Chapter 8
Characters and Strings

Associate Prof. Yuh-Shyan Chen
Dept. of Computer Science and Information Engineering
National Chung-Cheng University

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
8.2 Fundamentals of Strings and Characters (II)

- Strings
  - Series of characters treated as a single unit
    - Can include letters, digits, and certain special characters (*, /, $)
  - String literal (string constant) - written in double quotes
    - "Hello"
    - Somewhere in memory

8.2 Fundamentals of Strings and Characters (III)

- 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

8.2 Fundamentals of Strings and Characters (IV)

- Inputting strings
  - Use scanf
    - scanf("%s", word);
  - Copies input into word[], which does not need & (because a string is a pointer)
  - Remember to leave space 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

### 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 whitespace 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 */

#include <stdio.h>
#include <ctype.h>

int main()
{
    printf( "%s
%s%s
%s%s

", "According to isdigit: ",
          isdigit( '8' ) ? "8 is a " : "8 is not a ", "digit",
          isdigit( '#' ) ? "# is a " : "# is not a ", "digit" );
    printf( "%s
%s%s
%s%s
%s%s

",
            "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
%s%s
%s%s
%s%s

",
            "According to isalnum:",
            isalnum( 'A' ) ? "A is a digit or a letter",
            isalnum( '8' ) ? "8 is a digit or a letter",
            isalnum( '#' ) ? "# is not a digit or a letter" );
    printf( "%s
%s%s
%s%s
%s%s
%s%s
%s%s
",
            "According to isxdigit:",
            isxdigit( 'F' ) ? "F is a hexadecimal digit",
            isxdigit( 'J' ) ? "J is a hexadecimal digit",
            isxdigit( '7' ) ? "7 is a hexadecimal digit",
            isxdigit( '$' ) ? "$ is a hexadecimal digit",
            isxdigit( 'f' ) ? "f is a hexadecimal digit" );
    return 0;
}
```

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
8.4 String Conversion Functions

Conversion functions

- In \texttt{<stdlib.h>} (general utilities library)
- Convert strings of digits to integer and floating-point values

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{atof()}</td>
<td>Converts the string \texttt{nPtr} to double</td>
</tr>
<tr>
<td>\texttt{atoi()}</td>
<td>Converts the string \texttt{nPtr} to int</td>
</tr>
<tr>
<td>\texttt{atol()}</td>
<td>Converts the string \texttt{nPtr} to long int</td>
</tr>
<tr>
<td>\texttt{strtod()}</td>
<td>Converts the string \texttt{nPtr} to double</td>
</tr>
<tr>
<td>\texttt{strtol()}</td>
<td>Converts the string \texttt{nPtr} to long int, \texttt{base}</td>
</tr>
<tr>
<td>\texttt{strtoul()}</td>
<td>Converts the string \texttt{nPtr} to unsigned long int, \texttt{base}</td>
</tr>
</tbody>
</table>

Using \texttt{atof}:

```c
/* Fig. 8.7: Fig08_07.c */
int main()
{
    double d = atof("3593");
    printf("The string \"3593\" converted to int is \%,.2f\n", d);
    return 0;
}
```

Using \texttt{atoi}:

```c
/* Fig. 8.8: Fig08_08.c */
int main()
{
    int i = atoi("3593");
    printf("The string \"3593\" converted to int is \%, l\n", i);
    return 0;
}
```
Function `strtof` converts a sequence of characters representing a floating point value to double.

The function receives two arguments – a string (char *) and a pointer to a string (char **).

The string contains the character sequence to be converted portion of the string.

```
#include <stdio.h>

int main()
{
   double d;
   char *str;
   printf("Enter a floating point number: ");
   scanf("%lf", &d);
   printf("The floating point number is: %.2f\n", d);
   return 0;
}
```

Using `strtof`

```c
#include <stdio.h>

int main()
{
   double d;
   char *str;
   printf("Enter a floating point number: ");
   scanf("%lf", &d);
   printf("The floating point number is: %.2f\n", d);
   return 0;
}
```
Using `strtol`

```c
1 #include <stdio.h>
2
3 main()
4 {
5     char *s = "1234567890";
6     int i = strtol(s, NULL, 0);
7     printf("The original string is ", s);
8     printf(\n"The converted value is ", i);
9     printf(\n"The remainder of the original string is ", s);
10    printf(\n"The converted value minus 876 is ", i - 876);
11    return 0;
12 }
```

The original string is "1234567890"
The converted value is 8765
The remainder of the original string is "890"
The converted value minus 876 is 0

Fig. 8.11 Using `strtol`

Using `gets` and `putchar`

```c
1 /* Fig. 8.13 Fig08_13.c */
2 #include <stdio.h>
3
4 main()
5 {
6     char sentence[80];
7     void reverse(const char * const s);
8     printf("Enter a line of text: ");
9     gets(sentence);
10    printf(\n"The line printed backwards is: ");
11    reverse(sentence);
12    printf(\n"\nReturn \$\n" );
13    return 0;
14 }
```

```
19 void reverse(const char * const s)
20 {
21    if ( s[0] == '\0' )
22    return;
23    else
24        reverse(s + 1);
25    puts(s[0] + 1);
26 }
```

Enter a line of text: able was i saw a man
The line printed backwards is: man a saw i was able

Fig. 8.13 Using `gets` and `putchar` (part 1 of 2)

8.5 Standard Input/Output Library Functions

<table>
<thead>
<tr>
<th>Functions in <code>&lt;stdio.h&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Function</td>
</tr>
<tr>
<td>---</td>
</tr>
<tr>
<td><code>int getchar(void)</code></td>
</tr>
<tr>
<td><code>char *gets(char *s)</code></td>
</tr>
<tr>
<td><code>int putchar(int c)</code></td>
</tr>
<tr>
<td><code>int puts(const char *s)</code></td>
</tr>
<tr>
<td><code>int sprintf(char *s, const char *format, ...)</code></td>
</tr>
<tr>
<td><code>int sscanf(char *s, const char *format, ...)</code></td>
</tr>
</tbody>
</table>

Fig. 8.13 Using `gets` and `putchar` (part 2 of 2)
Using `getchar` and `puts`

```c
1 #include <stdio.h>
2
3 int main()
4 {
5    char c, sentence[40];
6    int i = 0;
7    puts("Enter a line of text: ");
8    while ((c = getchar()) != '\n')
9        sentence[i++] = c;
10    sentence[i] = '\0'; // insert null at end of string
11    puts("The line entered was: ");
12    puts(sentence);
13    return 0;
}
```

**Result**

Enter a line of text:
This is a test.
The line entered was:
This is a test.

Fig. 8.14 Using `getchar` and `puts` (part 1 of 2).

Using `sprintf`

```c
1 #include <stdio.h>
2
3 int main()
4 {
5    char c[100];
6    int x, y;
7    printf("Enter an integer and a double: ");
8    scanf("%d %f", &x, &y);
9    printf("x = %d, y = %.2f
", x, y);
10    return 0;
}
```

Enter an integer and a double:
3212.345
The formatted output stored in array `x` is:
`integer`: 3212
`double`: 3212.34

Using `sscanf`

```c
1 #include <stdio.h>
2
3 int main()
4 {
5    char c[100];
6    int x;
7    double y;
8    printf("Enter an integer and a double: ");
9    scanf("%d %f", &x, &y);
10    printf("The values stored in character array `c` are:",
11       "integer": %d, "double": %.2f
", x, y);
12    return 0;
}
```

The values stored in character array `c` are:
`integer`: 3212
`double`: 3212.34

Fig. 8.16 Using `sscanf`.
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

<table>
<thead>
<tr>
<th>Function prototype</th>
<th>Function description</th>
</tr>
</thead>
<tbody>
<tr>
<td>char *strcpy( char *s1, const char *s2 )</td>
<td>Copies string ( s2 ) into array ( s1 ). The value of ( s1 ) is returned.</td>
</tr>
<tr>
<td>char *strncpy( char *s1, const char *s2, size_t n )</td>
<td>Copies at most ( n ) characters of string ( s2 ) into array ( s1 ). The value of ( s1 ) is returned.</td>
</tr>
<tr>
<td>char *strcat( char *s1, const char *s2 )</td>
<td>Appends string ( s2 ) to array ( s1 ). The first character of ( s2 ) overwrites the terminating null character of ( s1 ). The value of ( s1 ) is returned.</td>
</tr>
<tr>
<td>char *strncat( char *s1, const char *s2, size_t n )</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

**Using strcpy and strncpy**

```c
#include <stdio.h>
#include <string.h>

int main()
{
    char x[] = "Happy Birthday to You";
    char y[35], z[35];

    printf( "Whoa! Whoa!
        The string in array x is: " , x );
    printf( "The string in array y is: ", strcpy( y , x ) );
    strcpy( x , "Happy Birthday to You" );
    printf( "The string in array x is: ", strcpy( x , "Happy Birthday to You" ) );
    printf( "The string in array y is: " , y );
}
```

---

**Result**

- The string in array x is: Happy Birthday to You
- The string in array y is: Happy Birthday to You
- The string in array y is: Happy Birthday
Using `strcat` and `strncat` functions by yourself.

- `char *strcp`y (char *s1, const char *s2)
  - Copies string s2 into 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 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.

### Comparison Functions of the String Handling Library

- `int strcmp` (const char *s1, const char *s2);
  - Compares string s1 to s2
  - Returns a negative number (s1 < s2), zero (s1 == s2), or a positive number (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.

### Function Prototype

- `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 `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. If there are no more tokens when the function is called, `NULL` is returned.
Using strchr

Using strcspn

Using strcspn and strpbrk

- Function strcspn determines the length of the initial part of the string in first argument that does not contain any characters from the string in its second argument.

- Function strcspn searches for the first occurrence in its first string argument of any character in its second string argument.

Using strpbrk

Using strpbrk
strrchr and strspn

- Function strrchr searches for the last occurrence of the specified character in a string.

- Function strspn determines the length of the initial part of the string in its first argument that contains only characters from the string in its second argument.

Using strrchr

```
1 /* Fig. 8.26: Fig8_26.c
2 include <stdio.h>
3 include <string.h>
4 main()
5 { char *string = "A doc has many animals";
6     int q = 0;
7     for (q = 0; string[q] != 0; q++)
8         if (string[q] == ')' )
9             printf("The remainder of string1 beginning with the last occurrence of character '(', q = ");
10            return q;
11 }
```

The remainder of string1 beginning with the last occurrence of character '(' is 'braa'

Using strspn

```
1 /* Fig. 8.27: Fig8_27.c
2 include <stdio.h>
3 include <string.h>
4 main()
5 { const char *string1 = "The value is 3.14159";
6     const char *string2 = "x = yz 1234567890";
7     printf("The length of the initial segment of string1 containing only characters from string2 = ",
8            strspn (string1, " ") );
9     return 2;
10 }
```

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

Strstr and strtok

- Function strstr searches for the first occurrence of its second string argument in its first string argument.

- Function strtok is used to break a string into a series of tokens.

  - `tokenPtr = strtok ( string, " ");`
  - Indicates that tokens in string are separated by spaces.
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 (II)

"Object" refers to a block of data

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>void *memcpy( void *s1, const void *s2, size_t n )</td>
<td>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.</td>
</tr>
<tr>
<td>void *memmove( void *s1, const void *s2, size_t n )</td>
<td>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.</td>
</tr>
<tr>
<td>int memcmp( const void *s1, const void *s2, size_t n )</td>
<td>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.</td>
</tr>
<tr>
<td>void *memchr(const void *s, int c, size_t n )</td>
<td>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.</td>
</tr>
<tr>
<td>void *memset( void *s, int c, size_t n )</td>
<td>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.</td>
</tr>
</tbody>
</table>

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
Try to write `str_tok`, `mem_cpy`, `str_len` functions by yourself.

- **char *str_tok (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`.

- **void *mem_cpy (void *s1, const void *s2, size_t n)**
  - Copies `n` characters from the object pointed to by `s2` into object pointer to by `s1`. A pointer to the resulting object is returned.

- **size_t *str_len (const char *s)**
  - Determines the length of string `s`. The number of characters preceding the terminating null character is returned.

---

Fig. 8.36 Using `strlen`