To:
Tenouk

MODULE X
USING C LIBRARY - THE C CHARACTER AND STRING 1

MODULE 25 & 26
THE C++ STL - CHARACTERS AND STRINGS
(Template based)

My Training Period: hours

WARNING: Many of the strings and characters library functions have a buffer overflow issue, if not used properly. There are a secure version of these functions already available and standard complied. If you use these functions in your programs, compiled using newer or standard complied compilers, expect warnings during the compilation. For the secure version, please refer to your compiler documentation. However some of the secure version functions already used in C Lab worksheet tutorials. The compiler used is Visual C++ 2005 Express Edition.

C string and character 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 pre-defined functions in C Standard Library. Here, we have to know which functions are 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** header files.
- 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. 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 a string.

  "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.

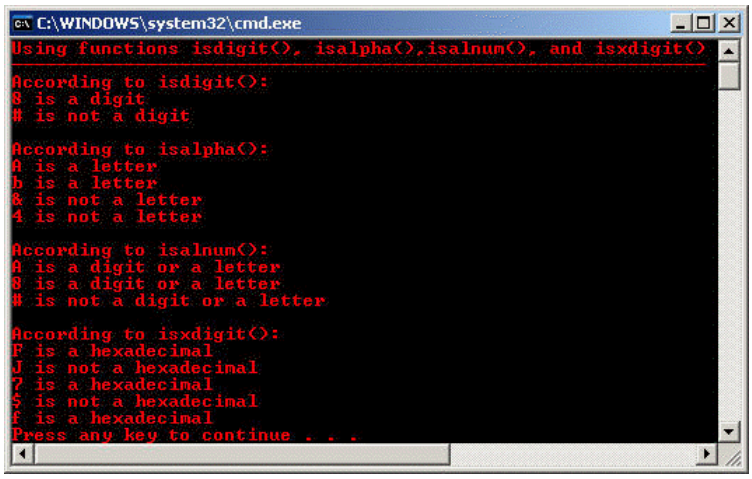
	Function Prototype	Function description
1	int isdigit(int c)	Returns a true value if c is a digit and 0 (false) otherwise.
2	int isalpha(int c)	Returns a true value if c is a letter and 0 otherwise.
3	int isalnum(int c)	Returns a true value if c is a digit or letter, and 0 otherwise.
4	int isxdigit(int c)	Returns a true value if c is a hexadecimal digit character and 0 otherwise.
5	int islower(int c)	Returns a true value if c is a lowercase letter and 0 otherwise.
6	int isupper(int c)	Returns a true value if c is an uppercase letter and 0 otherwise.
7	int tolower(int c)	If c is an uppercase letter, tolower() returns c as a lowercase letter. Otherwise, tolower() returns the argument unchanged.
8	int isspace(int c)	Returns a true value if c 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	int iscntrl(int c)	Returns a true value if c is a control character and 0 otherwise.
10	int ispunct(int c)	Returns a true value if c is printing character other than a space, a digit, or a letter and 0 otherwise.
11	int isprint(int c)	Returns a true value if c is a printing character including space (' '), and 0 otherwise.
12	int isgraph(int c)	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 isdigit(), isalpha(), isalnum(), and isxdigit() functions
#include <stdio.h>
#include <ctype.h>

int main()
{
   printf("Using functions isdigit(), isalpha(),");
   printf("isalnum(), and isxdigit()\n");
   printf("-------------------------------------------------------------");

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

   printf("\nAccording to isalpha():\n");
   isalpha('A') ? printf("A is a letter\n") : printf("A 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;
}
```

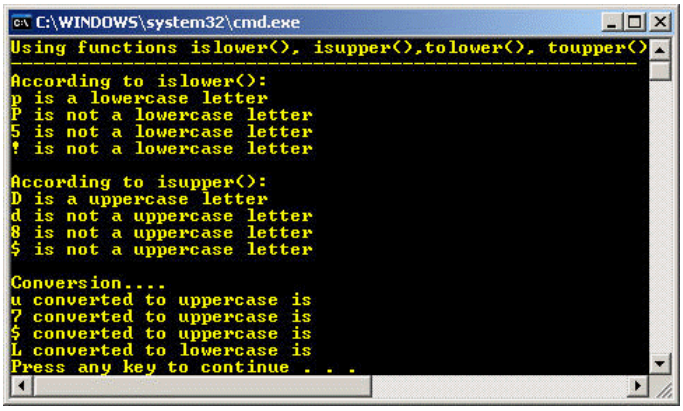
Output:



- Program example #2:

```
// using functions islower(), isupper(), tolower(), toupper()
#include <stdio.h>
#include <ctype.h>

int main()
{
   printf("Using functions islower(), isupper(),");
   printf("tolower(), toupper()\n");
   printf("-------------------------------------------------------");

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

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

   printf("\nConversion....\n");
   printf("u converted to uppercase is \n",(char)toupper('u'));
   printf("7 converted to uppercase is \n",(char)toupper('7'));
   printf("$ converted to uppercase is \n",(char)toupper('$'));
   printf("L converted to lowercase is \n",(char)tolower('L'));

   return 0;
}
```

Output:

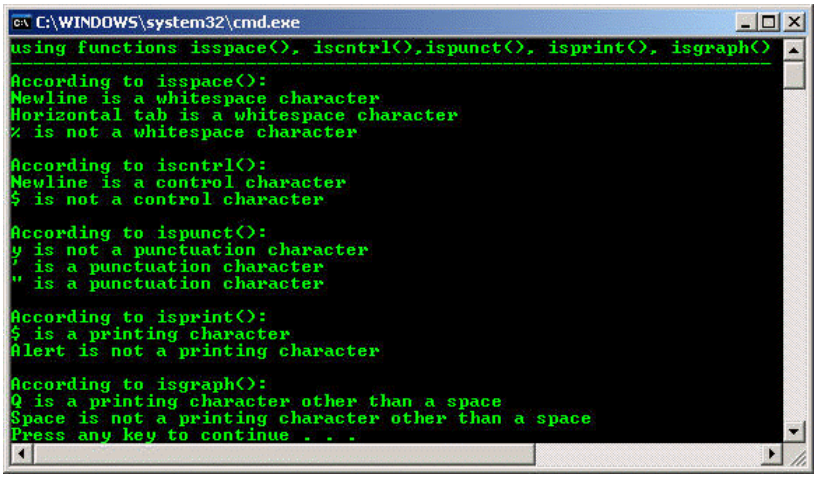


- Program example #3:

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

int main()
{
   printf("using functions isspace(), iscntrl(),");
   printf("ispunct(), isprint(), isgraph()\n");
   printf("-------------------------------------------------------------------\n");

   printf("According to isspace():\n");
   isspace('\n') ? printf("Newline is a whitespace character\n") : printf("Newline is not a
whitespace character\n");
   isspace('\t') ? printf("Horizontal tab is a whitespace character\n") : printf("Horizontal tab is not
a whitespace character\n");
```

```
    isspace('%') ? printf("%% is a whitespace character\n") : printf("%% is not a whitespace character\n");

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

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

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

    printf("\nAccording to isgraph():\n");
    isgraph('Q') ? printf("Q is a printing character other than a space\n") : printf("Q is not a
printing character other than a space\n");
    isgraph(' ') ? printf("Space is a printing character other than a space\n") : printf("Space is not
a printing character other than a space\n");

    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.

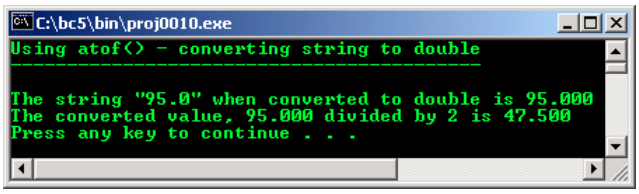
Function prototype	Function description
double atof(const char *nPtr)	Converts the string nPtr to double.
int atoi(const char *nPtr)	Converts the string nPtr to int.
long atol(const char *nPtr)	Converts the string nPtr to long int.
double strtod(const char *nPtr, char **endPtr)	Converts the string nPtr 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.

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

- The following are program examples for this section.

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

int main()
{
    double  dou;
    dou = atof("95.0");
    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);
    return 0;
}
```

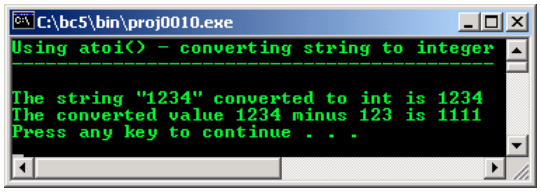
Output:



- The atoi() program example.

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

int main()
{
      int  i;
      i = atoi("1234");
      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);
      return 0;
}
```

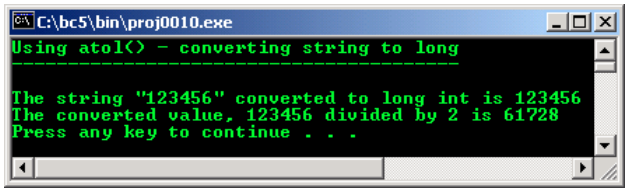
Output:



- The atol() program example.

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

int main()
{
      long newlong;
      newlong = atol("123456");
      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);
      return 0;
}
```

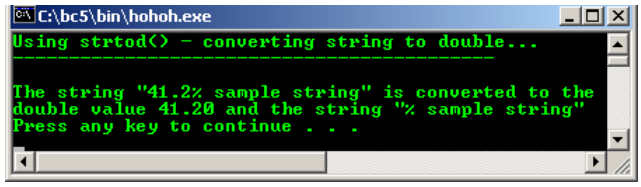
Output:



- The 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;
      p = strtod(thestring, &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);
      return 0;
}
```

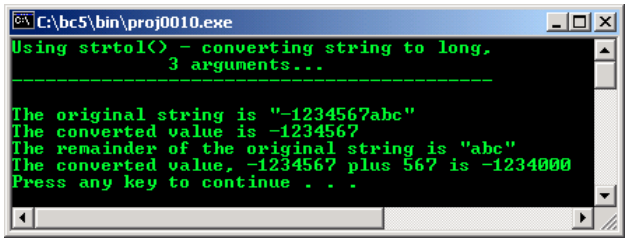
Output:



- The 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;
    x = strtol(thestring, &remainderPtr, 0);
    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);
    return  0;
}
```

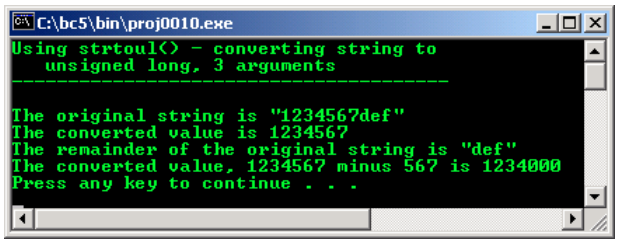
Output:



- The 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;
    x = strtoul(thestring, &remainderPtr, 0);
    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);
    return  0;
}
```

Output:



Further C string library related reading:

1. Check the best selling C / C++ books at Amazon.com.
2. Win32 Locale, Unicode & Wide Characters (Story) and Windows Win32 Users & Groups (Microsoft implementation) for Multibytes, Unicode characters and Localization.
3. For C++ using template based characters and string manipulations story and examples can be found C++ Template Based Strings & Characters 1 and C++ Template Based Strings & Characters 2.
