To:
Tenouk

MODULE Xa
USING C LIBRARY - THE C CHARACTER AND STRING 2

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.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
int getchar(void)	Input the next character from the standard input (keyboard) and return it as an integer.
char *gets(char *s)	Input characters from the standard input (keyboard) 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 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 the input is read from the array s instead of reading from the keyboard.

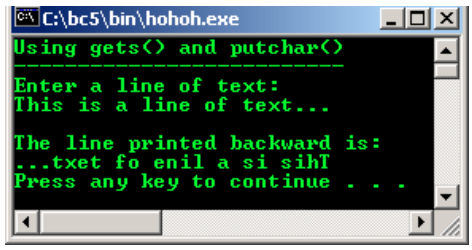
Table X.3 : The standard input/output library character and string functions

- Program examples functions from the stdio.h, beginning with gets() and putchar(). The secure version is put in comments.

```
// using the 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");
    // gets_s(*buffer, size_in_character);
    // gets_s(sentence, 79); // a secure version
    gets(sentence);

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

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

```
            reverse(&s[1]);
            putchar(s[0]);
        }
     return 0;
    }
```

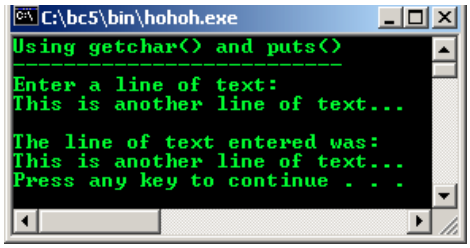
Output:



- The getchar() and puts().

```
// using the 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 = getchar()) != '\n')
          sentence[i++] = c;
    // insert NULL at the end of string
    sentence[i] = '\0';
    puts("\nThe line of text entered was: ");
    puts(sentence);
    return 0;
}
```

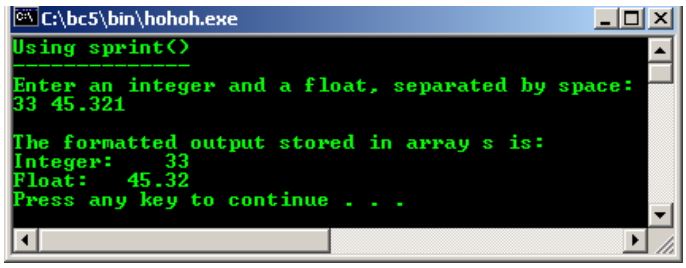
Output:



- The sprintf(). The secure version is in the comments.

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

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_s("%d%f", &x, &y, size_of_buffer, size_of_buffer);
    // scanf_s("%d%f", &x, &y, 4, 8);
    scanf("%d%f", &x, &y);
    // sprintf_s(s, 79, "Integer:%6d\nFloat:%8.2f", x, y);
    // sprintf_s(*buffer, size_of_buffer, "Integer:%6d\nFloat:%8.2f", 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;
}
```

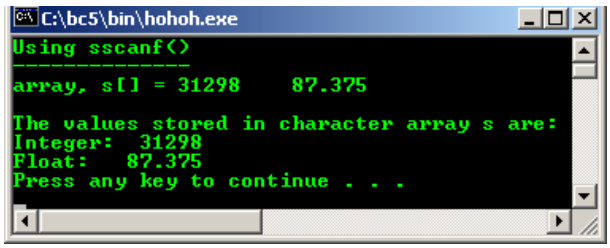
Output:



- The sscanf(). The secure version is in the comments.

```
// 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");
    // sscanf_s(s, "%d%f", &x, &y, x_buffer_size, y_buffer_size, ...);
    // sscanf_s(s, "%d%f", &x, &y, sizeof(int), sizeof(float));
    sscanf(s, "%d%f", &x, &y);
    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);
    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
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.

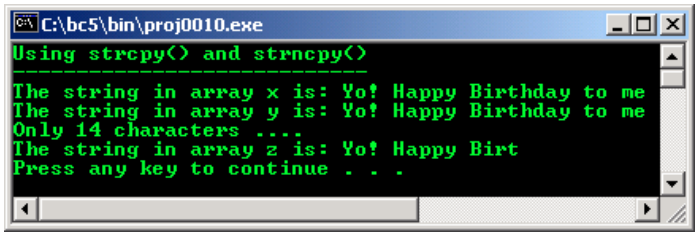
Table X.4: The string manipulation functions of the string handling library

- Let explore the program examples, beginning with strcpy() and strncpy(). The secure version is in the comments.

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

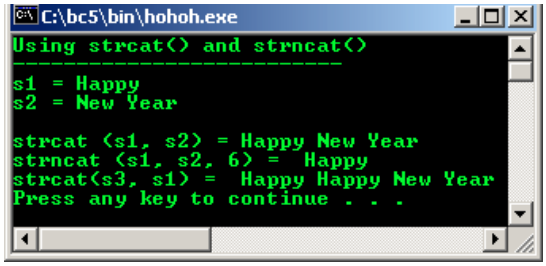
```
int main()
{
   char x[ ] = "Yo! Happy Birthday to me";
   char y[25], z[15];

   printf("Using strcpy() and strncpy()\n");
   printf("---------------------------\n");
   printf("The string in array x is: %s\n", x);
   // strcpy_s(*destination, dest_size_in_byte, *source)
   // printf("The string in array y is: %s\n", strcpy_s(y, 25, x));
   printf("The string in array y is: %s\n", strcpy(y, x));
   printf("The string in array y is: %s\n", y);
   // strncpy_s(*destination, dest_size_in_byte, *source, count);
   // strncpy_s(z, 15, x, 14);
   strncpy(z, x, 14);
   z[14] = '\0';
   printf("Copy only the first 14 characters ....\n", z);
   printf("The string in array z is: %s\n", z);
   return 0;
}
```

Output:




```
// using strcat() and strncat(). The secure version is in the comments.
#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);
      // strcat_s(*destination, dest_size_in_byte, *source)
      // strcat_s(s1, 20, s2);
      printf("\nstrcat (s1, s2) = %s\n", strcat(s1, s2));
      // strncat_s(*destination, dest_buffer_size_in_byte, *destination, count)
      // strncat_s(s3, 40, s1, 6);
      printf("strncat (s1, s2, 6) = %s\n", strncat(s3, s1, 6));
      // strcat_s(s3, 40, s1);
      printf("strcat(s3, s1) = %s\n", strcat(s3, s1));
      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 a summary of the functions and followed by 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 (character set):

1. ASCII – American Standard Code for Information Interchange.
2. EBCDIC – Extended Binary Coded Decimal Interchange Code.
3. Unicode.

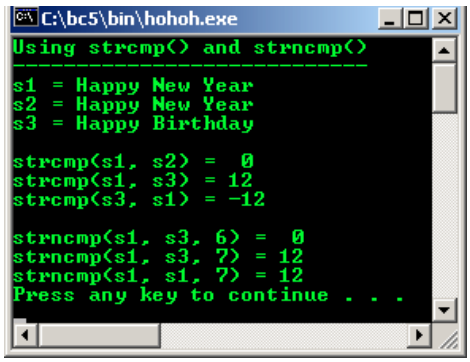
- ASCII, EBCDIC and Unicode are called character codes or character sets.
- String and character manipulations actually involve the manipulation of the appropriate numeric codes and not the characters themselves.
- These explain the interchangeability of characters and small integers in C/C++.

Function prototype	Function description
int strcmp(const char *s1, const char *s2)	Compares 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.
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.

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));
    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.

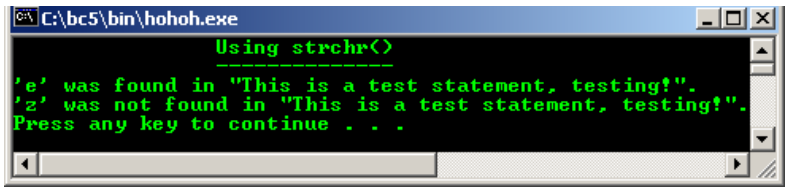
Function prototype	Function description
char *strchr(const char *s, int c)	Locates the first occurrence of character c in string s. If c is found, a pointer to c in s is returned. Otherwise a NULL pointer is returned.
size_t strcspn(const char *s1, const char *s2)	Determines and returns the length of the initial segment of string s1 consisting of characters not contained in string s2.
size_t strspn(const char *s1, const char *s2)	Determines and returns the length of the initial segment of string s1 consisting only of characters contained in string s2.
char *strpbrk(const char *s1, const char *s2)	Locates the first occurrence in string s1 of any character in string s2. If a character from string s2 is found, a pointer to the character in string s1 is returned. Otherwise a NULL pointer is returned.

char *strrchr(const char *s, int c)	Locates the last occurrence of c in string s. If c is found, a pointer to c in string s is returned. Otherwise is a NULL pointer is returned.
char *strstr(const char *s1, const char *s2)	Locates the first occurrence in string s1 of string s2. If the string is found, a pointer to the string in s1 is returned. Otherwise a NULL pointer is returned.
char *strtok(char *s1, const char *s2)	A sequence of calls to strtok() breaks string s1 into “tokens”, logical pieces such as words in a line of text, separated by characters contained in string s2. The first call contains s1 as the first argument, and subsequent calls to continue tokenizing the same string contain NULL as the first argument. A pointer to the current token is returned 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.

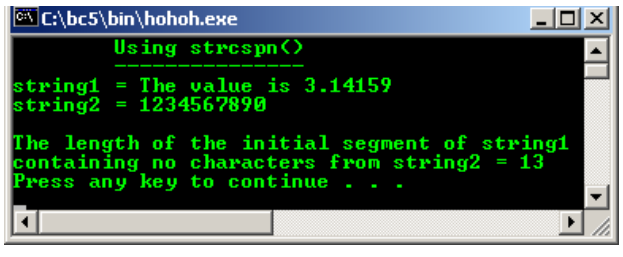
```
// using strchr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

int main()
{
    char *string = "This is a test statement, testing! ";
    char character1 = 'e', character2 = 'z';
    printf("          Using strchr()\n");
    printf("          ---------------\n");
    if (strchr(string, character1) != 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);
    return 0;
}
```

Output:




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

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

    printf("      Using strcspn()\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", strcspn(string1, string2));
    printf("\n");
    return 0;
}
```

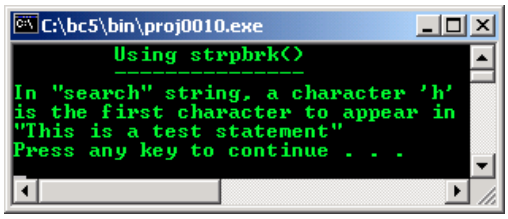
Output:




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

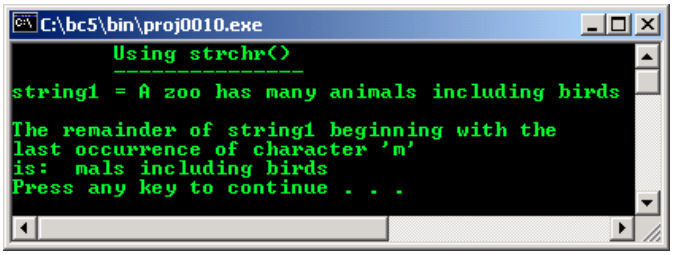
```
#include <string.h>

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\'\n",  string2, *strpbrk(string1, string2));
      printf("is the first character to appear in\n\"%s\"\n", string1);
      return 0;
}
```

Output:



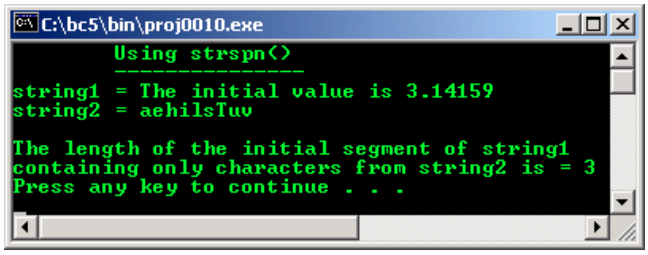

```
// using strchr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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", strrchr(string1, c));
      return 0;
}
```

Output:



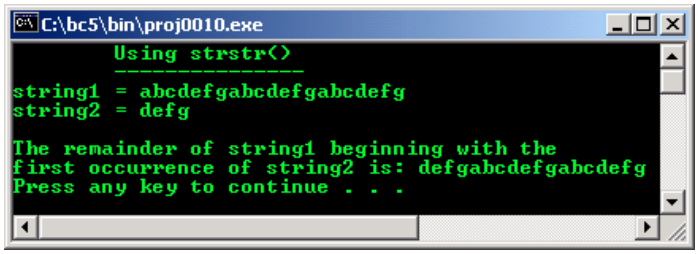

```
// using strspn()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

      printf("      Using strspn()\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", strspn(string1, string2));
      return 0;
}
```

Output:



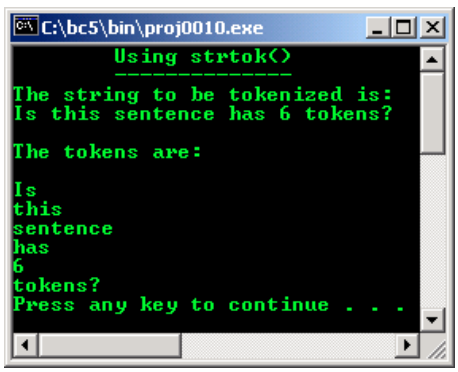

```
// using strstr()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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));
      return 0;
}
```

Output:



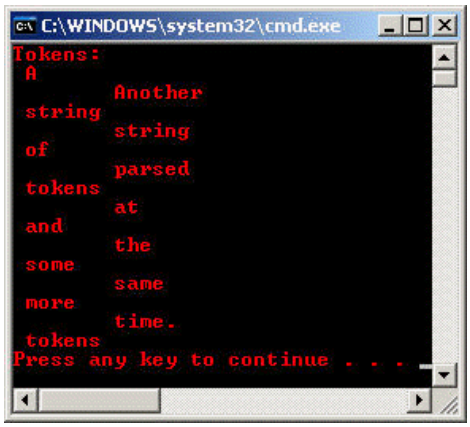

```
// using strtok()
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

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

Output:



```
// a loop uses strtok_s() to print all the tokens
// (separated by commas or blanks) in two
// strings at the same time. The secure version is in the comments.
#include <string.h>
#include <stdio.h>

char string1[ ] = "A string\tof ,,tokens\nand some more tokens";
char string2[ ] = "Another string\n\tparsed at the same time.";
char seps[ ] = " ,\t\n";
char *token1, *token2, *next_token1, *next_token2;

int main( void )
{
   printf( "Tokens:\n" );

   // establish string and get the first token:
   // char *strtok_s(char *strToken, const char *strDelimit, char **context);
   token1 = strtok_s( string1, seps, &next_token1);
   token2 = strtok_s ( string2, seps, &next_token2);
   // while there are tokens in "string1" or "string2"
   while ((token1 != NULL) || (token2 != NULL))
   {
      // Get next token:
      if (token1 != NULL)
      {
         printf( " %s\n", token1 );
         token1 = strtok_s( NULL, seps, &next_token1);
      }
      if (token2 != NULL)
      {
         printf(" %s\n", token2 );
         token2 = strtok_s (NULL, seps, &next_token2);
      }
   }
  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.
