

## MODULE X USING LIBRARY FUNCTIONS - C CHARACTER AND STRING

### MODULE 25 & 26 C++ STL - CHARACTER AND STRING (Template based)

My Training Period:        hours

#### Abilities

- ' Able to understand the fundamentals of string and character.
  - ' Able to find the C standard and non-standard header file resources.
  - ' Able to understand and use the standard and no-standard header files.
  - ' Able to understand and use pre-defined functions.
  - ' Able to manipulate characters and strings by using the pre-defined functions.
- 
- This Module presented just to show you how to use the functions in C Standard Library. Here, we have to know **which functions** are readily available and **which header file** to be included in our program as well as how to write the **proper syntax**.
  - The problem encountered by programmers mostly related to **data type**, the **number** and **order** of the arguments when passing them to the functions and the function **return type** during the function call.
  - The functions used in this Module are from `stdio.h`, `stdlib.h`, `string.h` and `ctype.h` headers.
  - These functions are used heavily in programming for character and string manipulation such as text and string search programs, from small programs, binary or linear search up to big and complex search routines.
  - In C++ these routines easily performed by the Standard Template Library (STL).
  - Remember that these are not a new constructs, but just normal functions **:o)**. Learn how to use them.
  - **gcc**, **g++** and Visual C++ compilation examples are given at the end of this Module.

#### X.1 Introduction

- In programming, solving the real world problems involved the numbers crunching, these numbers including characters. Characters are the fundamental building blocks of the program.
- Every program is composed of a sequence of characters interpreted by the computer as a series of instructions used to accomplish a task.
- A **character constant** is an `int` value represented as a character in single quotes.
- This means that the value of a character constant is the integer value of the character in the machine's character set.
- For example:

'z' represents the integer value of z.  
'\n' represents the integer value of newline.

- A string is a **series of characters** treated as a single unit.
- It may include:
  1. Letters.
  2. Digit.
  3. And various special characters such as +, -, \*, /, \$ and others.
- Or the set of characters lay on your keyboard. For example, every line of the following address is strings.

"Mr. Smith"  
"39, Big Picture Street"  
"Smithsonian, Florida, FL"

- Still remember the relation between array and pointers? A string in C/C++ is an array of characters ending with the null character ('\\0') and can be accessed via a **pointer to the first character**, as you have learned before.
- In declaring a string, it may be assigned to either,
  1. A character array or
  2. A variable of type `char *` (pointer)

- For example:

```
char color[] = "blue";           - an array
char *colorPtr = "blue";       - a pointer
```

- Each line of code initializes a variable to the string "blue".
- The first declaration creates a 5 elements array named `color` containing the characters 'b', 'l', 'u', 'e' and '\\0'.
- The second creates pointer variable `colorPtr` that points to the string "blue" somewhere in memory.
- We also can declare and initialize with initializer list such as:

```
char color[] = {'b', 'l', 'u', 'e', '\\0'};
```

- When declaring a character array to contain a string, the array must be large enough to store the string and its terminating NULL character.
- The previous declaration determines the size of the array automatically based on the number of initializer in the initializer list, which is also its initial value.
- A string can be assigned to an array using `scanf`. For example:

```
char word[20];
...
scanf("%s", word);
```

- The codes will assign a string to character array `word[20]` or string will be stored in an array `word`.
- Note that `word` is an array which is, a pointer, so the `&` is not needed with argument `word`. As you have learned, an **array name** (without bracket) is a **pointer** to the first array element.
- Function `scanf()` will read characters until a **space, newline, or end-of-file** indicator is encountered.
- The string should be no longer than 19 characters to leave room for the terminating NULL character.

## X.2 Character Handling Library

- This library includes several functions that perform useful tests and manipulations of character data.
- Each function receives a **character**, represented as an `int` or EOF as an argument.
- Character handling functions manipulate characters as integers.
- Table X.1 summarizes the functions of the character handling library, in `ctype.h`.

	Function Prototype	Function description
1	<code>int isdigit(int c)</code>	Returns a true value if <code>c</code> is a digit and 0 (false) otherwise.
2	<code>int isalpha(int c)</code>	Returns a true value if <code>c</code> is a letter and 0 otherwise.
3	<code>int isalnum(int c)</code>	Returns a true value if <code>c</code> is a digit or letter, and 0 otherwise.
4	<code>int isxdigit(int c)</code>	Returns a true value if <code>c</code> is a hexadecimal digit character and 0 otherwise.
5	<code>int islower(int c)</code>	Returns a true value if <code>c</code> is a lowercase letter and 0 otherwise.
6	<code>int isupper(int c)</code>	Returns a true value if <code>c</code> is an uppercase letter and 0 otherwise.
7	<code>int tolower(int c)</code>	If <code>c</code> is an uppercase letter, <code>tolower()</code> returns <code>c</code> as a lowercase letter. Otherwise, <code>tolower()</code> returns the argument unchanged.
8	<code>int isspace(int c)</code>	Returns a true value if <code>c</code> is a white space character such as newline('\\n'), space(' '), form feed('\\f'), carriage return('\\r'), horizontal tab('\\t') or vertical tab('\\v') and 0 otherwise.
9	<code>int iscntrl(int c)</code>	Returns a true value if <code>c</code> is a control character and 0 otherwise.

10	<code>int ispunct(int c)</code>	Returns a true value if c is printing character other than a space, a digit, or a letter and 0 otherwise.
11	<code>int isprint(int c)</code>	Returns a true value if c is a printing character including space ( ' '), and 0 otherwise.
12	<code>int isgraph(int c)</code>	Returns a true if c is a printing character other than space ( ' '), and 0 otherwise.

Table X.1: Summary of the character handling library function

- Let explore the program examples, don't forget to include `ctype.h` header file.

```
//Using functions isdigit(), isalpha(), isalnum(), and isxdigit()
//but using C++ :o), cout...
#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>

int main()
{
cout<<"Using functions isdigit(), isalpha(),"<<endl;
cout<<"isalnum(), and isxdigit()"<<endl;
cout<<"-----"<<endl;

cout<<"\nAccording to isdigit():"<<endl;
isdigit('8') ? cout<<"8 is a digit\n" : cout<<"8 is not a digit\n";
isdigit('#') ? cout<<"# is a digit\n" : cout<<"# is not a digit\n";

cout<<"\nAccording to isalpha():"<<endl;
isalpha('A') ? cout<<"A is a letter\n" : cout<<"A is not a letter\n";
isalpha('b') ? cout<<"b is a letter\n" : cout<<"b is not a letter\n";
isalpha('&') ? cout<<"& is a letter\n" : cout<<"& is not a letter\n";
isalpha('4') ? cout<<"4 is a letter\n" : cout<<"4 is not a letter\n";

cout<<"\nAccording to isalnum():"<<endl;
isalnum('A') ? cout<<"A is a digit or a letter\n" : cout<<"A is not a digit or a
letter\n";
isalnum('8') ? cout<<"8 is a digit or a letter\n" : cout<<"8 is not a digit or a
letter\n";
isalnum('#') ? cout<<"# is a digit or a letter\n" : cout<<"# is not a digit or a
letter\n";

cout<<"\nAccording to isxdigit():"<<endl;
isxdigit('F') ? cout<<"F is a hexadecimal\n" : cout<<"F is not a hexadecimal\n";
isxdigit('J') ? cout<<"J is a hexadecimal\n" : cout<<"J is not a hexadecimal\n";
isxdigit('7') ? cout<<"7 is a hexadecimal\n" : cout<<"7 is not a hexadecimal\n";
isxdigit('$') ? cout<<"$ is a hexadecimal\n" : cout<<"$ is not a hexadecimal\n";
isxdigit('f') ? cout<<"f is a hexadecimal\n" : cout<<"f is not a hexadecimal\n";

system("pause");
return 0;
}
```

**Output:**

- Program example #2:

```
//Using functions islower(), isupper(), tolower(), toupper()
//using C++, cout...
#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>

int main()
{

cout<<"Using functions islower(), isupper(),"<<endl;
cout<<"tolower(), toupper()"<<endl;
cout<<"-----"<<endl;

cout<<"\nAccording to islower():"<<endl;
islower('p') ? cout<<"p is a lowercase letter\n" : cout<<"p is not a lowercase
letter\n";
islower('P') ? cout<<"P is a lowercase letter\n" : cout<<"P is not a lowercase
letter\n";
islower('5') ? cout<<"5 is a lowercase letter\n" : cout<<"5 is not a lowercase
letter\n";
islower('!') ? cout<<"! is a lowercase letter\n" : cout<<"! is not a lowercase
letter\n";

cout<<"\nAccording to isupper():"<<endl;
isupper('D') ? cout<<"D is a uppercase letter\n" : cout<<"D is not a uppercase
letter\n";
isupper('d') ? cout<<"d is a uppercase letter\n" : cout<<"d is not a uppercase
letter\n";
isupper('8') ? cout<<"8 is a uppercase letter\n" : cout<<"8 is not a uppercase
letter\n";
isupper('$') ? cout<<"$ is a uppercase letter\n" : cout<<"$ is not a uppercase
letter\n";

cout<<"\nConversion..."<<endl;
cout<<"u converted to uppercase is "<<(char)toupper('u')<<endl;
cout<<"7 converted to uppercase is "<<(char)toupper('7')<<endl;
cout<<"$ converted to uppercase is "<<(char)toupper('$')<<endl;
cout<<"L converted to lowercase is "<<(char)tolower('L')<<endl;
system("pause");
return 0;
}
```

**Output:**

- Program example #3:

```
//using functions isspace(), iscntrl(), ispunct(),
//isprint(), isgraph()
#include <iostream.h>
#include <stdlib.h>
#include <ctype.h>

int main()
{
cout<<"using functions isspace(), iscntrl(),"<<endl;
cout<<"ispunct(), isprint(), isgraph()"<<endl;
cout<<"-----"<<endl;

cout<<"According to isspace(): "<<endl;
isspace('\n') ? cout<<"Newline is a whitespace character\n" : cout<<"Newline is
not a whitespace character\n";
isspace('\t') ? cout<<"Horizontal tab is a whitespace character\n" :
cout<<"Horizontal tab is not a whitespace character\n";
isspace('%') ? cout<<"% is a whitespace character\n" : cout<<"% is not a
whitespace character\n";

cout<<"\nAccording to iscntrl(): "<<endl;
iscntrl('\n') ? cout<<"Newline is a control character\n" : cout<<"Newline is not a
control character\n";
iscntrl('$') ? cout<<"$ is a control character\n" : cout<<"$ is not a control
character\n";

cout<<"\nAccording to ispunct(): "<<endl;
ispunct('y') ? cout<<"y is a punctuation character\n" : cout<<"y is not a
punctuation character\n";
ispunct('\\') ? cout<<"\ ' is a punctuation character\n" : cout<<"\ ' is not a
punctuation character\n";
ispunct('\"') ? cout<<"\" is a punctuation character\n" : cout<<"\" is not a
punctuation character\n";

cout<<"\nAccording to isprint(): "<<endl;
isprint('$') ? cout<<"$ is a printing character\n" : cout<<"$ is not a printing
character\n";
isprint('\a') ? cout<<"Alert is a printing character\n" : cout<<"Alert is not a
printing character\n";

cout<<"\nAccording to isgraph(): "<<endl;
isgraph('Q') ? cout<<"Q is a printing character other than a space\n" : cout<<"Q
is not a printing character other than a space\n";
isgraph(' ') ? cout<<"Space is a printing character other than a space\n":
cout<<"Space is not a printing character other than a space\n";
system("pause");
return 0;
}
```

**Output :**

### X.3 String Conversion Functions

- These functions are from the general utilities library, `stdlib.h` header file.
- They convert strings of digits to integer and floating-point values.
- Table X.2 summarizes the string conversion functions.
- Note the use of `const` to declare variable `nPtr` in the function headers (read from right to the left as `nPtr` is a pointer to a character constant).
- `const` declares that the argument values will not be modified during the program execution.

Function prototype	Function description
<code>double (const char *nPtr)</code>	Converts the string <code>nPtr</code> to double.
<code>int (const char *nPtr)</code>	Converts the string <code>nPtr</code> to int.
<code>long (const char *nPtr)</code>	Converts the string <code>nPtr</code> to long int.
<code>double (const char *nPtr, char **endPtr)</code>	Converts the string <code>nPtr</code> to double.
<code>long (const char *nPtr, char **endPtr, int base)</code>	Converts the string <code>nPtr</code> to long.
<code>unsigned long (const char *nPtr, char **endPtr, int base)</code>	Converts the string <code>nPtr</code> to unsigned long.

Table X.2: Summary of the string conversion functions of the general utilities library.

- The following are the program examples for this section.

```
//using atof() - converting string to double
#include <stdio.h>
#include <stdlib.h>

int main()
{
    double dou;
        ;

    printf("Using atof() - converting string to double\n");
    printf("-----\n\n");
    printf("The string \"95.0\" when converted to double is %.3f\n", dou);
    printf("The converted value, %.3f divided by 2 is %.3f\n", dou, dou / 2.0);

    system("pause");
    return 0;
}
```

**Output:**

- atoi() program example.

```
//using atoi() - converting string to integer
#include <stdio.h>
#include <stdlib.h>

int main()
{
    int i;
        ;

    printf("Using atoi() - converting string to integer\n");
    printf("-----\n\n");
    printf("The string \"1234\" converted to int is %d\n", i);
    printf("The converted value %d minus 123 is %d\n", i, i - 123);
    system("pause");
    return 0;
}
```

**Output:**

- atol() program example.

```
//Using atol() - converting string to long
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long newlong;
        ;

    printf("Using atol() - converting string to long\n");
    printf("-----\n\n");
    printf("The string \"123456\" converted to long int is %ld\n", newlong);
    printf("The converted value, %ld divided by 2 is %ld\n", newlong, newlong / 2);
    system("pause");
    return 0;
}
```

**Output:**

- strtod() – converting string to double with 2 arguments.

```
//using strtod() - string to double
#include <stdio.h>
#include <stdlib.h>

int main()
{
    double p;
    char *thestring = "41.2% sample string";
    char *thestringPtr;

    ;

    printf("Using strtod() - converting string to double...\n");
    printf("-----\n\n");
    printf("The string \"%s\" is converted to the\n", thestring);
    printf("double value %.2f and the string \"%s\" \n", p, thestringPtr);
    system("pause");
    return 0;
}
```

**Output:**

- strtol() – converting string to long with 3 arguments program example.

```
//Using strtol()-converting string to
//long with 3 arguments
#include <stdio.h>
#include <stdlib.h>

int main()
{
    long x;
    char *thestring = "-1234567abc", *remainderPtr;

    ;

    printf("Using strtol() - converting string to long,\n");
    printf("          3 arguments...\n");
    printf("-----\n\n");
    printf("The original string is \"%s\"\n", thestring);
    printf("The converted value is %ld\n", x);
    printf("The remainder of the original string is \"%s\"\n", remainderPtr);
    printf("The converted value, %ld plus 567 is %ld\n", x, x + 567);
    system("pause");
    return 0;
}
```

**Output:**

- `strtoul()` - converting string to unsigned long with 3 argument program example.

```
//Using strtoul() - converting string to
//unsigned long with 3 arguments
#include <stdio.h>
#include <stdlib.h>

int main()
{
    unsigned long x;
    char *thestring = "1234567def", *remainderPtr;

    ;

    printf("Using strtoul() - converting string to\n");
    printf("    unsigned long, 3 arguments\n");
    printf("-----\n\n");
    printf("The original string is \"%s\"\n", thestring);
    printf("The converted value is %lu\n", x);
    printf("The remainder of the original string is \"%s\"\n", remainderPtr);
    printf("The converted value, %lu minus 567 is %lu\n", x, x - 567);
    system("pause");
    return 0;
}
```

**Output:**

#### X.4 Standard Input/Output Library Functions

- These functions are from the standard input/output library, `stdio.h`.
- Are specifically for manipulating characters and string data.
- Table X.3 summarizes these functions and their usage.

Function prototype	Function description
<code>int (void)</code>	Input the next character from the standard input (keyboard) and return it as an integer.
<code>char * (char *s)</code>	Input characters from the standard input (keyboard) into the array <b>s</b> until a newline or end-of-file character is encountered. A terminating NULL character is appended to the array.
<code>int (int c)</code>	Print the character stored in <b>c</b> .
<code>int (const char *s)</code>	Print the string <b>s</b> followed by a newline character.
<code>int (char *s, const char *format, ...)</code>	Equivalent to <code>printf()</code> except the output is stored in the array <b>s</b> instead of printing on the screen.

int (char *s, const char *format, ...)	Equivalent to scanf ( ) except the input is read from the array <b>s</b> instead of reading from the keyboard.
---	--

Table X.3 : The standard input/output library character and string functions

- Program examples functions from the `stdio.h`, beginning with `gets ( )` and `putchar ( )`.

```
//Using gets() and putchar()
#include <stdio.h>
#include <stdlib.h>

//function prototype...
void reverse(char *);

int main()
{
    //an array for storing the string...
    char sentence[80];

    printf("Using gets() and putchar()\n");
    printf("-----\n");
    //prompt for user input...
    printf("Enter a line of text:\n");
    ;

    printf("\nThe line printed backward is:\n");
    //reverse() function call...
    reverse(sentence);
    printf("\n");
    system("pause");
    return 0;
}

void reverse(char *s)
{
    //test if nothing entered...
    if(s[0] == '\0')
        return;
    //if something entered...
    else
    {
        reverse(&s[1]);
        ;
    }
}
```

**Output :**

- `getchar ( )` and `puts ( )`.

```
//using getchar() and puts()
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char c, sentence[80];
    int i = 0;

    printf("Using getchar() and puts()\n");
    printf("-----\n");
    puts("Enter a line of text: ");
```

```

//while iteration/loop...
while (( c
        ) != '\n')
    sentence[i++] = c;
    //insert NULL at the end of string
    sentence[i] = '\0';
    puts("\nThe line of text entered was: ");
    puts(sentence);
    system("pause");
    return 0;
}

```

**Output:**

- sprintf().

```

//Using sprintf()
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char s[80];
    int x;
    float y;

    printf("Using sprintf()\n");
    printf("-----\n");
    printf("Enter an integer and a float, separated by space: \n");
    scanf("%d%f", &x, &y);

    printf("\n%s\n%s\n",
           "The formatted output stored in array s is: ", s);
    system("pause");
    return 0;
}

```

**Output:**

- sscanf().

```

//Using sscanf()
#include <stdio.h>
#include <stdlib.h>

int main()
{
    char s[] = "31298    87.375";
    int x;
    float y;
}

```

```

printf("Using sscanf()\n");
printf("-----\n");
;
printf("array, s[] = 31298      87.375\n");
printf("\n%s\n%s%6d\n%s%8.3f\n",
      "The values stored in character array s are: ",
      "Integer: ", x, "Float: ", y);
system("pause");
return 0;
}

```

**Output:**

## X.5 String Manipulation Functions of The String Handling Library

- These functions are for:
  1. Manipulating string data.
  2. Comparing strings.
  3. Searching strings for characters and other strings.
  4. Tokenizing strings (separating strings into logical pieces).
  5. Determining the length of strings.
- We call these functions from `string.h` header file.
- Table X.4 summarizes these functions.

Function prototype	Function description
<code>char * (char *s1, const char *s2)</code>	Copies the string <code>s2</code> into the array <code>s1</code> . The value of <code>s1</code> is returned.
<code>char * (char *s1, const char *s2, size_t n)</code>	Copies at most <code>n</code> characters of the string <code>s2</code> into the array <code>s1</code> . The value of <code>s1</code> is returned.
<code>char * (char *s1, const char *s2)</code>	Appends the string <code>s2</code> to the array <code>s1</code> . The first character of <code>s2</code> overwrites the terminating NULL character of <code>s1</code> . The value of <code>s1</code> is returned.
<code>char * (char *s1, const char *s2, size_t n)</code>	Appends at most <code>n</code> characters of string <code>s2</code> to array <code>s1</code> . The first character of <code>s2</code> overwrites the terminating NULL character of <code>s1</code> . The value of <code>s1</code> is returned.

Table X.4: The string manipulation functions of the string handling library

- Let explore the program examples, beginning with `strcpy()` and `strncpy()`.

```

//Using strcpy() and strncpy()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char x[] = "Yo! Happy Birthday to me";
    char y[25], z[15];

    printf("Using strcpy() and strncpy()\n");
    printf("-----\n");
    printf("The string in array x is: %s\n", x);
    printf("The string in array y is: %s\n",      );
}

```

```

        ;
z[14] = '\0';

printf("Only 14 characters ....\n", z);
printf("The string in array z is: %s\n", z);
system("pause");
return 0;
}

```

**Output:**

```

//Using strcat() and strncat()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

    printf("Using strcat() and strncat()\n");
    printf("-----\n");
    printf("s1 = %s\ns2 = %s\n", s1, s2);
    printf("\nstrcat (s1, s2) = %s\n",          );
    printf("strncat (s1, s2, 6) = %s\n",          );
    printf("strcat(s3, s1) = %s\n", strcat(s3, s1));
    system("pause");
    return 0;
}

```

**Output:**

## X.6 Comparison Functions Of The String Handling Library

- Let explore the string comparison functions, `strcmp()` and `strncmp()`, of the string handling library. Table X.5 is the summary of the functions and follow by the program examples.
- For these sections, how the computers know that one particular letter comes before another?
- **All characters** are represented inside the computer as **numeric codes**, when the computer compares two strings, it actually compares the numeric codes of the characters in the strings.
- There are three popular coding schemes for character representation:
  1. ASCII – American Standard Code for Information Interchange.
  2. EBCDIC – Extended Binary Coded Decimal Interchange Code.
  3. Unicode.

- ASCII, EBCDIC and Unicode are called **character codes** or **character sets**.
- String and character manipulations actually involve the manipulation of the appropriate numeric codes and not the characters themselves.
- These explain the interchangeability of characters and small integers in C/C++.

Function prototype	Function description
int (const char *s1, const char *s2)	Compares the string <b>s1</b> to the string <b>s2</b> . The function returns 0, less than 0, or greater than 0 if <b>s1</b> is equal to, less than, or greater than <b>s2</b> , respectively.
int (const char *s1, const char *s2, size_t n)	Compares up to <b>n</b> characters of the string <b>s1</b> to the string <b>s2</b> . The function returns 0, less than 0, or greater than 0 if <b>s1</b> is equal to, less than, or greater than <b>s2</b> , respectively.

Table X.5: The string comparison functions of the string handling library

```
//Using strcmp() and strncmp()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char * s1 = "Happy New Year";
    char *s2 = "Happy New Year";
    char *s3 = "Happy Birthday";

    printf("Using strcmp() and strncmp()\n");
    printf("-----\n");

    printf("s1 = %s\n", s1);
    printf("s2 = %s\n", s2);
    printf("s3 = %s\n", s3);
    printf("\nstrcmp(s1, s2) = %2d\n", strcmp(s1, s2));
    printf("strcmp(s1, s3) = %2d\n", strcmp(s1, s3));
    printf("strcmp(s3, s1) = %2d\n", strcmp(s3, s2));

    printf("\nstrncmp(s1, s3, 6) = %2d\n", strncmp(s1, s3, 6));
    printf("strncmp(s1, s3, 7) = %2d\n", strncmp(s1, s3, 7));
    printf("strncmp(s1, s1, 7) = %2d\n", strncmp(s1, s3, 7));
    system("pause");
    return 0;
}
```

**Output:**

## X.7 Search Functions Of The String handling Library

- Used to search strings for characters and other strings. Table X.6 is the summary of these functions and followed by program examples.

Function prototype	Function description
char * (const char *s, int c)	Locates the first occurrence of character <b>c</b> in string <b>s</b> . If <b>c</b> is found, a pointer to <b>c</b> in <b>s</b> is returned. Otherwise a NULL pointer is returned.
size_t (const char *s1, const char *s2)	Determines and returns the length of the initial segment of string <b>s1</b> consisting of characters not contained in string <b>s2</b> .
size_t (const char *s1, const char *s2)	Determines and returns the length of the initial segment of string <b>s1</b> consisting only of characters contained in string <b>s2</b> .
char * (const char *s1, const char *s2)	Locates the first occurrence in string <b>s1</b> of any character in string <b>s2</b> . If a character from string <b>s2</b> is found, a pointer to the character in string <b>s1</b> is returned. Otherwise a NULL pointer is returned.
char * (const char *s, int c)	Locates the last occurrence of <b>c</b> in string <b>s</b> . If <b>c</b> is found, a pointer to <b>c</b> in string <b>s</b> is returned. Otherwise is a NULL pointer is returned.
char * (const char *s1, const char *s2)	Locates the first occurrence in string <b>s1</b> of string <b>s2</b> . If the string is found, a pointer to the string in <b>s1</b> is returned. Otherwise a NULL pointer is returned.
char * (char *s1, const char *s2)	A sequence of calls to <code>strtok</code> breaks string <b>s1</b> into “tokens”, logical pieces such as words in a line of text, separated by characters contained in string <b>s2</b> . The first call contains <b>s1</b> 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.

Table X.6: String manipulation functions of the string handling library.

```
//Using strchr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char *string = "This is a test statement, testing! ";
    char character1 = 'e', character2 = 'z';

    printf("          Using strchr()\n");
    printf("          -----\n");

    if (
                != NULL)
        printf("\'%c\' was found in \"%s\".\n", character1, string);
    else
        printf("\'%c\' was not found in \"%s\".\n", character1, string);

    if(strchr(string, character2) != NULL)
        printf("\'%c\' was found in \"%s\".\n", character2, string);
    else
        printf("\'%c\' was not found in \"%s\".\n", character2, string);
    system("pause");
    return 0;
}
```

**Output:**

```
//Using strtok()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
```

```

int main()
{
    char *string1 = "The value is 3.14159";
    char *string2 = "1234567890";

    printf("        Using strchr()\n");
    printf("        -----\n");
    printf("string1 = %s\n", string1);
    printf("string2 = %s\n", string2);
    printf("\nThe length of the initial segment of string1\n");
    printf("not containing characters from string2 = %u",          );
    printf("\n");
    system("pause");
    return 0;
}

```

**Output:**

```

//Using strpbrk()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char *string1 = "This is a test statement";
    char *string2 = "search";

    printf("        Using strpbrk()\n");
    printf("        -----\n");
    printf("In \"%s\" string, a character \'%c\'", string2,
*strpbrk(string1, string2));
    printf("is the first character to appear in\n\"%s\"\n", string1);
    system("pause");
    return 0;
}

```

**Output:**

```

//Using strchr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char *string1 = "A zoo has many animals including birds";
    int c = 'm';

    printf("        Using strchr()\n");
    printf("        -----\n");
    printf("string1 = %s\n", string1);
    printf("\nThe remainder of string1 beginning with the\n");
    printf("last occurrence of character \'%c\'", c);
}

```

```

        printf("\nis: %s\n",          );
        system("pause");
        return 0;
    }

```

**Output:**

```

//Using strstr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char *string1 = "The initial value is 3.14159";
    char *string2 = "aehilsTuv";

    printf("        Using strstr()\n");
    printf("        -----\n");
    printf("string1 = %s\n", string1);
    printf("string2 = %s\n", string2);
    printf("\nThe length of the initial segment of string1\n");
    printf("containing only characters from string2 is = %u\n",
        );
    system("pause");
    return 0;
}

```

**Output:**

```

//Using strstr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char *string1 = "abcdefgabcdefgabcdefg";
    char *string2 = "defg";

    printf("        Using strstr()\n");
    printf("        -----\n");
    printf("string1 = %s\n", string1);
    printf("string2 = %s\n", string2);
    printf("\nThe remainder of string1 beginning with the");
    printf("\nfirst occurrence of string2 is: %s\n",
        strstr(string1, string2));
    system("pause");
    return 0;
}

```

**Output:**

```
//Using strtok()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char string[] = "Is this sentence has 6 tokens?";
    char *tokenPtr;

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

    tokenPtr = " ";
    while (tokenPtr != NULL)
    {
        printf("%s\n", tokenPtr);
        tokenPtr = strtok(NULL, " ");
    }
    system("pause");
    return 0;
}
```

**Output:**

## X.8 Memory Functions Of The String Handling Library

- These functions are for:
  1. Manipulating blocks of memory.
  2. Comparing blocks of memory.
  3. Searching blocks of memory.
- The functions treat blocks of memory as character arrays and can manipulate any block of data.
- Table X.7 summarizes the memory functions of the string handling library; the term object refers to a block of data.

Function prototype	Function description
void * (void *s1, const void *s2, size_t n)	Copies <b>n</b> characters from the object pointed to by <b>s2</b> into the object pointed to by <b>s1</b> . A pointer to the resulting object is returned.
void * (void *s1, const void *s2, size_t n)	Copies <b>n</b> characters from the object pointed to by <b>s2</b> into the object pointed to by <b>s1</b> . The copy is performed as if the characters are first copied from the object pointed to by <b>s2</b> into temporary array, then from the temporary array into the object pointed to by <b>s1</b> . A pointer to the resulting object is returned.
int (const void *s1, const void *s2, size_t n)	Compares the first <b>n</b> characters of the objects pointed to by <b>s1</b> and <b>s2</b> . The function return 0, less than 0, or greater than 0 if <b>s1</b> is equal to, less than, or greater than <b>s2</b> .
void * (const void *s, int c, size_t n)	Locates the first occurrence of <b>c</b> (converted to <b>unsigned char</b> ) in the first <b>n</b> characters of the object pointed to by <b>s</b> . If <b>c</b> is found, a pointer to <b>c</b> in the object is returned. Otherwise <b>NULL</b> is returned.
void * (void *s, int c, size_t n)	Copies <b>c</b> (converted to <b>unsigned char</b> ) into the first <b>n</b> characters of the object pointed to by <b>s</b> . A pointer to the result is returned.

Table X.7: The memory functions of the string handling library

- Let explore the program examples.

```
//Using memcpy()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char s1[20], s2[] = "Copying this string into s1";

    ;

    printf("    Using memcpy()\n");
    printf("    -----\n");
    printf("s1[20] = ?\n", s1);
    printf("s2[] = %s\n", s2);
    printf("\nAfter s2 is copied into s1 with memcpy(),\n");
    printf("using memcpy(s1, s2, 17)\n");
    printf("\ns1 contains \"%s\"\n", s1);
    system("pause");
    return 0;
}
```

**Output:**

```
//Using memmove()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char x[] = "My home is home sweet home";
```

```

printf("      Using memmove()\n");
printf("      -----\n");
printf("The string in array x before memmove() is: \n%s", x);
printf("\nThe string in array x after memmove() using \n");
printf("memmove(x, &x[7], 12) is:\n %s\n",          );
system("pause");
return 0;
}

```

**Output:**

```

//Using memcmp()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char s1[] = "ABCDEFGHGIJK", s2[] = "ABCDXYZPQR";

    printf("Using memcmp()\n");
    printf("-----\n");
    printf("s1 = %s\n", s1);
    printf("s2 = %s\n", s2);
    printf("\nmemcmp(s1, s2, 4) = %2d\n",          );
    printf("memcmp(s1, s2, 7) = %2d\n", memcmp(s1, s2, 7));
    printf("memcmp(s2, s1, 7) = %2d\n", memcmp(s2, s1, 7));
    system("pause");
    return 0;
}

```

**Output:**

```

//Using memchr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char *s = "This is a test string";
    char p = 'e';

    printf("Using memchr()\n");
    printf("-----\n");
    printf("char p = \\'e\'\n");
    printf("s = %s\n", s);
    printf("\nThe remainder of string s, after character \\'%c\'", p);
}

```

```

printf("\nis found, using strchr(s, p, 15)");
printf("\nis \"%s\"", );
system("pause");
return 0;
}

```

**Output:**

```

//Using memset()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char string[40] = "AAAAAAAABBBBBBBBCCCCCCCC";

    printf("Using memset()\n");
    printf("-----\n");
    printf("string = %s\n", string);
    printf("string after memset(string, 'b', 15) =\n%s\n", );
    system("pause");
    return 0;
}

```

**Output:**

## X.9 Other Functions Of The String Handling Library

- The remaining functions of the string handling library are `strerror()` and `strlen()`.
- Function `strerror()` takes an error number and creates an error message string. A pointer to the string is returned.
- Function `strlen()` takes a string as an argument, and returns the number of characters in a string, the terminating `NULL` character is not included in the length.
- The functions are summarized in table X.8.

Function prototype	Function description
<code>char * (int errornum)</code>	Maps <code>errornum</code> into a full text string in a system dependent manner. A pointer to the string is returned.
<code>size_t (const char *s)</code>	Determines the length of string <code>s</code> . The number of characters preceding the terminating <code>NULL</code> character is returned.

Table X.8: The string manipulation functions of the string handling library

- Program example.

```

//Using strerror()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    printf("strerror() - string errors\n");
    printf("-----\n");
    printf("strerror(1)->%s", strerror(1));
    printf("strerror(2)->%s", strerror(2));
    printf("strerror(3)->%s", strerror(3));
    printf("strerror(4)->%s", strerror(4));
    printf("strerror(5)->%s", strerror(5));
    printf("strerror(6)->%s", strerror(6));
    printf("strerror(7)->%s", strerror(7));
    printf("strerror(8)->%s", strerror(8));
    printf("strerror(9)->%s", strerror(9));
    printf("strerror(9)->%s", strerror(9));
    printf("strerror(10)->%s", strerror(10));
    system("pause");
    return 0;
}

```

**Output :**

- Program example compiled using VC++ .Net.

```

//Using strchr()
#include <cstdio>
#include <cstring>

int main()
{
    char *string1 = "A zoo has many animals including birds";
    int c = 'm';

    printf("    Using strchr()\n");
    printf("    -----\n");
    printf("string1 = %s\n", string1);
    printf("\nThe remainder of string1 beginning with the\n");
    printf("last occurrence of character \'%c\'", c);
    printf("\nis: %s\n", strchr(string1, c));
    return 0;
}

```

**Output :**

- For C++ character and string manipulations that use the Standard Template Library (STL), please refer to Module 25 and 26.
- Program examples compiled using **g++** (C++) and **gcc** (C).

```

/*****ctystring.cpp*****/
//Using sprintf()
#include <cstdio>
using namespace std;

int main()
{
    char s[80];
    int x;
    float y;

    printf("Using sprintf()\n");
    printf("-----\n");
    printf("Enter an integer and a float, separated by space: \n");
    scanf("%d%f", &x, &y);
    sprintf(s, "Integer:%6d\nFloat:%8.2f", x, y);
    printf("\n%s\n%s\n", "The formatted output stored in array s is: ", s);
    return 0;
}

```

```

[bodo@bakawali ~]$ g++ ctystring.cpp -o ctystring
[bodo@bakawali ~]$ ./ctystring

```

```

Using sprintf()
-----
Enter an integer and a float, separated by space:
100 33.354

The formatted output stored in array s is:
Integer: 100
Float: 33.35

```

```

/****ctstring2.c, using memcpy()*/
#include <stdio.h>
#include <string.h>

int main()
{
    char s1[20], s2[] = "Copying this string into s1";
    memcpy(s1, s2, 17);
    printf("      Using memcpy()\n");
    printf("      -----\n");
    printf("s1[20] = ?\n", s1);
    printf("s2[] = %s\n", s2);
    printf("\nAfter s2 is copied into s1 with memcpy(),\n");
    printf("using memcpy(s1, s2, 17)\n");
    printf("\ns1 contains \"%s\"\n", s1);
    return 0;
}

```

```

[bodo@bakawali ~]$ gcc ctstring2.c -o ctstring2
[bodo@bakawali ~]$ ./ctstring2

```

```

      Using memcpy()
      -----
s1[20] = ?
s2[] = Copying this string into s1

After s2 is copied into s1 with memcpy(),

```

```
using memcpy(s1, s2, 17)
```

```
s1 contains "Copying this stri"
```

```
/******cstr.c*****  
/*Using functions isdigit(), isalpha(), isalnum(), and isxdigit()*/  
#include <stdio.h>  
#include <stdlib.h>  
#include <ctype.h>  
  
int main()  
{  
printf("Using functions isdigit(), isalpha(),\n");  
printf("isalnum(), and isxdigit()\n");  
printf("-----\n");  
  
printf("\nAccording to isdigit():\n");  
isdigit('7') ? printf("7 is a digit\n") : printf("7 is not a digit\n");  
isdigit('$') ? printf("$ is a digit\n") : printf("$ is not a digit\n");  
  
printf("\nAccording to isalpha():\n");  
isalpha('B') ? printf("B is a letter\n") : printf("B is not a letter\n");  
isalpha('b') ? printf("b is a letter\n") : printf("b is not a letter\n");  
isalpha('&') ? printf("& is a letter\n") : printf("& is not a letter\n");  
isalpha('4') ? printf("4 is a letter\n") : printf("4 is not a letter\n");  
  
printf("\nAccording to isalnum():\n");  
isalnum('A') ? printf("A is a digit or a letter\n") : printf("A is not a digit or a  
letter\n");  
isalnum('8') ? printf("8 is a digit or a letter\n") : printf("8 is not a digit or a  
letter\n");  
isalnum('#') ? printf("# is a digit or a letter\n") : printf("# is not a digit or a  
letter\n");  
  
printf("\nAccording to isxdigit():\n");  
isxdigit('F') ? printf("F is a hexadecimal\n") : printf("F is not a hexadecimal\n");  
isxdigit('J') ? printf("J is a hexadecimal\n") : printf("J is not a hexadecimal\n");  
isxdigit('7') ? printf("7 is a hexadecimal\n") : printf("7 is not a hexadecimal\n");  
isxdigit('$') ? printf("$ is a hexadecimal\n") : printf("$ is not a hexadecimal\n");  
isxdigit('f') ? printf("f is a hexadecimal\n") : printf("f is not a hexadecimal\n");  
  
return 0;  
}
```

```
[bodo@bakawali ~]$ gcc cstr.c -o cstr
```

```
[bodo@bakawali ~]$ ./cstr
```

```
Using functions isdigit(), isalpha(),  
isalnum(), and isxdigit()  
-----
```

```
According to isdigit():  
7 is a digit  
$ is not a digit
```

```
According to isalpha():  
B 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  
J is not a hexadecimal  
7 is a hexadecimal  
$ is not a hexadecimal  
f is a hexadecimal
```

```
-----oO-----
```

## Further reading and digging:

1. Check the best selling C/C++ books at Amazon.com.
2. Module G (Story) and Module M (Microsoft implementation) for Multibytes, Unicode characters and Localization.