 8 - Characters and Strings

Outline

8.1 Introduction

- Introduce some standard library functions
  - Easy string and character processing
  - Programs can process characters, strings, lines of text, and blocks of memory
- These techniques used to make
  - Word processors
  - Page layout software
  - Typesetting programs

8.2 Fundamentals of Strings and Characters

- Characters
  - Building blocks of programs
    - Every program is a sequence of meaningfully grouped characters
  - Character constant
    - An **`int`** value represented as a character in single quotes
    - **`'z'`** represents the integer value of **`z`**
- Strings
  - Series of characters treated as a single unit
    - Can include letters, digits and special characters (**`*`**, **`/`**, **`$`**)
  - String literal (string constant) - written in double quotes
    - **`"Hello"`**
  - Strings are arrays of characters
    - String a pointer to first character
    - Value of string is the address of first character

8.2 Fundamentals of Strings and Characters

- String declarations
  - Declare as a character array or a variable of type **`char *`**

```
char color[] = "blue";
char *colorPtr = "blue";
```

  - Remember that strings represented as character arrays end with **`'\0'`**
    - **`color`** has **`5`** elements
- Inputting strings
  - Use **`scanf`**

```
scanf("%s", word);
```

    - Copies input into **`word[]`**
    - Do not need **`&`** (because a string is a pointer)
  - Remember to leave room in the array for **`'\0'`**

8.3 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 slide contains a table of all the functions in **`<ctype.h>`**

8.3 Character Handling Library

Prototype	Description
`int isdigit( int c )`	Returns **`true`** if **`c`** is a digit and **`false`** otherwise.
`int isalpha( int c )`	Returns **`true`** if **`c`** is a letter and **`false`** otherwise.
`int isalnum( int c )`	Returns **`true`** if **`c`** is a digit or a letter and **`false`** otherwise.
`int isxdigit( int c )`	Returns **`true`** if **`c`** is a hexadecimal digit character and **`false`** otherwise.
`int islower( int c )`	Returns **`true`** if **`c`** is a lowercase letter and **`false`** otherwise.
`int isupper( int c )`	Returns **`true`** if **`c`** is an uppercase letter; **`false`** otherwise.
`int tolower( int c )`	If **`c`** is an uppercase letter, **`tolower`** returns **`c`** as a lowercase letter. Otherwise, **`tolower`** returns the argument unchanged.
`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 **`true`** if **`c`** is a white-space character—newline (**`'\n'`**), space (**`' '`**), form feed (**`'\f'`**), carriage return (**`'\r'`**), horizontal tab (**`'\t'`**), or vertical tab (**`'\v'`**)—and **`false`** otherwise
`int iscntrl( int c )`	Returns **`true`** if **`c`** is a control character and **`false`** otherwise.
`int ispunct( int c )`	Returns **`true`** if **`c`** is a printing character other than a space, a digit, or a letter and **`false`** otherwise.
`int isprint( int c )`	Returns **`true`** value if **`c`** is a printing character including space (**`' '`**) and **`false`** otherwise.
`int isgraph( int c )`	Returns **`true`** if **`c`** is a printing character other than space (**`' '`**) and **`false`** otherwise.

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

int main()
{
   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" );
   printf( "%s\n%s%s\n%s%s\n%s%s\n\n",
       "According to isalnum:",
       isalnum( 'A' ) ? "A is a " : "A is not a ",
       "digit or a letter",
       isalnum( '8' ) ? "8 is a " : "8 is not a ",
       "digit or a letter",
       isalnum( '#' ) ? "# is a " : "# is not a ",
       "digit or a letter" );
   printf( "%s\n%s%s\n%s%s\n%s%s\n%s%s\n%s%s\n",
       "According to isxdigit:",
       isxdigit( 'F' ) ? "F is a " : "F is not a ",
       "hexadecimal digit",
       isxdigit( 'J' ) ? "J is a " : "J is not a ",
       "hexadecimal digit",
```

Outline

1. Load header

2. Perform tests

3. Print

```
         isxdigit( '7' ) ? "7 is a " : "7 is not a ",
         "hexadecimal digit",
         isxdigit( '$' ) ? "$ is a " : "$ is not a ",
         "hexadecimal digit",
         isxdigit( 'f' ) ? "f is a " : "f is not a ",
         "hexadecimal digit" );
   return 0;
}
```

3. Print

```
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
```

Program Output

8.4 String Conversion Functions

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

Prototype	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`**.

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

int main()
{
   double d;

   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;
}
```

Outline

1. Initialize variable

2. Convert string

2.1 Assign to variable

3. Print

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

Program Output

8.5 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.
`int putchar( int c );`	Prints the character stored in **`c`**.
`int puts( const char *s );`	Prints the string **`s`** followed by a newline character.
`int sprintf( char *s, const char *format, ... );`	Equivalent to **`printf`**, except the output is stored in the array **`s`** instead of printing it on the screen.
`int sscanf( char *s, const char *format, ... );`	Equivalent to **`scanf`**, except the input is read from the array **`s`** instead of reading it from the keyboard.

Outline

1. Initialize variables

2. Input

3. Print

3.1 Function definition (note recursion)

```
/* Fig. 8.13: fig08_13.c
   Using gets and putchar */
#include <stdio.h>

int main()
{
   char sentence[ 80 ];
   void reverse( const char * const );

   printf( "Enter a line of text:\n" );
   gets( sentence );

   printf( "\nThe line printed backwards is:\n" );
   reverse( sentence );

   return 0;
}

void reverse( const char * const sPtr )
{
   if ( sPtr[ 0 ] == '\0' )
      return;
   else {
      reverse( &sPtr[ 1 ] );
      putchar( sPtr[ 0 ] );
   }
}
```

`reverse` calls itself using substrings of the original string. When it reaches the `'\0'` character it prints using `putchar`

Program Output

```
Enter a line of text:
Characters and Strings

The line printed backwards is:
sgnirtS dna sretcarahC
```

8.6 String Manipulation Functions of the String Handling Library

- String handling library has functions to
  - Manipulate string data
  - Search strings
  - Tokenize strings
  - Determine string length

Function prototype	Function description
`char *strcpy( char *s1, const char *s2 )`	Copies string **s2** into array **s1**. The value of **s1** is returned.
`char *strncpy( char *s1, const char *s2, size_t n )`	Copies at most **n** characters of string **s2** into array **s1**. The value of **s1** is returned.
`char *strcat( char *s1, const char *s2 )`	Appends string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.
`char *strncat( char *s1, const char *s2, size_t n )`	Appends at most **n** characters of string **s2** to array **s1**. The first character of **s2** overwrites the terminating null character of **s1**. The value of **s1** is returned.

Outline

1. Initialize variables

2. Function calls

3. Print

```c
/* Fig. 8.19: fig08_19.c
   Using strcat and strncat */
#include <stdio.h>
#include <string.h>

int main()
{
   char s1[ 20 ] = "Happy ";
   char s2[] = "New Year ";
   char s3[ 40 ] = "";

   printf( "s1 = %s\ns2 = %s\n", s1, s2 );
   printf( "strcat( s1, s2 ) = %s\n", strcat( s1, s2 ) );
   printf( "strncat( s3, s1, 6 ) = %s\n", strncat( s3, s1, 6 ) );
   printf( "strcat( s3, s1 ) = %s\n", strcat( s3, s1 ) );
   return 0;
}
```

Program Output

```
s1 = Happy
s2 = New Year
strcat( s1, s2 ) = Happy New Year
strncat( s3, s1, 6 ) = Happy
strcat( s3, s1 ) = Happy Happy New Year
```

8.7 Comparison Functions of the String Handling Library

- Comparing strings
  - Computer compares numeric ASCII codes of characters in string
  - Appendix D has a list of character codes

```
int strcmp( const char *s1, const char *s2 );
```

  - Compares string **s1** to **s2**
  - Returns a negative number if **s1 < s2**, zero if **s1 == s2** or a positive number if **s1 > s2**

```
int strncmp( const char *s1, const char *s2,
     size_t n );
```

  - Compares up to **n** characters of string **s1** to **s2**
  - Returns values as above

8.8 Search Functions of the String Handling Library

Function prototype	Function description
`char *strchr( const char *s, int c );`	Locates the first occurrence of character `c` in string `s`. If `c` is found, a pointer to `c` in `s` is returned. Otherwise, a `NULL` pointer is returned.
`size_t strcspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting of characters not contained in string `s2`.
`size_t strspn( const char *s1, const char *s2 );`	Determines and returns the length of the initial segment of string `s1` consisting only of characters contained in string `s2`.
`char *strpbrk( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of any character in string `s2`. If a character from string `s2` is found, a pointer to the character in string `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strrchr( const char *s, int c );`	Locates the last occurrence of `c` in string `s`. If `c` is found, a pointer to `c` in string `s` is returned. Otherwise, a `NULL` pointer is returned.
`char *strstr( const char *s1, const char *s2 );`	Locates the first occurrence in string `s1` of string `s2`. If the string is found, a pointer to the string in `s1` is returned. Otherwise, a `NULL` pointer is returned.
`char *strtok( char *s1, const char *s2 );`	A sequence of calls to `strtok` breaks string `s1` into “tokens”—logical pieces such as words in a line of text—separated by characters contained in string `s2`. The first call contains `s1` as the first argument, and subsequent calls to continue tokenizing the same string contain `NULL` as the first argument. A pointer to the current token is returned by each call. If there are no more tokens when the function is called, `NULL` is returned.

```c
/* Fig. 8.27: fig08_27.c
   Using strspn */
#include <stdio.h>
#include <string.h>

int main()
{
   const char *string1 = "The value is 3.14159";
   const char *string2 = "aehi lsTuv";

   printf( "%s%s\n%s%s\n\n%s\n%s%u\n",
           "string1 = ", string1, "string2 = ", string2,
           "The length of the initial segment of string1",
           "containing only characters from string2 = ",
           strspn( string1, string2 ) );
   return 0;
}
```

Outline

1. Initialize variables

2. Function calls

3. Print

```
string1 = The value is 3.14159
string2 = aehi lsTuv

The length of the initial segment of string1
containing only characters from string2 = 13
```

Program Output

Outline

1. Initialize variables

2. Function calls

3. Print

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

int main()
{
   char string[] = "This is a sentence with 7 tokens";
   char *tokenPtr;

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

   tokenPtr = strtok( string, " " );

   while ( tokenPtr != NULL ) {
      printf( "%s\n", tokenPtr );
      tokenPtr = strtok( NULL, " " );
   }

   return 0;
}
```

Program Output

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

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

8.9 Memory Functions of the String- handling Library

- Memory Functions
  - In **`<stdlib.h>`**
  - Manipulate, compare, and search blocks of memory
  - Can manipulate any block of data
- Pointer parameters are **`void *`**
  - Any pointer can be assigned to **`void *`**, and vice versa
  - **`void *`** cannot be dereferenced
    - Each function receives a size argument specifying the number of bytes (characters) to process

8.9 Memory Functions of the String- handling Library

Prototype	Description
`void *memcpy( void *s1, const void *s2, size_t n )`	Copies **`n`** characters from the object pointed to by **`s2`** into the object pointed to by **`s1`**. A pointer to the resulting object is returned.
`void *memmove( void *s1, const void *s2, size_t n )`	Copies **`n`** characters from the object pointed to by **`s2`** into the object pointed to by **`s1`**. The copy is performed as if the characters are first copied from the object pointed to by **`s2`** into a temporary array, and then copied from the temporary array into the object pointed to by **`s1`**. A pointer to the resulting object is returned.
`int memcmp( const void *s1, const void *s2, size_t n )`	Compares the first **`n`** characters of the objects pointed to by **`s1`** and **`s2`**. The function returns **`0`**, less than **`0`**, or greater than **`0`** if **`s1`** is equal to, less than or greater than **`s2`**, respectively.
`void *memchr(const void *s, int c, size_t n )`	Locates the first occurrence of **`c`** (converted to **`unsigned char`**) in the first **`n`** characters of the object pointed to by **`s`**. If **`c`** is found, a pointer to **`c`** in the object is returned. Otherwise, **`0`** is returned.
`void *memset( void *s, int c, size_t n )`	Copies **`c`** (converted to **`unsigned char`**) into the first **`n`** characters of the object pointed to by **`s`**. A pointer to the result is returned.

Outline

1. Initialize variables

2. Function calls

3. Print

```c
/* Fig. 8.32: fig08_32.c
   Using memmove */
#include <stdio.h>
#include <string.h>

int main()
{
   char x[] = "Home Sweet Home";

   printf( "%s%s\n",
           "The string in array x before memmove is: ", x );
   printf( "%s%s\n",
           "The string in array x after memmove is:  ",
           memmove( x, &x[ 5 ], 10 ) );

   return 0;
}
```

Program Output

```
The string in array x before memmove is: Home Sweet Home
The string in array x after memmove is:  Sweet Home Home
```

8.10 Other Functions of the String Handling Library

- **`char *strerror( int errornum );`**
  - Creates a system-dependent error message based on **`errornum`**
  - Returns a pointer to the string
- **`size_t strlen( const char *s );`**
  - Returns the number of characters (before **`NULL`**) in string **`s`**

Outline

1. Function call

2. Print

```c
/* Fig. 8.37: fig08_37.c
   Using strerror */
#include <stdio.h>
#include <string.h>

int main()
{
   printf( "%s\n", strerror( 2 ) );
   return 0;
}
```

```
No such file or directory
```

Program Output