Chapter 8: Character & String

- In this chapter, you'll learn about;
  - Fundamentals of Strings and Characters
  - The difference between an integer digit and a character digit
  - Character handling library
  - String conversion functions
  - Standard input/output library functions
  - String manipulation functions

Fundamentals of Characters and Strings

- Characters in C consist of any printable or nonprintable character in the computer's character set including lowercase letters, uppercase letters, decimal digits, special characters and escape sequences.
- A character is usually stored in the computer as an 8-bits (1 byte) integer.
- The integer value stored for a character depends on the character set used by the computer on which the program is running.

Fundamentals of Characters and Strings

- There are two commonly used character sets:
  - ASCII (American Standard Code for Information Interchange)
  - EBCDIC (Extended Binary Coded Decimal Interchange Code)

ASCII Table

The ASCII Table

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
00	00	NUL	32	20	SP	64	40	@	96	60	‘
01	01	SOH	33	21	!	65	41	A	97	61	a
02	02	STX	34	22	"	66	42	B	98	62	b
03	03	ETX	35	23	#	67	43	C	99	63	c
04	04	EOT	36	24	$	68	44	D	100	64	d
05	05	ENQ	37	25	%	69	45	E	101	65	e
06	06	ACK	38	26	&	70	46	F	102	66	f
07	07	BEL	39	27	’	71	47	G	103	67	g
08	08	BS	40	28	(	72	48	H	104	68	h
09	09	HT	41	29	)	73	49	I	105	69	i
10	0A	LF	42	2A	*	74	4A	J	106	6A	j
11	0B	VT	43	2B	+	75	4B	K	107	6B	k
12	0C	FF	44	2C	,	76	4C	L	108	6C	l
13	0D	CR	45	2D	-	77	4D	M	109	6D	m
14	0E	SO	46	2E	.	78	4E	N	110	6E	n
15	0F	SI	47	2F	/	79	4F	O	111	6F	o
16	10	DLE	48	30	0	80	50	P	112	70	p
17	11	DC1	49	31	1	81	51	Q	113	71	q
18	12	DC2	50	32	2	82	52	R	114	72	r
19	13	DC3	51	33	3	83	53	S	115	73	s
20	14	DC4	52	34	4	84	54	T	116	74	t
21	15	NAK	53	35	5	85	55	U	117	75	u
22	16	SYN	54	36	6	86	56	V	118	76	v
23	17	ETB	55	37	7	87	57	W	119	77	w
24	18	CAN	56	38	8	88	58	X	120	78	x
25	19	EM	57	39	9	89	59	Y	121	79	y
26	1A	SUB	58	3A	:	90	5A	Z	122	7A	z
27	1B	ESC	59	3B	;	91	5B	[	123	7B	{
28	1C	FS	60	3C	<	92	5C	\	124	7C	|
29	1D	GS	61	3D	=	93	5D	]	125	7D	}
30	1E	RS	62	3E	>	94	5E	^	126	7E	~
31	1F	US	63	3F	?	95	5F	_	127	7F	DEL

Difference between an integer digit and a character digit

- char num = 1 and char num = ‘1’ are not the same.
- char num = 1 is represented in the computer as 00000001.
- char num = ‘1’ on the other hand is number 49 according to the ASCII character set. Therefore, it is represented in the computer as 00110001.

Example: ASCII character

```
#include <stdio.h>

int main(void)
{
   char my_A = 'A';
   char my_Z = 'Z';
   char my_a = 'a';
   char my_z = 'z';

   printf("\nASCII value for A is %d", my_A);
   printf("\nASCII value for Z is %d",my_Z);
   printf("\nASCII value for a is %d", my_a);
   printf("\nASCII value for z is %d",my_z);

   printf("\n");
   printf("\n65 in ASCII represents %c",65);
   printf("\n90 in ASCII  represents %c",90);
   printf("\n97 in ASCII represents %c",97);
   printf("\n122 in ASCII  represents %c",122);

   return(0);
}
```

```
ASCII value for A is 65
ASCII value for Z is 90
ASCII value for a is 97
ASCII value for z is 122

65 in ASCII represents A
90 in ASCII  represents Z
97 in ASCII represents a
122 in ASCII  represents z
```

Example cont…

```
#include <stdio.h>

void main(void)
{
    char ch;

    printf("enter a character: ");
    scanf("%c", &ch);

    if (ch >= 'A' && ch <= 'Z')
    {
        printf("\ncapital letter\n");
    }
    return (0);
}
```

equivalent to

```
#include <stdio.h>

void main(void)
{
    char ch;

    printf("enter a character: ");
    scanf("%c", &ch);

    if (ch >= 65 && ch <= (65+26))
    {
        printf("\ncapital letter\n");
    }
    return (0);
}
```

Character Handling Library

- Character handling library includes several function that perform useful tests and manipulation of character data.
- Each function receives a character, represented as an int or EOF, as an argument.
- When using functions from the character handling library, the header file <ctype.h> needs to be included.
- Characters in these functions are manipulated as integers (since a character is basically a 1 byte integer).

Prototype	Function Descriptions
int isdigit(int c)	Returns a true if value c is a digit, and 0 (false) otherwise.
int isalpha(int c)	Returns a true if value c is a letter, and 0 otherwise.
int isalnum(int c)	Returns a true if value 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.
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.
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 0 otherwise.
int iscntrl(int c)	Returns a true if c is a control character, and 0 otherwise.
int ispunct(int c)	Returns a true if c is a printing character other than a space, a digit or a letter, and 0 otherwise.
int isprint(int c)	Returns a true value if c is a printing character including space (‘ ’), and 0 otherwise.
int isgraph(int c)	Returns a true value if c is a printing character other than space (‘ ’), and 0 otherwise.

Example

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

int main(void)
{
    char loop = 'y';
    char my_char;

    while (loop == 'y' || loop == 'Y')
    {
        fflush(stdin);
        printf("Enter a character: ");
        my_char = getchar();

        if (isalpha(my_char))
        {
            printf("The character is an alphabet\n");

            if (islower(my_char))
                printf("and it is also a lower case alphabet\n");
            if (isupper(my_char))
                printf("and it is also an upper case alphabet\n");
        }
        if (isdigit(my_char))
            printf("The character is a digit\n");

        if (ispunct(my_char))
            printf("The character is a punctuator\n");

        fflush(stdin);
        printf("\nanother character? [y = yes, n = no]: ");
        loop = getchar();
        printf("\n");
    } return (0);
}
```

```
C:\WINDOWS\system32\cmd.exe
Enter a character: g
The character is an alphabet
and it is also a lower case alphabet

another character? [y = yes, n = no]: y

Enter a character: G
The character is an alphabet
and it is also an upper case alphabet

another character? [y = yes, n = no]: y

Enter a character: 50
The character is a digit

another character? [y = yes, n = no]: y

Enter a character: -
The character is a punctuator

another character? [y = yes, n = no]: y

Enter a character: *
The character is a punctuator

another character? [y = yes, n = no]: n

Press any key to continue . . .
```

Fundamentals of Characters and Strings

- A string in C is an **array** of characters ending with the null character (‘\0’). It is written inside a double quotation mark (“ ”)
- A string may be assigned (in a declaration) to either a char array or to a char pointer:
  - `char colour[];` OR
  - ~~`char *colour;`~~

Fundamentals of Characters and Strings

- A string can be initialised as follows:
  - `char colour[] = {'g', 'r', 'e', 'e', 'n', '\0'};`
  - `char colour[] = "green";`
- In memory, this is how the characters are stored.

g	r	e	e	n	\0

Fundamentals of Characters and Strings

- Notice that even though there are only five characters in the word ‘green’, six characters are stored in the computer. The last character, the character ‘\0’, is the NULL character which indicates the end of the string.
- Therefore, if an array of characters is to be used to store a string, the array must be large enough to store the string and its terminating NULL character.

```
char color[6] = "green";
```

The minimum size of a string must at least the ‘total no of char in the string + 1’

Briefly review about strings :

- We can initialise string variables at compile time such as;
  - `char name[10] = "Arris";`
  - This initialisation creates the following spaces in the memory.

A	r	r	i	s	\0	\0	\0	\0	\0
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Fundamentals of Characters and

- If we happen to declare a string like this:

```
char my_drink[3] = "tea";
```

- We will get the following syntax error:

```
error C2117: 'tea' : array bounds overflow
```

- Instead, we need to at least declare the array with (the size of the string + 1) to accommodate the null terminating character '\0'.

```
char my_drink[4] = "tea";
```

Example: string and '\0'

```
#include <stdio.h>

int main(void)    /* a program that counts the number of
                     characters in a string */
{

   char sentence[20] = "I love Malaysia";

   int i, count = 0;

   for (i = 0; sentence[i] != '\0'; i++)
   {
        count++;
   }

   printf("%s has %d characters including the whitespace", sentence, count);
   return (0);
}
```

Sample output:

I love Malaysia has 15 characters including the whitespace

Reading and Displaying Strings

- Standard Input Functions
  - scanf( )
  - gets( )
- Standard Output Functions
  - printf( )
  - puts( )
- Use scanf function together with the format specifier %s for interactive input string (no whitespace character).
- If the string to be read as an input has embedded whitespace characters, use standard *gets* function.

Example

```
#include <stdio.h>

int main(void)
{
   char string1[50];
   char string2[50];

   printf("Enter a string less than 50 characters with spaces: \n ");
   gets(string1);

   printf("\nYou have entered: ");
   puts(string1);

   printf("\Enter a string less than 50 characters, with spaces: \n");
   scanf("%s",string2);

   printf("\nYou have entered: %s\n", string2);
   return(0);
}
```

```
Enter a string less than 50 characters with spaces:
hello world

You have entered: hello world

Enter a string less than 50 characters, with spaces:
hello world


You have entered: hello
```

String Conversion Functions

- These functions convert strings of digits to integer and floating-point values.
- To use these functions, the general utilities library <stdlib.h>, needs to be included.
- Note that these functions take a constant value as their argument. This means that we can only pass a constant string to the functions. For example:
  - atoi (“1234”);
  - const char *hello = “9999”;

    atoi(hello);

String Conversion Functions

Function Prototype	Function Description
double atof (const char *nPtr)	Converts the sting 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 *long*.
unsigned long strtoul (const char *nPtr, char **endptr, int base)	Converts the string nPtr to *unsigned long*.

Example

```
/*1.  Converting a String Into an int Using atoi. */
#include <stdio.h>
#include <stdlib.h>
int main(void)
{
   char str1[ ] = "124z3yu87";
   char str2[ ] = "-3.4";
   char *str3 = "e24";
   printf("str1: %d\n", atoi(str1));
   printf("str2: %d\n", atoi(str2));
   printf("str3: %d\n", atoi(str3));
   return(0);
}
```

Output:

```
   str1: 124
   str2: -3
   str3: 0
```

Standard Input/Output Library Functions

- Include <stdio.h> to use these functions.

Function Prototype	Function Description
int getchar(void)	Get the next character from the standard input and return it as an integer
char *gets(char *s)	Get 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)	Print the character stored in **c**
int puts(const char *s)	Print the string **s** followed by a newline character
int sprintf(char *s, const char *format, …)	Equivalent to printf except that the output is stored in the array **s** instead of printing on the screen
int sscanf(char *s, const char *format, …)	Equivalent to scanf except that the input is read from the array **s** instead of reading from the keyboard

String Manipulation Functions

- Include <string.h> to use these functions.

Function Prototype	Function Description
char *strcpy (char *s1, const char *s2)	Copies the string **s2** into the array **s1**. The value of **s1** is returned
char *strncpy (char *s1, const char *s2, size_t n)	Copies at most **n** characters of the string **s2** into the array **s1**. The value of **s1** is returned.
char *strcat (char *s1, const char *s2)	Appends the string **s2** to the 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.

String Comparison Functions

- Include <string.h> to use these functions

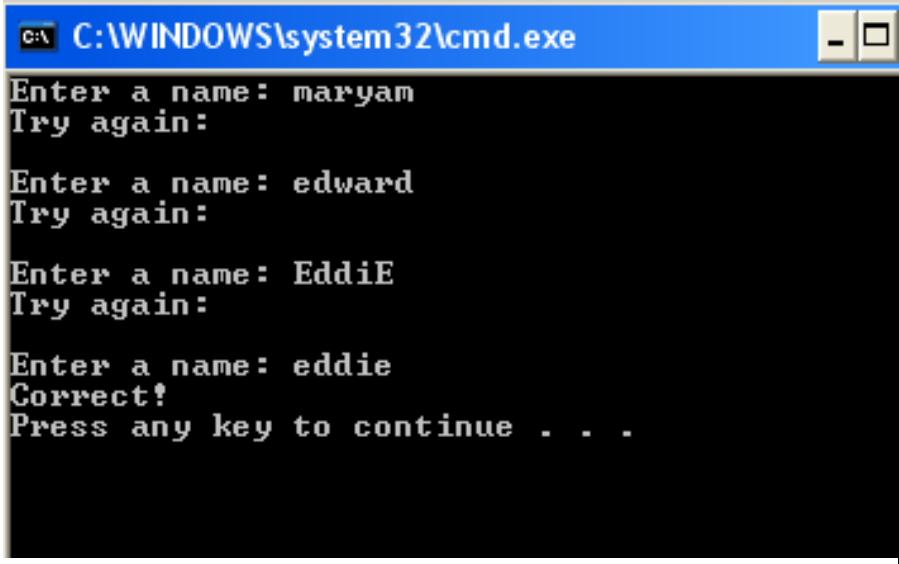
Function Prototype	Function Description
int strcmp (const char *s1, const char *s2)	Compares the string **s1** to the string **s2**. The function returns 0, less than 0 (negative value), or greater than 0 if **s1** is equal to, less than or greater than **s2** respectively.
int strncmp (const char *s1, const char *s2, size_t n)	Compares up to **n** characters of the string **s1** to the string **s2**. The function returns 0, less than 0, or greater than 0 if **s1** is equal to, less than or greater than **s2** respectively.

strcmp

- int strcmp (const char *s1, const char *s2);
  - strcmp will accept two strings. It will return an integer. This integer will either be:
    - Negative if s1 is less than s2.
    - Zero if s1 and s2 are equal.
    - Positive if s1 is greater than s2.
- strcmp is case sensitive.
- strcmp also passes the address of the character array to the function to allow it to be accessed.

Example: strcmp

```
#include <stdio.h>
#include <string.h>
int main(void)
{
   char name[20] = "eddie";
   char guess[20];
   int correct = 1;

   while(correct==1)
   {
        printf("Enter a name: ");
        gets(guess);

        if(strcmp(name, guess)==0) /*both string are indentical */
        {
             printf("Correct!\n");
             correct = 0;
        }
        else
            printf("Try again: \n\n");
   }
   return(0);
}
```



strcat

- char *strcat (char *dest, const char *src);
  - strcat is short for string concatenate, which means to add to the end, or append.
  - It adds the second string to the first string.
  - It returns a pointer to the concatenated string.
  - Make sure that the size of *dest* is large enough to hold the entire contents of *src* as well as its own contents.

Example 1: strcat

```
/* Concatenating Strings Using strcat  */
#include <stdio.h>
#include <string.h>

int main(void)
{
   char str1[50] = "Hello ";
   char str2[15] = "World";
   strcat(str1, str2);
   printf("str1: %s\n", str1);
   return(0);
}
```

```
str1: Hello World
Press any key to continue
```

Note : This only works if you've defined the str1 array to be large enough to hold the characters of the new string. If you don't specify a size, the program will crash.

Example 2: strcat

```
/* Concatenating Strings Using strcat  */
#include <stdio.h>
#include <string.h>

int main(void)
{
   char str1[50] = "Hello ";
   char str2[15] = "World";
   strcat(str2, str1);
   printf("str2: %s\n", str2);
   return(0);
}
```

```
str2: WorldHello
Press any key to continue
```

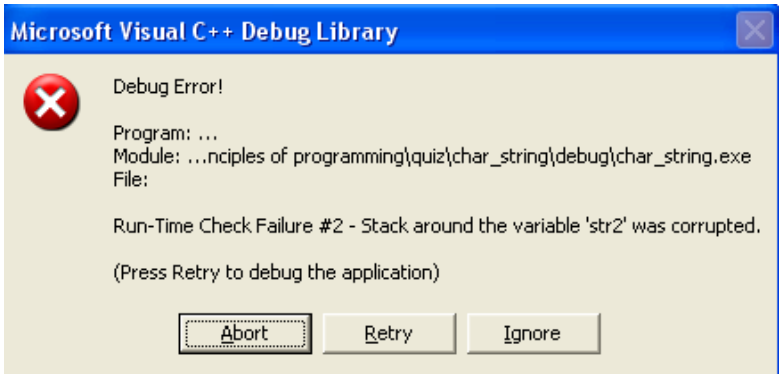
Note : This only works if you've defined the str1 array to be large enough to hold the characters of the new string. If you don't specify a size, the program will crash.

Example 3: strcat

```
/* Concatenating Strings Using strcat  */
#include <stdio.h>
#include <string.h>

int main(void)
{
   char str1[50] = "Hello ";
   char str2[7] = "World";
   strcat(str2, str1);
   printf("str2: %s\n", str2);
   return(0);
}
```

```
str2: WorldHello
Press any key to continue
```

since the destination string has not enough space to hold the combined string, an error message is generated.



strncat

- char *strncat (char *dest, const char *src, size_t n);
  - strncat adds n characters from the second string to the first string.
  - It returns a pointer to the concatenated string.
  - Make sure that the size of *dest* is large enough to hold the entire contents of *src* as well as its own contents.

Example: strncat

/* Concatenating Strings Using strncat */

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

int main(void)
{
   char str1[50] = "Hello ";
   char str2[15] = "World";
   strncat(str1, str2, 2);
   printf("str1: %s\n", str1);
   return(0);
}
```

```
str1: Hello Wo
Press any key to continue
```

strcpy

- char *strcpy (char *dest, const char *src);
  - strcpy is short for string copy, which means it copies the entire contents of *src* into *dest*. The contents of *dest* after strcpy will be exactly the same as src.

Example 1: strcpy

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

int main(void)
{
   char string1[100] = "Malaysia";
   char string2[50] = "Gemilang";
   strcpy(string1,string2);
   printf("string1: %s\n", string1);
   printf("string2: %s\n", string2);
   return(0);
}
```

```
string1: Gemilang
string2: Gemilang
Press any key to continue
```

Example 2: strcpy

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

int main(void)
{
   char string1[100] = "Malaysia";
   char string2[50] = "Boleh";
   strcpy(string1,string2);
   printf("string1: %s\n", string1);
   printf("string2: %s\n", string2);
   return(0);
}
```

string2 has less character than string1

```
string1: Boleh
string2: Boleh
Press any key to continue
```

Example 3: strcpy

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

int main(void)
{
   char string1[100] = "Semenanjung ";
   char string2[50] = "Malaysia";
   strcpy(string1,string2);
   printf("string1: %s\n", string1);
   printf("string2: %s\n", string2);
   return(0);
}
```

string1 has more character than string2

```
string1: Malaysia
string2: Malaysia
Press any key to continue
```

strncpy

- char *strcpy (char *dest, const char *src, size_t n);
  - Strncpy copies the first n characters from the *src* into *dest*.

Example: strncpy

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

int main(void)
{
   char string1[100]  = "Malaysia";
   char string2[50] = "Gemilang";

   strncpy(string1, string2, 4);

   printf("string1: %s\n", string1);
   return(0);
}
```

```
string1: Gemiysia

Press any key to continue
```

strlen

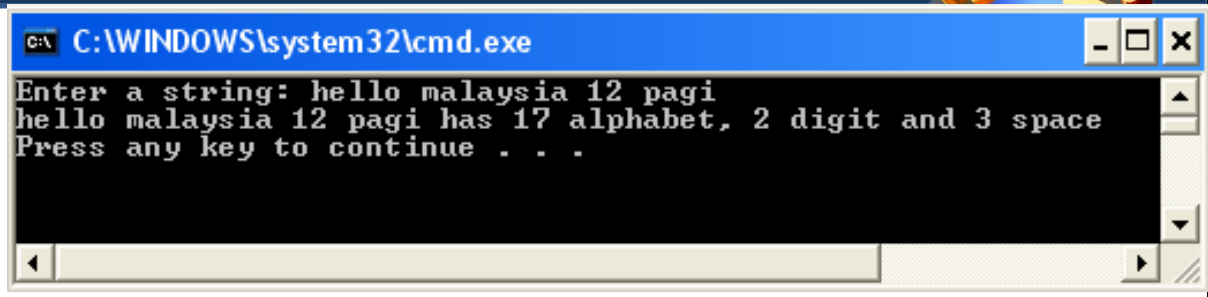
- size_t strlen (const char *s);
  - strlen will return the length of a string, minus the null character ('\0'). The size_t is nothing to worry about. Just treat it as an integer that cannot be negative, which it is.

Example 1: strlen

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

int main(void)
{
   char sentence[50] = "I love malaysia";

   int i, count = 0;

   count = strlen(sentence);
   printf("%s has %d characters including the whitespace",
   sentence, count);

   return(0);
}
```

```
I love Malaysia has 15 characters including the whitespace

Press any key to continue
```

Example 2: strlen




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

int main(void)
{
    char string[50];
    int length, i, alpha = 0, digit = 0, space = 0;
    printf("Enter a string: ");
    gets(string);

    length = strlen(string);
    for (i = 0; i < length; i++)
    {
          if (isalpha(string[i]))
                alpha++;
          if (isdigit(string[i]))
                digit++;
          if (isspace(string[i]))
                space++;
    }

    printf("%s has %d alphabet, %d digit and %d space \n",string,alpha,digit,space);

    return(0);
}
```

String Search Functions

- Include <string.h> to use these functions.

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, s pointer to **c** 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 found 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 of 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 call to strtok breaks string **s1** into "tokens" – logical pieces such as words in a line of text – separated by characters contained in strnig **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.

SUMMARY

- C has a standard character-handling library that includes some useful functions for testing types of characters and for converting letters to uppercase and lowercase.
- String is another structured data type. C does not support strings as a data type. But we can use character arrays to represent strings.
- Standard functions printf, puts
- Standard functions scanf, gets
- String manipulation functions => to copy strings, to compare, to compute length, to concatenate