To:
Tenouk

MODULE Xb
USING C LIBRARY - THE C CHARACTER AND STRING 3

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

My Training Period: hours

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.

X.8 Memory Functions

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

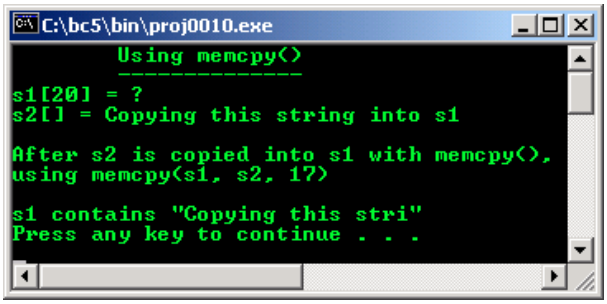
Table X.7: The memory functions of the string handling library

- 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";

    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");
```

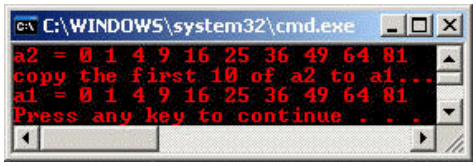
```
    printf("using memcpy(s1, s2, 17)\n");
    printf("\ns1 contains \"%s\"\n", s1);
    return 0;
}
```

Output:




```
// memcpy_s(), copy memory in a more secure way.
#include <memory.h>
#include <stdio.h>

int main()
{
   int a1[10], a2[100], i;
   errno_t err;

   // populate a2 with squares of integers
   for (i = 0; i < 100; i++)
      { a2[i] = i*i; }

   printf("a2 = ");
   for (i = 0; i < 10; i++)
      printf("%d ", a2[i]);
   printf("\n");

   // tell memcpy_s to copy 10 ints (40 bytes), giving
   // the size of the a1 array (also 40 bytes).
   printf("copy the first 10 of a2 to a1...\n");
   err = memcpy_s(a1, sizeof(a1), a2, 10 * sizeof (int));

   if (err)
   { printf("Error executing memcpy_s.\n"); }
   else
   {
      printf("a1 = ");
      for (i = 0; i < 10; i++)
         printf("%d ", a1[i]);
   }
   printf("\n");
   return 0;
}
```

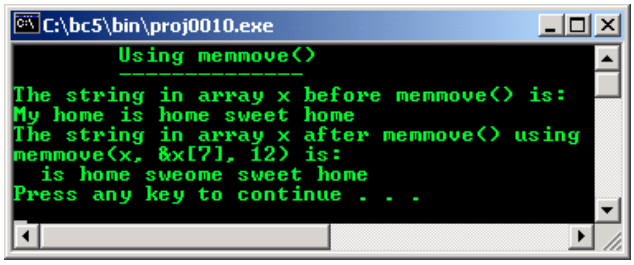
Output:




```
   // using memmove(). The secure version is in the comments.
   #include <stdio.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");
```

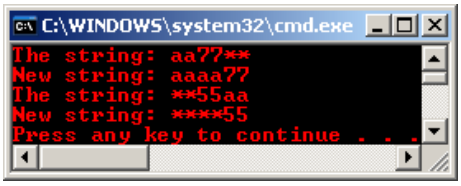
```
    // memmove_s(void *dest, size_t sizeInBytes, const void *src, size_t count); - a secure version
    printf("memmove(x, &x[7], 12) is:\n %s\n", memmove(x, &x[7], 12));
    return  0;
}
```

Output:



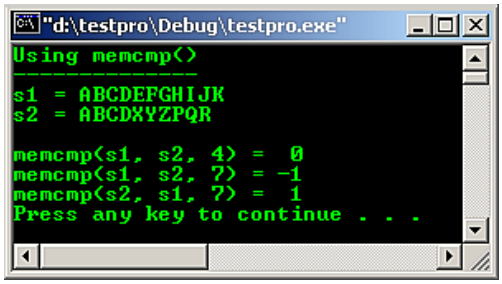

```
// demonstrates the overlapping copy using memmove()
// always handles it correctly; memcpy may handle
// it correctly.
#include <memory.h>
#include <string.h>
#include <stdio.h>

char str1[7] = "aa77**";

int main( void )
{
    printf("The string: %s\n", str1);
    memcpy(str1 + 2, str1, 4);
    printf("New string: %s\n", str1);
    // reset string
    strcpy_s(str1, sizeof(str1), "**55aa");

    printf("The string: %s\n", str1);
    memmove(str1 + 2, str1, 4);
    printf("New string: %s\n", str1);
    return 0;
}
```

Output:



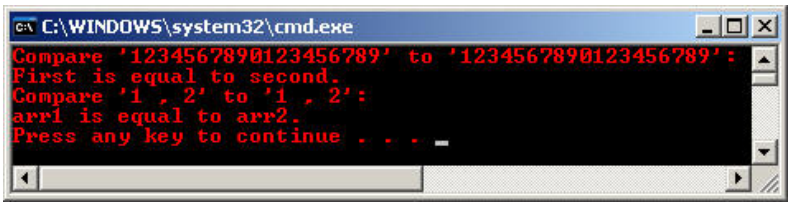

```
    // using memcmp()
    #include <stdio.h>
    #include <stdlib.h>
    #include <string.h>

    int main()
    {
      char s1[ ] = "ABCDEFGHIJK", 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", memcmp(s1, s2, 4));
      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:




```
// This program uses memcmp to compare
// the strings named first and second. If the first
// 19 bytes of the strings are equal, the program
// considers the strings to be equal.
#include <string.h>
#include <stdio.h>

int main(void)
{
   char first[ ] = "12345678901234567890";
   char second[ ] = "12345678901234567891";
   int arr1[ ] = {1,2,3,4};
   int arr2[ ] = {1,2,3,4};
   int result;

   printf("Compare '%.19s' to '%.19s':\n", first, second);
   result = memcmp(first, second, 19);
   if(result < 0)
      printf("First is less than second.\n");
   else if(result == 0)
      printf("First is equal to second.\n");
   else
      printf("First is greater than second.\n");

   printf("Compare '%d , %d' to '%d , %d':\n", arr1[0], arr1[1], arr2[0], arr2[1]);
   result = memcmp(arr1, arr2, sizeof(int) * 2);
   if(result < 0)
      printf("arr1 is less than arr2.\n");
   else if(result == 0 )
      printf("arr1 is equal to arr2.\n");
   else
      printf("arr1 is greater than arr2.\n");
   return 0;
}
```

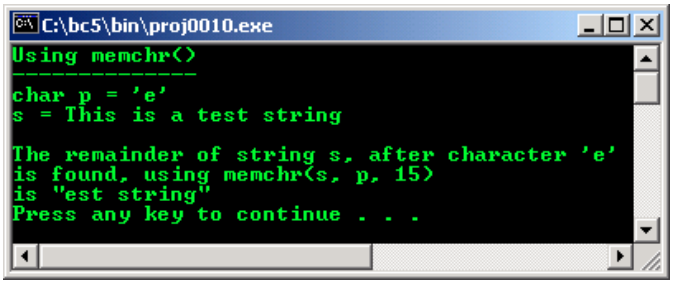
Output:




```
   // 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 memchr(s, p, 15)");
      printf("\nis \"%s\"\n", memchr(s, p, 15));
      return 0;
```

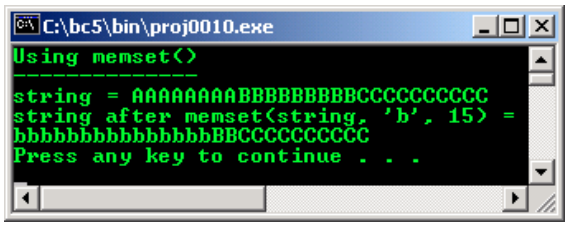
```
}
```

Output:



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

int main()
{
   char string[40] = "AAAAAAAABBBBBBBBBCCCCCCCCCC";
   printf("Using memset()\n");
   printf("--------------\n");
   printf("string = %s\n", string);
   printf("string after memset(string, 'b', 15) =\n%s\n", memset(string, 'b', 15));
   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 summarizes in table X.8.

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

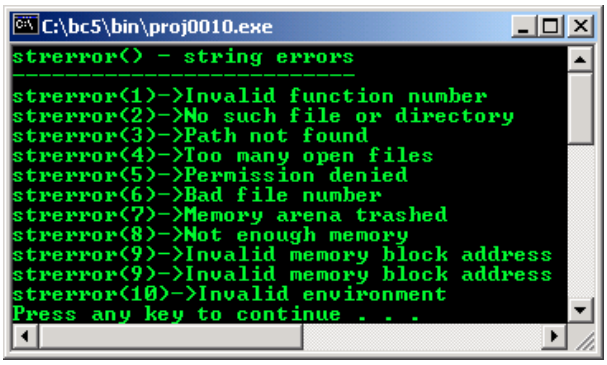
Table X.8: The string manipulation functions of the string handling library

- 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\n", strerror(1));
    printf("strerror(2)->%s\n", strerror(2));
    printf("strerror(3)->%s\n", strerror(3));
    printf("strerror(4)->%s\n", strerror(4));
    printf("strerror(5)->%s\n", strerror(5));
```

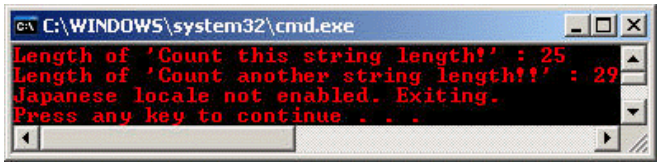
```
    printf("strerror(6)->%s\n", strerror(6));
    printf("strerror(7)->%s\n", strerror(7));
    printf("strerror(8)->%s\n", strerror(8));
    printf("strerror(9)->%s\n", strerror(9));
    printf("strerror(9)->%s\n", strerror(9));
    printf("strerror(10)->%s\n", strerror(10));
    return 0;
}
```

Output:



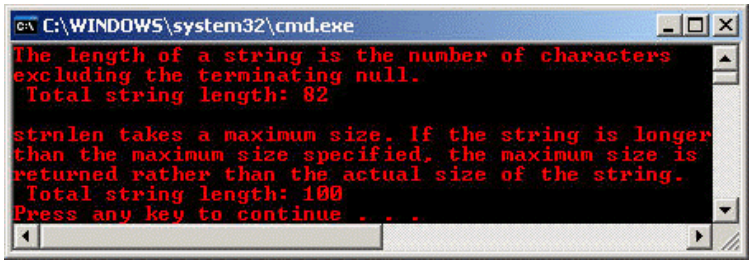

```
// strlen() - determine the length of a string.
// for the multi-byte character example to work correctly,
// the Japanese language support for non-Unicode programs
// must be enabled by the operating system.
#include <string.h>
#include <locale.h>
#include <stdio.h>
// for exit()
#include <stdlib.h>
// Microsoft specific
#include <mbstring.h>

int main()
{
    char* str1 = "Count this string length!";
    wchar_t* wstr1 = L"Count another string length!!";
    char * mbstr1;
    char * locale_string;

    // strlen gives the length of single-byte character string
    printf("Length of '%s' : %d\n", str1, strlen(str1));
    // wstrlen() gives the length of a wide character string
    wprintf(L"Length of '%s' : %d\n", wstr1, wcslen(wstr1));
    // A multibyte string: [A] [B] [C] [katakana A] [D] [\0]
    // in Code Page 932. For this example to work correctly,
    // the Japanese language support must be enabled by the operating system.
    mbstr1 = "ABC" "\x83\x40" "D";
    locale_string = setlocale(LC_CTYPE, "Japanese_Japan");
    if (locale_string == NULL)
    {
        printf("Japanese locale not enabled. Exiting.\n");
        exit(1);
    }
    else
    { printf("Locale set to %s\n", locale_string); }

    // _mbslen() will recognize the Japanese multibyte character if the
    // current locale used by the operating system is Japanese
    printf("Length of '%s' : %d\n", mbstr1, _mbslen(mbstr1));

    // _mbstrlen() will recognize the Japanese multibyte character
    // since the CRT locale is set to Japanese even if the OS locale is not.
    printf("Length of '%s' : %d\n", mbstr1, _mbstrlen(mbstr1));
    printf("Bytes in '%s' : %d\n", mbstr1, strlen(mbstr1));
    return 0;
}
```

Output:



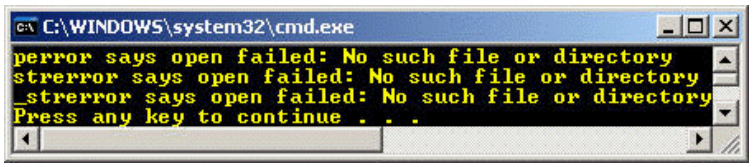

```
// strnlen() - the strlen() secure version
#include <stdio.h>
#include <string.h>

int main()
{
   // str1 is 82 characters long. str2 is 159 characters long
   char* str1 = "The length of a string is the number of characters\n"
                "excluding the terminating null.";
   char* str2 = "strnlen takes a maximum size. If the string is longer\n"
                "than the maximum size specified, the maximum size is\n"
                "returned rather than the actual size of the string.";
   size_t len;
   size_t maxsize = 100;
   len = strnlen(str1, maxsize);
   printf("%s\n Total string length: %d \n\n", str1, len);
   len = strnlen(str2, maxsize);
   printf("%s\n Total string length: %d \n", str2, len);
   return 0;
}
```

Output:




```
// an error message created using perror, strerror, and _strerror.
// program tries to open non exist file...
#include <io.h>
#include <fcntl.h>
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <share.h>

int main(void)
{
   // a file handle
   int fh;

   // the ISO C++ conformant _sopen() and the secure version, _sopen_s()
   if(_sopen_s(&fh, "nonexist.fil", _O_RDONLY, _SH_DENYNO, 0) != 0)
   {
      // three ways to create error message
      perror("perror says open failed");
      // non secure strerror() will generate warning....
      printf("strerror says open failed: %s\n", strerror(errno));
      printf(_strerror("_strerror says open failed"));
      // Note: strerror and _strerror are deprecated; consider
      // using strerror_s and _strerror_s instead.
      // e.g. strerror_s(*buffer, sizeInBytes, *strErrMsg);
   }
   else
   {
      printf("open succeeded on input file\n");
      _close(fh);
   }
   return 0;
}
```

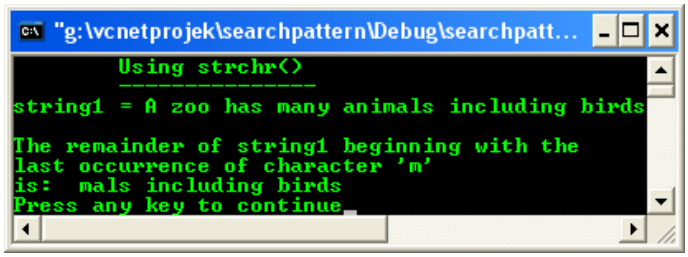
Output:



- Program example compiled using VC++ .Net.

```
// the strchr() - using C++ wrapper
// using C functions in C++ environment
#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", strrchr(string1, c));
    return 0;
}
```

Output:



- For C++ character and string manipulations that use the Standard Template Library (STL), please refer to C++ Template Based Strings & Characters 1 and C++ Template Based Strings & Characters 2.
- 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 sprint()\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 sprint()
--------------
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
```

----------www.tenouk.com----------

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.

|< C Characters & Strings 2 | Main | The main() and command line arguments >| Site Index | Download |

C++ File Input/Output: Part 1 | Part 2 | Part 3