【 기술 노트 23 】

ATmega128 및 WinAVR에서 printf() 함수의 사용 방법

오늘날 대부분의 마이크로콘트롤러용 C컴파일러에서는 printf() 함수를 지원하고 있으며 이것은 내부적으로 1문자를 출력하는 저수준의 스트림 출력(low-level stream output) 함수 PUTCHAR()를 사용하여 특정한 장치로 서식지정 문자열을 출력한다. 따라서, 사용자는 이 PUTCHAR() 함수를 수정함으로써 사용자가 printf() 함수로 출력하는 장치를 변경할 수 있다. 디폴트 출력장치로는 흔히 직렬통신 포트를 사용하고 있다.

8051용의 Keil C컴파일러와 80196용의 IC96 C컴파일러에서 printf() 함수를 사용하는 방법은 [기술 노트 17]에서 이미 설명한 바 있다. 여기서는 WinAVR 20060421 버전 이후부터 사용하는 편리한 printf() 함수 사용 방법을 설명한다.

WinAVR에서도 printf() 함수의 모든 표준적인 기능을 지원하고 있는데, 여기서는 printf() 함수가 내부적으로 저수준 스트림 출력함수의 이름을 PUTCHAR()로 고정하지 않고 사용자가 임의로 1문자 출력 함수를 만든 다음에 이를 표준출력장치 stdout으로 지정하여 사용할 수 있기 때문에 훨씬 더 편리하다. 즉, 각 출력장치에 대한 저수준 스트림 출력함수들을 별도로 만들어 놓고 이것들 중에서 원하는 것을 필요할 때마다 표준출력장치 stdout으로 지정할 수 있으므로 하나의 프로그램에서 printf() 함수에 의한 출력장치를 여러가지로 변경하여 사용할 수 있다는 것이다.

이와 같이 C언어 프로그램에서 printf() 함수를 사용하여 %로 서식 지정하는 데이터를 출력한다면 각종 출력장치의 액세스가 매우 편리해진다. 예를 들어 이제까지 우리가 텍스트형 LCD 모듈의 출력 예제에서 했던 것처럼 수치 데이터를 출력하기 위하여 각종 사용자 정의 함수 LCD_2hex(), LCD_2d(), LCD_2d1() 등을 만들어 사용할 필요가 없기 때문이다. 이러한 편리함은 RS-232C 직렬통신 포트는 물론 텍스트형 LCD 모듈이나 그래픽형 LCD 모듈에도 모두 적용된다. 그러나, 이와 같이 printf() 함수를 사용하여 % 서식 지정으로 출력하는 방법은 사용하기에 편리한 장점이 있는 반면에 오브젝트 파일이 상당히 길어지고 그에 따라 이 루틴의 실행속도도 낮아진다는 단점을 갖는다. 따라서, 플래시 메모리의 용량이 작은 일부 ATtiny 모델에서는 이를 사용하기 어려울 경우도 있다.

WinAVR 20060421 버전 이후부터의 C컴파일러에서 printf() 함수를 사용하는 절차와 방법은 다음과 같다.

① 사용자 프로그램의 서두에서 헤더파일 stdio.h를 인클루드한다. printf() 함수에 관련된 모든 사항은 이 헤더파일에 정의되어 있다.

② 1문자를 출력하는 저수준 스트림 함수에서 사용할 주변장치를 초기화한다. 일반적으로 1문자를 출력하는 저수준 스트림 함수에는 해당 출력장치를 초기화하는 기능을 포함하지 않으므로 이를 별도로 미리 초기화해 놓아야 한다.

③ 1문자를 출력하는 저수준 스트림 함수를 만든다. 이 함수의 이름은 사용자가 임의로 정할 수 있다. 이 함수에서는 모든 ASCII 문자를 출력할 수 있어야 한다.

④ 1문자를 출력하는 저수준 스트림 함수를 표준출력장치 stdout으로 지정한다.

⑤ 필요할 때 printf() 함수를 사용한다. 여기서는 일반적인 ANSI C 표준의 모든 % 서식 지정이 사용될 수 있다.

⑥ 위에서 ④~⑤의 과정은 여러 가지의 주변장치에 대하여 반복적으로 사용할 수 있다.

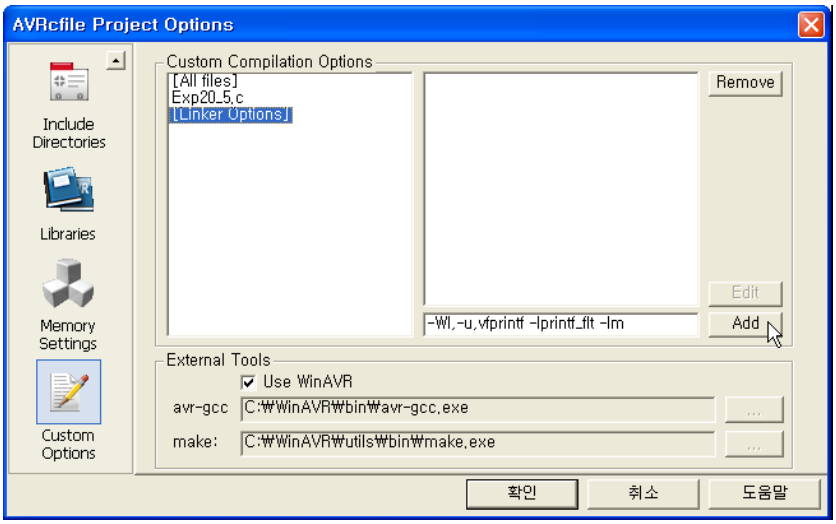
이렇게 printf() 함수를 사용하면 단순한 문자열이나 정수형 데이터를 % 서식으로 지정하여 출력하는 기능은 올바르게 수행된다. 그러나, 부동소수점 데이터를 % 서식으로 지정하여 출력하려면 이 사용자 프로그램을 컴파일할 때 반드시 별도의 라이브러리 파일 libprintf_flt.a를 링크시켜 주어야 한다. 부동소수점 수를 처리하는 루틴은 상당한 용량을 차지하므로 WinAVR에서는 이를 필요할 때만 사용자 프로그램에 링크할 수 있도록 별도의 라이브러리 파일로 만들어 놓았기 때문이다.(기본적으로 정수형 데이터는 라이브러리 파일 libprintf_min.a에서 처리하며 이는 따로 링크시키지 않아도 자동으로 처리된다.)

따라서, 만약 printf() 함수에서 부동소수점 수치 데이터를 사용한다면 아래와 같이 링커 옵션을 주어야 한다.

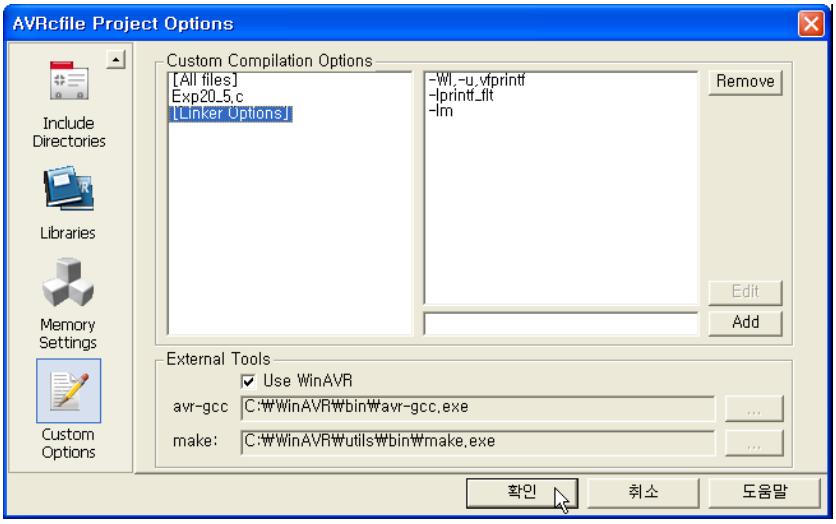
```
-Wl,-u,vfprintf -lprintf_flt -lm
```

여기서 l은 모두 아라비아 숫자 1이 아니라 영문 알파벳 L의 소문자라는네 유의해야 한다.

이와 같이 printf() 함수에서 실수를 사용하기 위하여 링커 옵션을 설정하려면 AVR Studio의 Project → Configuration Options → Custom Options 메뉴에서 <그림 22.1>과 같이 입력하고 Add 버튼을 누른다. 그리고 나서 <그림 22.2>에 보인 것처럼 [Linker Options]를 다시 누르면 방금 입력한 옵션이 나타나는 것을 볼 수 있으며, 여기서 확인 버튼을 눌러서 옵션 설정을 종료한다.



<그림 22.1> printf() 함수에서 실수를 사용하기 위한 링커 옵션의 설정



<그림 22.2> printf() 함수를 실수를 사용하기 위한 링커 옵션의 설정 결과

WinAVR의 printf() 함수에서 사용할 수 있는 %서식 지정 형식을 요약하면 다음과 같다. 이는 ANSI C에서 표준으로 사용하는 %서식 지정 형식을 모두 포함하고 있다.

%[-][+][[0]w][.n][l]변환문자

% : 서식 제어 문자열의 시작을 의미한다.
- : 서식은 항상 필드의 오른쪽으로 정렬이 되지만 -를 붙이면 왼쪽으로 정렬된다.
+ : 필드의 서두에 항상 부호를 붙인다.

0 : 필드의 왼쪽 남는 자리를 공백으로 두지 않고 0을 채워 표시한다.
w : 전체 필드의 폭(field width)을 자리수로 나타낸다.
.n : 실수의 경우 소수점 이하의 자리수(precision)를 나타낸다.
l : long형 정수를 나타내며 변환문자 d, u, o, x의 앞에 붙을 수 있다.
변환문자 d : 부호 있는 10진수(signed int)로 표시한다.
변환문자 u : 부호 없는 10진수(unsigned int)로 표시한다.
변환문자 o : 부호 없는 8진수(unsigned int)로 표시한다.
변환문자 x : 부호 없는 16진수(unsigned int)로 표시한다.(a~f를 소문자로 출력)
변환문자 X : 부호 없는 16진수(unsigned int)로 표시한다.(A~F를 대문자로 출력)
변환문자 f : 부동소수점 수(double)를 소수점을 포함하는 실수 형식으로 표시한다.
변환문자 e : 부동소수점 수(double)를 지수 형식으로 표시한다.(e를 소문자로 출력)
변환문자 E : 부동소수점 수(double)를 지수 형식으로 표시한다.(E를 대문자로 출력)
변환문자 g : %f나 %e 중에서 적당한 방식으로 표시한다.
변환문자 G : %f나 %E 중에서 적당한 방식으로 표시한다.
변환문자 c : 1문자(unsigned char)를 표시한다.
변환문자 s : 문자열(char *)을 표시한다.

예
```
printf("%5d\n",123);          ---> ␣␣123
printf("%05d\n",123);         ---> 00123
printf("%-5d\n",123);         ---> 123␣␣
printf("%7.2f\n",123.456);    ---> ␣123.46
printf("%07.2f\n",123.456);   ---> 0123.46
printf("%+7.2f\n",123.456);   ---> +123.46
printf("%10.2e\n",123.456);   ---> ␣1.23e+002
printf("%10.2E\n",123.456);   ---> ␣1.23E+002
```

☞ 필드를 지정한 값이 실제 숫자의 폭보다 작으면 서식지정은 무시되고 숫자 전체가 표시된다. 부동소수점수에서 소수점 이하의 맨 아래 자리는 그 아래에서 반올림된다.

(1) USART 직렬통신 포트로 출력하는 경우

ATmega128에는 USART0와 USART1 등 2개의 직렬포트를 가지고 있다. 여기서는 printf() 함수를 사용하여 USART0으로 % 서식지정 출력을 내보내는 프로그램을 작성하여 보면 다음과 같다. 물론 USART1에 대하여도 동일한 방법으로 프로그램을 작성할 수 있다.

<프로그램 Exp23_1.c> printf() 함수를 사용한 RS-232C 출력 프로그램

```
/* ===================================================================== */
/*                  Exp23_1.c : printf() Function to USART0              */
/* ===================================================================== */
/*                        Designed and programmed by Duck-Yong Yoon in 2006.  */

#include <avr/io.h>
#include <stdio.h>
#include "OK128.h"
```

```
static int USART0_putchar(char c, FILE *stream);
static FILE device = FDEV_SETUP_STREAM(USART0_putchar, NULL, _FDEV_SETUP_WRITE);

void USART0_initialize(void)                    /* initialize USART0 */
{
  UBRR0H = 0;                                   // 19200 bps
  UBRR0L = 51;
  UCSR0A = 0x00;                                // asynchronous normal mode
  UCSR0B = 0x18;                                // Rx/Tx enable, 8 data
  UCSR0C = 0x06;                                // no parity, 1 stop, 8 data
}

static int USART0_putchar(char c, FILE *stream) /* print a character to USART0 */
{
  if(c == '\n')                                 // process CR(carriage return)
    { loop_until_bit_is_set(UCSR0A,UDRE0);
      UDR0 = 0x0D;
      loop until bit_is_set(UCSR0A,UDRE0);
      UDR0 = 0x0A;
    }
  else
    { loop until bit is set(UCSR0A,UDRE0);      // Tx ready ?
      UDR0 = c;                                 // if yes, print
    }
  return 0;
}

int main(void)
{ unsigned char i = 1;
  double x = 0.001;

  MCU_initialize();                             // initialize MCU and kit
  Delay_ms(50);                                 // wait for system stabilization
  LCD_initialize();                             // initialize text LCD module
  Beep();

  USART0_initialize();                          // initialize USART0

  LCD_string(0x80,"printf to USART0");          // display title
  LCD_string(0xC0,"Press KEY4-KEY1!");

  stdout = &device;                             // set USART0 to stdout

  while(1)
    { switch(Key_input())                       // key input
        { case 0xE0 : PORTB = 0x10;             // KEY1 ?
                      printf("KEY1 is OK !  i=%3d  x=%5.3f\n",i++,x);
                      x += 0.001;
                      break;
          case 0xD0 : PORTB = 0x20;             // KEY2 ?
                      printf("KEY2 is OK !  i=%3d  x=%5.3f\n",i++,x);
                      x += 0.001;
                      break;
          case 0xB0 : PORTB = 0x40;             // KEY3 ?
                      printf("KEY3 is OK !  i=%3d  x=%5.3f\n",i++,x);
                      x += 0.001;
```

```
                    break;
          case 0x70 : PORTB = 0x80;              // KEY4 ?
                    printf("KEY4 is OK !  i=%3d  x=%5.3f\n",i++,x);
                    x += 0.001;
                    break;
          default:  break;
        }
    }
}
```

☞ 먼저 FDEV_SETUP_STREAM() 매크로를 사용하여 원하는 출력장치로 1문자를 출력하는 함수를 임의의 이름 device을 갖는 장치로 지정하고, 나중에 이를 표준출력장치인 stdout으로 지정하는 문법 형식을 주의하여 익혀두기 바란다.

이 프로그램을 컴파일하여 *OK-128* 키트에 다운로드하고 실행한 후에 KEY1~KEY4를 차례로 누르면서 퍼스널컴퓨터로 진송되는 출력을 확인하면 printf() 함수의 동작을 눈으로 볼 수 있다.

이 프로그램의 출력 결과를 확인하려면 *OK-128* 키트의 콘넥터 CN4와 퍼스널컴퓨터의 직렬포트를 RS-232C 통신 케이블로 연결해야 한다. 그리고, 퍼스널컴퓨터에서 하이퍼터미널을 사용할 때 19200 bps, 8 데이터, 패리티 없음, 1 스톱 비트로 설정해야 한다.

(2) 텍스트형 LCD 모듈에 출력하는 경우

이번에는 printf() 함수를 사용하여 텍스트형 LCD 모듈에 % 서식지정 출력을 내보내는 프로그램을 작성하여 보면 다음과 같다.

<프로그램 Exp23_2.c> printf() 함수를 사용한 텍스트형 LCD 모듈 출력 프로그램

```
/* ===================================================================== */
/*              Exp23_2.c : printf() Function to Text LCD Module         */
/* ===================================================================== */
/*                        Designed and programmed by Duck-Yong Yoon in 2006.  */

#include <avr/io.h>
#include <stdio.h>
#include "OK128.h"

static int LCD_putchar(char c, FILE *stream);
static FILE device = FDEV_SETUP_STREAM(LCD_putchar, NULL, _FDEV_SETUP_WRITE);

static int LCD_putchar(char c, FILE *stream)    /* print a character to LCD */
{
  if((c >= 0x20) && (c <= 0x7E))                // check from 0x20 to 0x7E
    LCD_data(c);
  return 0;
}
```

```
int main(void)
{ unsigned char i = 1;
  double x = 0.001;

  MCU_initialize();                          // initialize MCU and kit
  Delay_ms(50);                              // wait for system stabilization
  LCD_initialize();                          // initialize text LCD module
  Beep();

  LCD_string(0x80," printf  to LCD ");       // display title
  LCD_string(0xC0,"Press KEY4-KEY1!");

  stdout = &device;                          // set text LCD to stdout

  while(1)
    { switch(Key_input())                    // key input
        { case 0xE0 : PORTB = 0x10;          // KEY1 ?
                      LCD_command(0x80);
                      printf("  KEY1 is OK !  ");
                      LCD_command(0xC0);
                      printf(" i=%03d  x=%5.3f ",i,x);
                      i++;
                      x += 0.001;
                      break;
          case 0xD0 : PORTB = 0x20;          // KEY2 ?
                      LCD_command(0x80);
                      printf("  KEY2 is OK !  ");
                      LCD_command(0xC0);
                      printf(" i=%03d  x=%5.3f ",i,x);
                      i++;
                      x += 0.001;
                      break;
          case 0xB0 : PORTB = 0x40;          // KEY3 ?
                      LCD_command(0x80);
                      printf("  KEY3 is OK !  ");
                      LCD_command(0xC0);
                      printf(" i=%03d  x=%5.3f ",i,x);
                      i++;
                      x += 0.001;
                      break;
          case 0x70 : PORTB = 0x80;          // KEY4 ?
                      LCD_command(0x80);
                      printf("  KEY4 is OK !  ");
                      LCD_command(0xC0);
                      printf(" i=%03d  x=%5.3f ",i,x);
                      i++;
                      x += 0.001;
                      break;
          default:    break;
        }
    }
}
```

☞ 텍스트형 LCD로 1문자를 출력하는 함수 LCD_putchar()에서는 ASCII 코드값이 0x20~0x7E의 범위일 때만

정상적으로 출력하도록 하였다. 이렇게 하면 텍스트형 LCD 모듈에서 아무 의미가 없는 여러가지 제어문자를 무시할 수 있다.

이 프로그램을 컴파일하여 *OK-128* 키트에 다운로드하고 실행한 후에 KEY1~KEY4를 차례로 누르면서 텍스트형 LCD 모듈의 표시 내용을 확인하면 printf() 함수의 동작을 눈으로 볼 수 있다.

(3) 그래픽형 LCD 모듈에 출력하는 경우

이번에는 printf() 함수를 사용하여 그래픽형 LCD 모듈에 % 서식지정 출력을 내보내는 프로그램을 작성하여 보면 다음과 같다.

<프로그램 Exp23_3.c> printf() 함수를 사용한 그래픽형 LCD 모듈 출력 프로그램

```
/* ================================================================================ */
/*             Exp23_3.c : printf() Function to Graphic LCD Module                  */
/* ================================================================================ */
/*                          Designed and programmed by Duck-Yong Yoon in 2006.  */

#include <avr/io.h>
#include <stdio.h>
#include "OK128.h"
#include "GLCD128.h"
static int GLCD_putchar(char c, FILE *stream);
static FILE device = FDEV_SETUP_STREAM(GLCD_putchar, NULL, _FDEV_SETUP_WRITE);

static int GLCD_putchar(char c, FILE *stream)   /* print a character to GLCD */
{
  if((c >= 0x20) && (c <= 0x7E))                // check from 0x20 to 0x7E
    GLCD_English(c);
  return 0;
}

int main(void)
{ unsigned char i = 1;
  double x = 0.001;

  MCU_initialize();                             // initialize MCU and kit
  Delay_ms(50);                                 // wait for system stabilization
  LCD_initialize();                             // initialize text LCD module
  Beep();

  GLCD_clear();                                 // initialize GLCD screen
  cursor_flag = 0;                              // cursor off

  LCD_string(0x80," printf to GLCD ");          // display title
  LCD_string(0xC0,"Press KEY4-KEY1!");

  stdout = &device;                             // set graphic LCD to stdout
```

```
    while(1)
      { switch(Key_input())                        // key input
          { case 0xE0 : PORTB = 0x10;              // KEY1 ?
                        GLCD_string(0,0,"  KEY1 is OK !  ");
                        GLCD_xy(2,3);
                        printf("i = %03d",i);
                        GLCD_xy(3,3);
                        printf("x = %5.3f",x);
                        i++;
                        x += 0.001;
                        break;
            case 0xD0 : PORTB = 0x20;              // KEY2 ?
                        GLCD_string(0,0,"  KEY2 is OK !  ");
                        GLCD_xy(2,3);
                        printf("i = %03d",i);
                        GLCD_xy(3,3);
                        printf("x = %5.3f",x);
                        i++;
                        x += 0.001;
                        break;
            case 0xB0 : PORTB = 0x40;              // KEY3 ?
                        GLCD_string(0,0,"  KEY3 is OK !  ");
                        GLCD_xy(2,3);
                        printf("i = %03d",i);
                        GLCD_xy(3,3);
                        printf("x = %5.3f",x);
                        i++;
                        x += 0.001;
                        break;
            case 0x70 : PORTB = 0x80;              // KEY4 ?
                        GLCD_string(0,0,"  KEY4 is OK !  ");
                        GLCD_xy(2,3);
                        printf("i = %03d",i);
                        GLCD_xy(3,3);
                        printf("x = %5.3f",x);
                        i++;
                        x += 0.001;
                        break;
            default:    break;
          }
      }
    }
```

☞ 그래픽형 LCD로 1문자를 출력하는 함수 GLCD_putchar()에서는 ASCII 코드값이 0x20~0x7E의 범위일 때만 정상적으로 출력하도록 하였다. 이렇게 하면 그래픽형 LCD 모듈에서 아무 의미가 없는 여러가지 제어문자를 무시할 수 있다. printf()에는 한글 메시지를 출력하는 기능이 없으며, 이런 기능이 필요한 경우에는 GLCD_string() 함수로 별도 처리하면 된다.

이 프로그램을 컴파일하여 *OK-128* 키트에 다운로드하고 실행한 후에 KEY1~KEY4를 차례로 누르면서 그래픽형 LCD 모듈의 표시 내용을 확인하면 printf() 함수의 동작을 눈으로 볼 수 있다.

(4) 여러개의 출력장치를 수시로 변경하면서 사용하는 경우

이제 printf() 함수를 사용하여 표준출력장치 stdout을 USART0과 텍스트형 LCD 모듈로 수시로 변경하면서 이들 각 장치에 % 서식지정 출력을 내보내는 프로그램을 작성하여 보면 다음과 같다.

<프로그램 Exp23_4.c> printf() 함수의 출력장치를 수시로 변경하는 프로그램

```
/* ===================================================================== */
/*                  Exp23_4.c : Change printf() Function                 */
/* ===================================================================== */
/*                        Designed and programmed by Duck-Yong Yoon in 2006.  */

#include <avr/io.h>
#include <stdio.h>
#include "OK128.h"

static int USART0_putchar(char c, FILE *stream);
static FILE device1 = FDEV_SETUP_STREAM(USART0_putchar, NULL, _FDEV_SETUP_WRITE);

static int LCD_putchar(char c, FILE *stream);
static FILE device2 = FDEV_SETUP_STREAM(LCD_putchar, NULL, _FDEV_SETUP_WRITE);

void USART0_initialize(void)                    /* initialize USART0 */
{
  UBRR0H = 0;                                   // 19200 bps
  UBRR0L = 51;
  UCSR0A = 0x00;                                // asynchronous normal mode
  UCSR0B = 0x18;                                // Rx/Tx enable, 8 data
  UCSR0C = 0x06;                                // no parity, 1 stop, 8 data
}

static int USART0_putchar(char c, FILE *stream) /* print a character to USART0 */
{
  if(c == '\n')                                 // process CR(carriage return)
    { loop_until_bit_is_set(UCSR0A,UDRE0);
      UDR0 = 0x0D;
      loop_until_bit_is_set(UCSR0A,UDRE0);
      UDR0 = 0x0A;
    }
  else
    { loop_until_bit_is_set(UCSR0A,UDRE0);      // Tx ready ?
      UDR0 = c;                                 // if yes, print
    }
  return 0;
}

static int LCD_putchar(char c, FILE *stream)    /* print a character to LCD */
{
  if((c < 0x20) | (c > 0x7E))                   // check from 0x20 to 0x7E
    return 0;

  LCD_data(c);
```

```
  return 0;
}

int main(void)
{ unsigned char i = 1;
  double x = 0.001;

  MCU_initialize();                          // initialize MCU and kit
  Delay_ms(50);                              // wait for system stabilization
  LCD_initialize();                          // initialize text LCD module
  Beep();

  USART0_initialize();                       // initialize USART0

  LCD_string(0x80," 2=LCD 1=USART0 ");       // display title
  LCD_string(0xC0,"Press KEY2/KEY1!");

  while(1)
    { switch(Key_input())                    // key input
        { case 0xE0 : PORTB = 0x10;          // KEY1 ?
                      stdout = &device1;     // set USART0 to stdout
                      printf(" i=%03d  x=%5.3f\n",i,x);
                      i++;
                      x += 0.001;
                      break;
          case 0xD0 : PORTB = 0x20;          // KEY2 ?
                      stdout = &device2;     // set text LCD to stdout
                      LCD_command(0xC0);
                      printf(" i=%03d  x=%5.3f ",i,x);
                      i++;
                      x += 0.001;
                      break;
          default:    break;
        }
    }
}
```

☞ USART0은 장치 이름을 device1로 정의하고 텍스트형 LCD 모듈은 장치 이름을 device2로 정의하였다. 그러면 printf() 함수로 문자열을 출력할 때마다 해당 장치 이름을 표준출력장치인 stdout으로 지정해 주면 된다.

이 프로그램을 실행한 후에 *OK-128* 키트에서 KEY1을 누르면 printf() 함수가 출력하는 내용이 USART0으로 출력되고, KEY2를 누르면 printf() 함수가 출력하는 내용이 텍스트형 LCD 모듈로 출력되는 것을 확인할 수 있다.

【 참고문헌 】

1. 윤덕용, 고성능 AVR 정복 시리즈① - AVR ATmega128 정복, 도서출판 Ohm사, 2006