임베디드 C 프로그래밍

실습 자료

ACE Lab.

KNU

목록

	이름	내용
메모리 모델	정수 표현	부호있는 정수와 부호없는 정수의 표현 방법
	Byte Ordering	Little endian & big endian
	구조체	구조체 표현 방법, 구조체의 크기, 패딩, 바이트 정렬, 비트 단위 사용
	공용체	공용체 사용 방법, 구조체와 공용체의 결합
	포인터	포인터의 원리, * 및 & 키워드 활용
	배열	포인터와 배열의 관계, 2차원 및 3차원 배열 활용과 포인터 활용, 주소 계산
	volatile	volatile 키워드 활용
	메모리 접근	주소를 이용해 직접 메모리에 접근하는 방법
	함수 포인터	함수 포인터의 활용
변수 활용	변수 별 사용	전역변수, 지역변수, 정적 (전역 & 지역)변수의 활용
	비트 조작	원하는 비트 설정, 클리어, 토글 방법
	전역변수의 위험성	전역변수를 ISR에서 사용할 경우 위험성
	함수 매개변수 전달	Call-by-value & call-by-reference, const 키워드
	고정 소수점	부동 소수점을 고정 소수점으로 변환하여 사용

목록

	이름	내용
RTOS	태스크 스케줄링	RMS (Rate-monotonic scheduling) 설명
	스케줄 가능 분석	여러 태스크의 실행 시간 기준으로 스케줄 가능 분석
	동기화	여러 태스크 사이의 공유 데이터 동기화

	이름	내용
최적화	변수 사용	각 변수 종류별 실행 시간 측정 및 비교
	배열과 포인터	배열과 포인터를 사용하는 코드 실행 시간 측정 및 비교
	매개변수 갯수	매개변수 갯수에 대한 실행 시간 측정 및 비교

	이름	내용
테스팅		
	MC/DC 분석	

정수 표현

- 예제1 : 부호 있는 정수와 부호 없는 정수

```c
#include<stdio.h>

#define EX1

int main(int argc, char *argv[])
{
#ifdef EX1
    /* Example 1 : signed & unsigned integer */
    int sint;
    unsigned int uint;

    sint = 0x12345678;
    uint = 0x12345678;
    fprintf(stdout, "================== Test 1 ==================\n");
    fprintf(stdout, "Value of the signed integer :  %d\n", sint);
    fprintf(stdout, "Value of the unsigned integer :    %u\n", uint);

    sint = 0x87654321;
    uint = 0x87654321;
    fprintf(stdout, "\n================== Test 2 ==================\n");
    fprintf(stdout, "Value of the signed integer :  %d\n", sint);
    fprintf(stdout, "Value of the unsigned integer :    %u\n", uint);
#endif

    return 0;
}
```

소스 코드

```
================== Test 1 ==================
Value of the signed integer :   305419896
Value of the unsigned integer : 305419896

================== Test 2 ==================
Value of the signed integer :   -2023406815
Value of the unsigned integer : 2271560481
```

실행 결과

Byte Ordering

- 예제1 : Byte ordering (Little endian)

```c
#include<stdio.h>

#define EX1

int main(int argc, char *argv[])
{
#ifdef EX1
    /* Example 1 : byte ordering of data (little endian) */
    int a = 0x12345678;
    char *ch_ptr = (char *)&a;

    fprintf(stdout, "================== Test 1 ==================\n");
    fprintf(stdout, "Address of 'a' :       %p\n", &a);
    fprintf(stdout, "Data of 'a' :          0x%x\n", a);
    fprintf(stdout, "Data in address(%p) :  0x%x\n", ch_ptr, *(ch_ptr));
    fprintf(stdout, "Data in address(%p) :  0x%x\n", ch_ptr+1, *(ch_ptr+1));
    fprintf(stdout, "Data in address(%p) :  0x%x\n", ch_ptr+2, *(ch_ptr+2));
    fprintf(stdout, "Data in address(%p) :  0x%x\n", ch_ptr+3, *(ch_ptr+3));
#endif

    return 0;
}
```

소스 코드

```
================== Test 1 ==================
Address of 'a' :                0x2001ffd4
Data of 'a' :                   0x12345678
Data in address(0x2001ffd4) :   0x78
Data in address(0x2001ffd5) :   0x56
Data in address(0x2001ffd6) :   0x34
Data in address(0x2001ffd7) :   0x12
```

실행 결과

구조체

- 예제1 : 구조체 표현 방법

```c
#include <stdio.h>

#define EX1
//#define EX2
//#define EX3

#ifdef EX1

//declaration
struct mydata{
    int a;
    int b;
};

//declaration & initialization
struct mydata2{
    int a;
    char b;
    short c;
}object2;

//declaration with typedef
typedef struct {
    char c;
    int a;
    short b;
} mydata3;

int main(int argc, char* argv[])
{
    struct mydata object1;
    // object2;
    struct mydata2 object3;
    mydata3 object4;

    fprintf(stdout, "mydata struct size : %d\n", sizeof(object1));
    fprintf(stdout, "mydata2 struct size : %d\n", sizeof(object2));
    fprintf(stdout, "mydata2 struct size : %d\n", sizeof(object3));
    fprintf(stdout, "mydata3 struct size : %d\n", sizeof(object4));

}

#endif
```

구조체

- 예제1 : 구조체 표현 방법

```
PRIGROUP unimplemented
mydata struct size : 8
mydata2 struct size : 8
mydata2 struct size : 8
mydata3 struct size : 12

QEMU semihosting exit(0)
```

```c
//declaration & initialization
struct mydata2{
    int a;
    char b;
    short c;
}object2;

//declaration with typedef
typedef struct {
    char c;
    int a;
    short b;
} mydata3;
```

구조체

- 예제2 : 구조체의 크기, 패딩, 바이트 정렬

```c
#ifdef EX2

//declaration & initialization
struct mydata2{
    int a;
    char b;
    short c;
}object2;

//declaration with typedef
typedef struct {
    char c;
    int a;
    short b;
} mydata3;

typedef struct {
    short b;
    double d;
}mydata4;
```

```c
int main(int argc, char* argv[])
{
    mydata3 object4;
    mydata4 object5;

    fprintf(stdout, "mydata2 size : %d\n", sizeof(object2));
    fprintf(stdout, "mydata2 int size   : %d\n", sizeof(object2.a));
    fprintf(stdout, "mydata2 char size  : %d\n", sizeof(object2.b));
    fprintf(stdout, "mydata2 short size : %d\n", sizeof(object2.c));
    fprintf(stdout, "==========================\n");
    fprintf(stdout, "mydata3 size : %d\n", sizeof(object4));
    fprintf(stdout, "mydata3 char size  : %d\n", sizeof(object4.c));
    fprintf(stdout, "mydata3 int size   : %d\n", sizeof(object4.a));
    fprintf(stdout, "mydata3 short size : %d\n", sizeof(object4.b));
    fprintf(stdout, "==========================\n");
    fprintf(stdout, "mydata4 size : %d\n", sizeof(object5));
    fprintf(stdout, "mydata4 short size  : %d\n", sizeof(object5.b));
    fprintf(stdout, "mydata4 double size : %d\n", sizeof(object5.d));
    fprintf(stdout, "==========================\n");
    fprintf(stdout, "mydata2 address : %p\n", &object2);
    fprintf(stdout, "mydata2 int address   : %p\n", &(object2.a));
    fprintf(stdout, "mydata2 char address  : %p\n", &(object2.b));
    fprintf(stdout, "mydata2 short address : %p\n", &(object2.c));
    fprintf(stdout, "==========================\n");
    fprintf(stdout, "mydata3 address : %p\n", &object4);
    fprintf(stdout, "mydata3 char address  : %p\n", &(object4.c));
    fprintf(stdout, "mydata3 int address   : %p\n", &(object4.a));
    fprintf(stdout, "mydata3 short address : %p\n", &(object4.b));
    fprintf(stdout, "==========================\n");
    fprintf(stdout, "mydata4 address : %p\n", &object5);
    fprintf(stdout, "mydata4 short address  : %p\n", &(object5.b));
    fprintf(stdout, "mydata4 double address : %p\n", &(object5.d));
}
```

구조체

- 예제2 : 구조체의 크기, 패딩, 바이트 정렬

```
PRIGROUP unimplemented
mydata2 size : 8                mydata2 address : 0x2000022c
mydata2 int size   : 4          mydata2 int address   : 0x2000022c
mydata2 char size  : 1          mydata2 char address  : 0x20000230
mydata2 short size : 2          mydata2 short address : 0x20000232
=========================       =========================
mydata3 size : 12               mydata3 address : 0x2001ffcc
mydata3 char size  : 1          mydata3 char address  : 0x2001ffcc
mydata3 int size   : 4          mydata3 int address   : 0x2001ffd0
mydata3 short size : 2          mydata3 short address : 0x2001ffd4
=========================       =========================
mydata4 size : 16               mydata4 address : 0x2001ffb8
mydata4 short size  : 2         mydata4 short address  : 0x2001ffb8
mydata4 double size : 8         mydata4 double address : 0x2001ffc0
```

구조체

- 예제3: 구조체의 비트단위 사용

```c
#ifdef EX3

struct bit_field {
    unsigned int a : 1;
    unsigned int b : 3;
    unsigned int c : 7;
}flag;

int main(int argc, char* argv[])
{
    flag.a = 1;     // 1bit (1)
    flag.b = 15;    // 4bit (1111)
    flag.c = 255;   // 8bit (1111 1111)

    fprintf(stdout, "flag.a : %u\n", flag.a);
    fprintf(stdout, "flag.b : %u\n", flag.b);   // only 3bit
    fprintf(stdout, "flag.c : %u\n", flag.c);   // only 7bit

    // can't make bit-field pointer
//  fprintf(stdout, "flag.a address : %p\n", &(flag.a));
//  fprintf(stdout, "flag.b address : %p\n", &(flag.b));
//  fprintf(stdout, "flag.c address : %p\n", &(flag.c));
}
#endif
```

구조체

- 예제3: 구조체의 비트단위 사용

```
PRIGROUP unimplemented
flag.a : 1
flag.b : 7
flag.c : 127

QEMU semihosting exit(0)
```

공용체

- 예제1 : 공용체 사용 방법

```c
#include <stdio.h>

#define EX1
//#define EX2

#ifdef EX1

union Box{
    int a;
    short b;
    char c;
};

int main(int argc, char* argv[]){
    union Box b1;

    fprintf(stdout, "Box union size : %d\n", sizeof(b1));

    b1.a = 0x12345678;
    fprintf(stdout, "Box.a data: 0x%x\n", b1.a);
    fprintf(stdout, "Box.b data: 0x%x\n", b1.b);
    fprintf(stdout, "Box.c data: 0x%x\n", b1.c);
    fprintf(stdout, "====================\n");
    fprintf(stdout, "Box.a address: %p\n", &(b1.a));
    fprintf(stdout, "Box.b address: %p\n", &(b1.b));
    fprintf(stdout, "Box.c address: %p\n", &(b1.c));

}


#endif
```

공용체

- 예제1 : 공용체 사용 방법

```
PRIGROUP unimplemented
Box union size : 4
Box.a data: 0x12345678
Box.b data: 0x5678
Box.c data: 0x78
====================
Box.a address: 0x2001ffdc
Box.b address: 0x2001ffdc
Box.c address: 0x2001ffdc

QEMU semihosting exit(0)
```

공용체

- 예제2 : 구조체와 공용체의 결합

```c
#ifdef EX2

union IP_addr{
    unsigned int ipv4;
    struct {
        unsigned int addr4 : 8;
        unsigned int addr3 : 8;
        unsigned int addr2 : 8;
        unsigned int addr1 : 8;
    };
};

int main(int argc, char* argv[]){
    union IP_addr host;
    host.addr4 = 127;
    host.addr3 = 0;
    host.addr2 = 0;
    host.addr1 = 1;

    fprintf(stdout, "host ipv4 : 0x%x\n", host.ipv4); // little-endian
    fprintf(stdout, "host ipv4 : %u.%u.%u.%u\n", host.addr4, host.addr3,host.addr2,host.addr1);

}

#endif
```

공용체

- 예제2 : 구조체와 공용체의 결합

```
PRIGROUP unimplemented
host ipv4 : 0x100007f
host ipv4 : 127.0.0.1

QEMU semihosting exit(0)
```

포인터

- 예제1 : 변수 타입과 포인터 변수 타입의 크기 비교

*main.cpp

```c
#include <stdio.h>

#define Ex1
//#define Ex2

int main(int argc, char* argv[])
{
#ifdef Ex1
    /* Example 1 : size of data & ptr data type */
    fprintf(stdout, "===== Size of data type =====\n");
    fprintf(stdout, "Size of char : %d\n", sizeof(char));
    fprintf(stdout, "Size of short : %d\n", sizeof(short));
    fprintf(stdout, "Size of int : %d\n", sizeof(int));
    fprintf(stdout, "Size of double : %d\n", sizeof(double));
    fprintf(stdout, "Size of float : %d\n", sizeof(float));
    fprintf(stdout, "Size of long : %d\n", sizeof(long));
    fprintf(stdout, "Size of long long : %d\n", sizeof(long long));

    fprintf(stdout, "\n===== Size of pointer data type ======\n");
    fprintf(stdout, "Size of char * : %d\n", sizeof(char *));
    fprintf(stdout, "Size of short * : %d\n", sizeof(short *));
    fprintf(stdout, "Size of int * : %d\n", sizeof(int *));
    fprintf(stdout, "Size of double * : %d\n", sizeof(double *));
    fprintf(stdout, "Size of float * : %d\n", sizeof(float *));
    fprintf(stdout, "Size of long * : %d\n", sizeof(long *));
    fprintf(stdout, "Size of long long * : %d\n", sizeof(long long *));
#endif
```

소스 코드

Console | Problems | Executables | Debugger Console

```
<terminated> test [GDB QEMU Debugging] qemu-system-gnuarmeclipse
========== Test 1 ==========
Address of a :                  0x2001ffd4
Address of *a_ptr :             0x2001ffd0
Stored address in *a_ptr :      0x2001ffd4
Data pointed by a_ptr :         0x12345678

========== Test 2 ==========
Data pointed by int ptr :       0x12345678
Data pointed by short ptr :     0x5678
Data pointed by char ptr :      0x78

========== Test 3 ==========
data(0) : 0x78
data(1) : 0x56
data(2) : 0x34
data(3) : 0x12

QEMU semihosting exit(0)
```

실행 결과

포인터

- 예제2 : 포인터 변수와 *, & 키워드 및 포인터와 메모리 주소 관계 학습

```
28
29  #ifdef Ex2
30      /* Example 2 : Calculating address and value with ptr  */
31      int a = 0x12345678;
32      int *a_ptr = &a;
33
34      fprintf(stdout, "========== Test 1 ==========\n");
35      fprintf(stdout, "Address of a :                %p\n", &a);
36      fprintf(stdout, "Address of *a_ptr :           %p\n", &a_ptr);
37      fprintf(stdout, "Stored address in *a_ptr :    %p\n", a_ptr);
38      fprintf(stdout, "Data pointed by a_ptr :       0x%x\n", *a_ptr);
39
40      fprintf(stdout, "\n========== Test 2 ==========\n");
41      fprintf(stdout, "Data pointed by int ptr :      0x%x\n", *a_ptr);
42      fprintf(stdout, "Data pointed by short ptr :    0x%x\n", *(short *)a_ptr);
43      fprintf(stdout, "Data pointed by char ptr :     0x%x\n", *(char *)a_ptr);
44
45      char *ch_ptr = (char *)a_ptr;
46
47      fprintf(stdout, "\n========== Test 3 ==========\n");
48      fprintf(stdout, "data(0) : 0x%x\n", *(ch_ptr++));
49      fprintf(stdout, "data(1) : 0x%x\n", *(ch_ptr++));
50      fprintf(stdout, "data(2) : 0x%x\n", *(ch_ptr++));
51      fprintf(stdout, "data(3) : 0x%x\n", *(ch_ptr));
52  #endif
53      return 0;
54  }
```

소스 코드

```
Console  Problems  Executables  Debugger Console
<terminated> test [GDB QEMU Debugging] qemu-system-gnuarmeclipse
========== Test 1 ==========
Address of a :                0x2001ffd4
Address of *a_ptr :           0x2001ffd0
Stored address in *a_ptr :    0x2001ffd4
Data pointed by a_ptr :       0x12345678

========== Test 2 ==========
Data pointed by int ptr :      0x12345678
Data pointed by short ptr :    0x5678
Data pointed by char ptr :     0x78

========== Test 3 ==========
data(0) : 0x78
data(1) : 0x56
data(2) : 0x34
data(3) : 0x12

QEMU semihosting exit(0)
```

실행 결과

배열

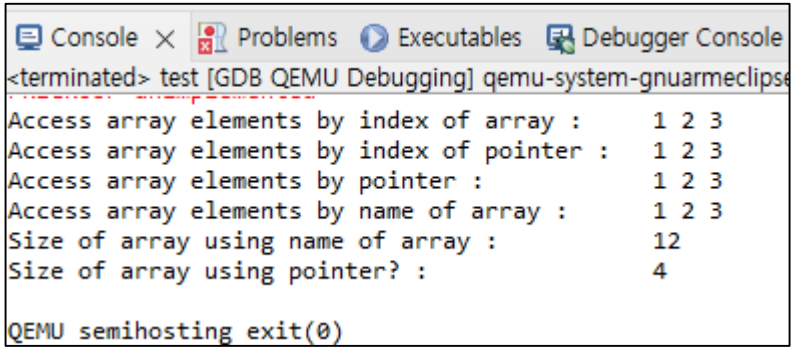
- 예제1 : 포인터와 배열의 관계

```c
#include <stdio.h>

#define Ex1
//#define Ex2
//#define Ex3

int main(int argc, char* argv[])
{
#ifdef Ex1
    /* Example 1 : Relationship between pointer and array  */
    int arr[3] = {1, 2, 3};
    int *arr_ptr = arr;

    fprintf(stdout, "Access array elements by index of array :     %d %d %d\n", arr[0], arr[1], arr[2]);
    fprintf(stdout, "Access array elements by index of pointer :   %d %d %d\n", arr_ptr[0], arr_ptr[1], arr_ptr[2]);
    fprintf(stdout, "Access array elements by pointer :            %d %d %d\n", *(arr_ptr), *(arr_ptr+1), *(arr_ptr+2));
    fprintf(stdout, "Access array elements by name of array :      %d %d %d\n", *(arr), *(arr+1), *(arr+2));
    fprintf(stdout, "Size of array using name of array :           %d\n", sizeof(arr));
    fprintf(stdout, "Size of array using pointer? :                %d\n", sizeof(arr_ptr)); // size of ptr variable

#endif
```

소스 코드

배열

- 예제1 : 포인터와 배열의 관계



실행 결과

배열

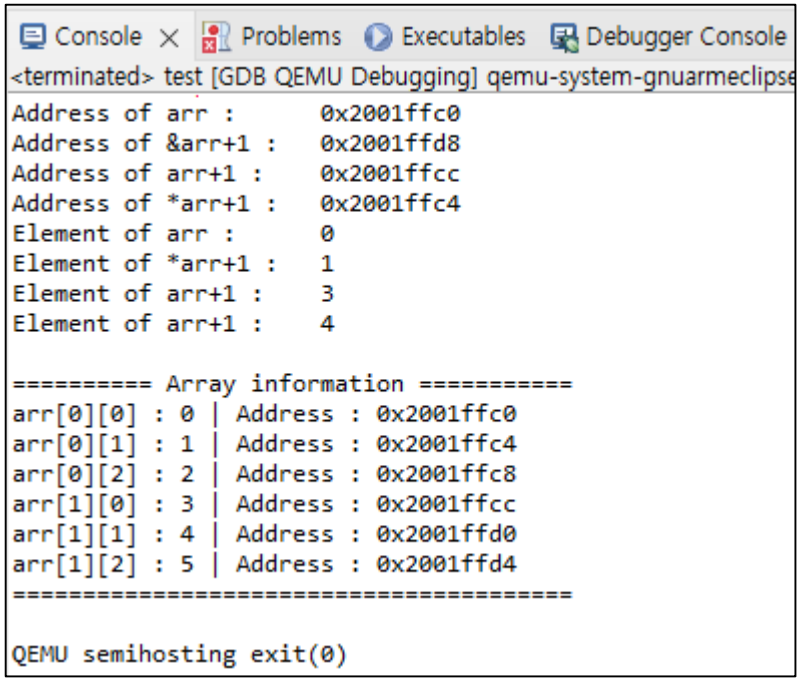
- 예제2 : 2차원 배열에서의 포인터 연산 학습과 포인터 연산과 반복문을 이용한 전체 배열 정보 출력

```
22
23  #ifdef Ex2
24      /* Example 2 : 2-Dimention array */
25      int arr[2][3] = {{0, 1, 2}, {3, 4, 5}};
26
27      fprintf(stdout, "Address of arr :      %p\n", arr);
28      fprintf(stdout, "Address of &arr+1 :   %p\n", &arr+1);
29      fprintf(stdout, "Address of arr+1 :    %p\n", arr+1);
30      fprintf(stdout, "Address of *arr+1 :   %p\n", *arr+1);
31      fprintf(stdout, "Element of arr :      %d\n", *(*arr));
32      fprintf(stdout, "Element of *arr+1 :   %d\n", *(*arr+1));
33      fprintf(stdout, "Element of arr+1 :    %d\n", *(*(arr+1)));
34      fprintf(stdout, "Element of arr+1 :    %d\n", (*(arr+1))[1]);
35
36      fprintf(stdout, "\n========== Array information ==========\n");
37      for(int i=0; i<2; i++)
38      {
39          for(int j=0; j<3; j++)
40          {
41              fprintf(stdout, "arr[%d][%d] : %d | Address : %p\n", i, j, *(*(arr+i)+j), *(arr+i)+j);
42          }
43      }
44      fprintf(stdout, "======================================\n");
45
46  #endif
```

소스 코드

배열

- 예제2 : 2차원 배열에서의 포인터 연산 학습과 포인터 연산과 반복문을 이용한 전체 배열 정보 출력



실행 결과

배열

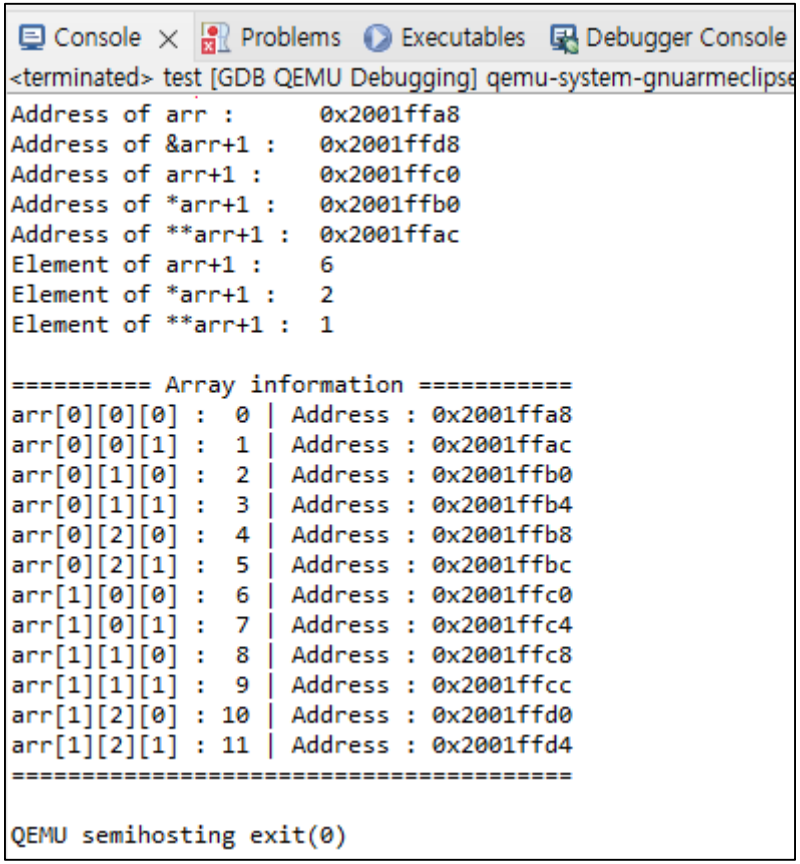
- 예제3 : 3차원 배열에서의 포인터 연산 학습과 포인터 연산과 반복문을 이용한 전체 배열 정보 출력

```c
#ifdef Ex3
    /* Example 3 : 3-Dimention array */
    int arr[2][3][2] = { {{0, 1},{2, 3},{4, 5}}, {{6, 7},{8, 9},{10, 11}} };

    fprintf(stdout, "Address of arr :      %p\n", arr);
    fprintf(stdout, "Address of &arr+1 :   %p\n", &arr+1);
    fprintf(stdout, "Address of arr+1 :    %p\n", arr+1);
    fprintf(stdout, "Address of *arr+1 :   %p\n", *arr+1);
    fprintf(stdout, "Address of **arr+1 :  %p\n", **arr+1);
    fprintf(stdout, "Element of arr+1 :    %d\n", *(*(*(arr+1))));
    fprintf(stdout, "Element of *arr+1 :   %d\n", *(*(*arr+1)));
    fprintf(stdout, "Element of **arr+1 :  %d\n", *(**arr+1));

    fprintf(stdout, "\n========== Array information ==========\n");
    for(int i=0; i<2; i++)
    {
        for(int j=0; j<3; j++)
        {
            for(int k=0; k<2; k++)
            {
                fprintf(stdout, "arr[%d][%d][%d] : %2d | Address : %p\n", i, j, k, *(*(*(arr+i)+j)+k), (*(*(arr+i)+j)+k));
            }
        }
    }
    fprintf(stdout, "=======================================\n");
#endif
    return 0;
}
```

소스 코드

배열

- 예제3 : 3차원 배열에서의 포인터 연산 학습과 포인터 연산과 반복문을 이용한 전체 배열 정보 출력



실행 결과

Volatile

- 예제1 : Volatile 키워드 사용

```c
#include <stdio.h>
#include <stdlib.h>

int
main(int argc, char* argv[])
{
    volatile int a = 0;
    int b = 0;

    b = 1;
    b = 2;
    b = 3;
    b = 4;
    b = 5;

    /* 최적화 옵션에서 중간과정을 생략하고 최종 값만을 b에 할당 */
    printf("Non volatile b: %d\n",b);

    a = 1;
    a = 2;
    a = 3;
    a = 4;
    a = 5;

    printf("volatile a: %d\n",a);

  return 0;
}
```

소스 코드

```
                main(int, char**):
08001664:         push    {r4, lr}
08001666:         sub     sp, #8
36                  volatile int a = 0;
08001668:         movs    r4, #0
0800166a:         str     r4, [sp, #4]
46                  printf("Non volatile b: %d\n",b);
0800166c:         movs    r1, #5
0800166e:         ldr     r0, [pc, #40]   ; (0x8001698 <main(int, char**)+52>)
08001670:         bl      0x8001c68 <printf>
48                  a = 1;
08001674:         movs    r3, #1
08001676:         str     r3, [sp, #4]
49                  a = 2;
08001678:         movs    r3, #2
0800167a:         str     r3, [sp, #4]
50                  a = 3;
0800167c:         movs    r3, #3
0800167e:         str     r3, [sp, #4]
51                  a = 4;
08001680:         movs    r3, #4
08001682:         str     r3, [sp, #4]
52                  a = 5;
08001684:         movs    r3, #5
08001686:         str     r3, [sp, #4]
54                  printf("volatile a: %d\n",a);
08001688:         ldr     r1, [sp, #4]
0800168a:         ldr     r0, [pc, #16]   ; (0x800169c <main(int, char**)+56>)
0800168c:         bl      0x8001c68 <printf>
57              }
08001690:         mov     r0, r4
08001692:         add     sp, #8
08001694:         pop     {r4, pc}
```

assembly

```
Non volatile b: 5
volatile a: 5
```

실행결과

메모리 접근

- 예제1 : 주소를 이용해 직접 메모리에 접근

```c
#include <stdio.h>
#include <stdlib.h>

#define ADDR                0x20001000
#define ACCESS_ADDR         *((unsigned int *)ADDR)
#define DATA                0xFFFFFFFF

int
main(int argc, char* argv[])
{

    printf("DATA_ADDRESS : 0x%X\n", ADDR);
    printf("DATA : 0x%08X\n", ACCESS_ADDR);

    /* Write "DATA" in "ADDR" */
    ACCESS_ADDR = DATA;

    printf("\n===== Write DATA in ADDR =====\n");
    printf("DATA_ADDRESS : 0x%X\n", ADDR);
    printf("DATA : 0x%08X\n", ACCESS_ADDR);

  return 0;
}
```

소스 코드

```
DATA_ADDRESS : 0x20001000
DATA : 0x00000000

===== Write DATA in ADDR =====
DATA_ADDRESS : 0x20001000
DATA : 0xFFFFFFFF
```

실행결과

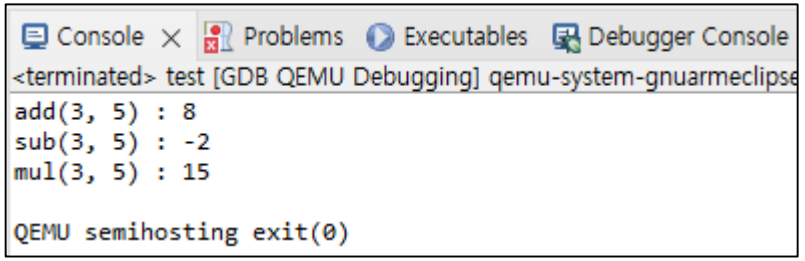
함수 포인터

- 예제1 : 함수 포인터를 활용한 함수 호출

```c
#include <stdio.h>

#define Ex1
//#define Ex2

int add(int num1, int num2);
int sub(int num1, int num2);
int mul(int num1, int num2);

int main(int argc, char* argv[])
{
#ifdef Ex1
    int (*func)(int, int);

    func = add;
    fprintf(stdout, "add(3, 5) : %d\n", func(3, 5));

    func = sub;
    fprintf(stdout, "sub(3, 5) : %d\n", func(3, 5));

    func = mul;
    fprintf(stdout, "mul(3, 5) : %d\n", func(3, 5));
#endif
```

```c
int add(int num1, int num2){
    return num1 + num2;
}

int sub(int num1, int num2){
    return num1 - num2;
}

int mul(int num1, int num2){
    return num1 * num2;
}
```

소스 코드



실행 결과

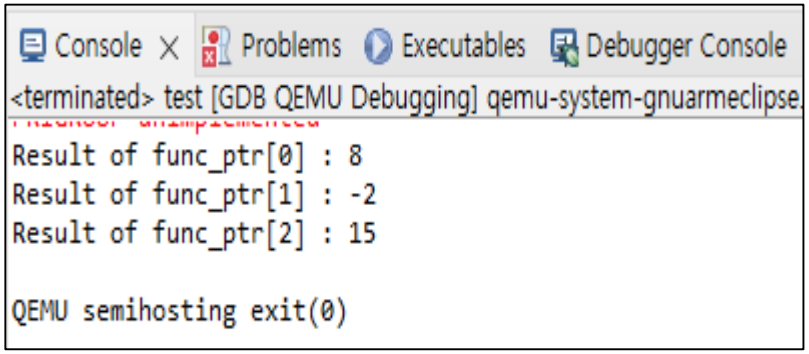
함수 포인터

- 예제2 : 함수 포인터 배열을 활용한 함수 호출

```c
#ifdef Ex2
    int (*func_ptr[3])(int, int) = {add, sub, mul};

    for(int i=0; i<3; i++)
    {
        fprintf(stdout, "Result of func_ptr[%d] : %d\n", i, func_ptr[i](3, 5));
    }

#endif
    return 0;
}
```

```c
int add(int num1, int num2){
    return num1 + num2;
}

int sub(int num1, int num2){
    return num1 - num2;
}

int mul(int num1, int num2){
    return num1 * num2;
}
```

소스 코드



실행 결과

변수 별 사용

- 예제1 : 전역,지역, 정적 변수의 활용

```c
#include <stdio.h>
#include <stdlib.h>

void call();
int global = 100;

int
main(int argc, char* argv[])
{

/* local variable */
    int local = 10;

    printf("local : %d\n",local);
    printf("global : %d\n",global);
    call();
    printf("=========================\n");

    {
        int local = 5;

        printf("local : %d\n",local);
        printf("global : %d\n",global);
        call();
        printf("=========================\n");
    }

    printf("local : %d\n",local);
    printf("global : %d\n",global);
    call();
    printf("=========================\n");

  return 0;
}

void call()
{
    static int st_call = 0;
    int call = 0;
    printf("st_call : %d \n", st_call);
    printf("call : %d \n", call);
    st_call++;
    call++;
}
```

소스 코드

```
local : 10
global : 100
st_call : 0
call : 0
=========================
local : 5
global : 100
st_call : 1
call : 0
=========================
local : 10
global : 100
st_call : 2
call : 0
=========================
```

실행결과

비트 조작

- 예제1 : 원하는 비트 설정, 클리어, 토글 수행

```c
#include <stdio.h>
#include <stdlib.h>

#define N_BIT   3

int
main(int argc, char* argv[])
{
    char num = 0b00000000;
    printf("num : 0x%08x\n",num);
    printf("================\n");

    /* N_BIT set */
    num |= (1 << N_BIT);
    printf("(set) num : 0x%08x\n",num);
    printf("================\n");

    /* N_BIT clear */
    num &= ~(1 << N_BIT);
    printf("(clear) num : 0x%08x\n",num);
    printf("================\n");

    /* N_BIT toggle */
    num ^= (1 << N_BIT);
    printf("(toggle) num : 0x%08x\n",num);
    num ^= (1 << N_BIT);
    printf("(toggle) num : 0x%08x\n",num);
    num ^= (1 << N_BIT);
    printf("(toggle) num : 0x%08x\n",num);
    printf("================\n");

  return 0;
}
```

소스 코드

```
num : 0x00000000
================
(set) num : 0x00000008
================
(clear) num : 0x00000000
================
(toggle) num : 0x00000008
(toggle) num : 0x00000000
(toggle) num : 0x00000008
================
```

실행결과

함수 매개변수 전달

- 예제1 : Call-by-value & Call-by-reference

```c
#include<stdio.h>

#define EX1
//#define EX2

int increment_num1(int);
int increment_num2(int *);
void test_function(const int data1, const int *data2, int * const data3, const int * const data4);

int main(int argc, char *argv[])
{
#ifdef EX1
    /* Example 1 : function call by value/reference */
    int a = 5;
    int b = 5;

    fprintf(stdout, "======================== Test 1 ========================\n");
    fprintf(stdout, "Address of 'a' in 'main' :         %p\n", &a);
    fprintf(stdout, "Value of 'a' in 'main' (before function call) :    %d\n", a);
    increment_num1(a);
    fprintf(stdout, "Value of 'a' in 'main' (after function call) : %d\n", a);

    fprintf(stdout, "\n======================== Test 2 ========================\n");
    fprintf(stdout, "Address of 'b' in 'main' :         %p\n", &b);
    fprintf(stdout, "Value of 'b' in 'main' (before function call) :    %d\n", b);
    increment_num2(&b);
    fprintf(stdout, "Value of 'b' in 'main' (after function call) : %d\n", b);
#endif
```

소스 코드

```c
int increment_num1(int num)
{
    num++;

    fprintf(stdout, "Address of 'num' in 'increment_num1' :     %p\n", &num);
    fprintf(stdout, "Value of 'num' in 'increment_num1' :       %d\n", num);

    return 0;
}

int increment_num2(int *num)
{
    (*num)++;

    fprintf(stdout, "Address of 'num' in 'increment_num2' :     %p\n", num);
    fprintf(stdout, "Value of 'num' in 'increment_num2' :       %d\n", *num);

    return 0;
}
```

```
======================== Test 1 ========================
Address of 'a' in 'main' :                          0x2001ffdc
Value of 'a' in 'main' (before function call) : 5
Address of 'num' in 'increment_num1' :              0x2001ffcc
Value of 'num' in 'increment_num1' :                6
Value of 'a' in 'main' (after function call) :  5

======================== Test 2 ========================
Address of 'b' in 'main' :                          0x2001ffd8
Value of 'b' in 'main' (before function call) : 5
Address of 'num' in 'increment_num2' :              0x2001ffd8
Value of 'num' in 'increment_num2' :                6
Value of 'b' in 'main' (after function call) :  6
```

실행 결과

함수 매개변수 전달

- 예제2 : const 키워드

```
29
30  #ifdef EX2
31      /* Example 2 : function call with const keyword */
32      int w = 1;
33      int x = 2;
34      int y = 3;
35      int z = 4;
36
37      test_function(w, &x, &y, &z);
38  #endif
39
40      return 0;
41  }
```

```
63  void test_function(const int data1, const int *data2, int * const data3, const int * const data4)
64  {
65      /* Modify data1 */
66      data1 = 5;          // Error
67
68      /* Modify data2 */
69      data2 = &data1;
70      *data2 = data1;     // Error
71
72      /* Modify data3 */
73      data3 = &data1;     // Error
74      *data3 = data1;
75
76      /* Modify data4 */
77      data4 = &data1;     // Error
78      *data4 = data1;     // Error
79  }
```

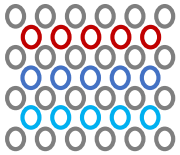
소스 코드

```
../src/main.cpp: In function 'void test_function(int, const int*, int*, const int*)':
../src/main.cpp:66:8: error: assignment of read-only parameter 'data1'
   66 |   data1 = 5;   // Error
      |   ~~~~~~~^~~
../src/main.cpp:70:9: error: assignment of read-only location '* data2'
   70 |   *data2 = data1;  // Error
      |   ~~~~~~~~^~~~~~~
../src/main.cpp:73:8: error: assignment of read-only parameter 'data3'
   73 |   data3 = &data1;  // Error
      |   ~~~~~~~^~~~~~~~
../src/main.cpp:77:8: error: assignment of read-only parameter 'data4'
   77 |   data4 = &data1;  // Error
      |   ~~~~~~~^~~~~~~~
../src/main.cpp:78:9: error: assignment of read-only location '*(const int*)data4'
   78 |   *data4 = data1;  // Error
      |   ~~~~~~~~^~~~~~~
make: *** [src/subdir.mk:40: src/main.o] Error 1
"make all" terminated with exit code 2. Build might be incomplete.
```

실행 결과

Q & A

Thank you for your attention



Architecture and Compiler
for Embedded Systems Lab.

School of Electronics Engineering, KNU
ACE Lab. (jcho@knu.ac.kr)