EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv1 과정

Ch2.

제 3기

2021. 12. 28

여건영

CONTENTS

EDDI
Electronics Design Development Institute

* x & ~(2의 y승 – 1)의 의미
* 변수와 메모리
* 16진수와 그 체계
* xor 을 이용한 대소문자 변환 방식
* Memory 구조 (stack, heap, data, text)
* 변수의 주소값과 실제 메모리 값의 일치성 확인(디버깅)
* 두 개의 변수를 입력받아 대소 비교
* 피보나치 수열 프로그래밍
* 주사위 게임
* 학습 내용 정리
* QnA

X & ~(2의 y승 -1)의 의미

```c
#include <stdio.h>
//비트 정렬
#include <time.h>       //time()함수 포함 라이브러리
#include <stdlib.h>     //rand()함수 포함 라이브러리
#define MAX     10
#define BIT_ALIGN       32
int main(void)
{
        int num;
        int i;
        srand(time(NULL));      //매번 다른 시드값 생성

        for (i=0;i<MAX;i++)
        {
                num=rand()%500+501;//501~1000
                //rand()%a+b    b ~ (a-1)+b 사이의 난수를 생성

                printf("bit 정렬전 결과 = %d, bit 정렬 결과 = %d\n", num, num&~(BIT_ALIGN-1));
        }
        return 0;
}
```

* x & ~(2의 y승 -1)의 의미는
  x가 어떤 특정 크기의 정수라도
  사용자가 지정한 2의 y승의 배수로 정렬 시켜 준다.

변수와 메모리 주소

* 변수 : 특정한 데이터 타입을 가지고 있으며 정보(data)를 저장할 수 있는 메모리 공간
  컴퓨터에서 사용하는 모든 정보는 메모리에 적재 되어진다. PC에서 메모리라 부르는 것이
  DRAM에 해당되며 DRAM에 데이터가 올라가는 것 자체가 변수에 해당됨

* 메모리 주소(Memory Address)
  메모리 한 칸은 1byte의 크기를 가짐
  32bit 운영체제에서는 4byte 길이의 주소를 가짐
  2^32 까지의 경우의 수가 있고, 4,294,967,296 개의 주소를 가리킬 수 있으며,
  이는 1바이트 크기의 메모리가 4,294,967,296 개 까지 인식이 가능하다는 것,
  즉 메모리의 최대 크기는 4GB

  64bit는 8byte이므로 하나의 주소가 8byte 길이의 주소를 가짐
  32bit에서는 0x00000000 ~ 0xFFFFFFFF
  64bit에서는 0x0000000000000000 ~ 0xFFFFFFFFFFFFFFFF

  메모리 한 칸의 크기는 1byte 이다.
  64비트 운영체제는 메모리 한 칸의 주소를 64bit로 표현하며
  이는 8byte와 같은 의미이고, 메모리 주소를 8바이트로 표현하기 때문에
  포인터(주소를 가리키는 변수)의 크기 또한 8바이트다.

16진수와 그 체계

10진수 : 표현할 수 있는 숫자가 10개 0 ~ 9
1856 // 10^3*1+10^2*8+10^1*5+10^0*6 과 같음
10^3 째 자리 숫자 : 1
10^2 째 자리 숫자 : 8
10^1 째 자리 숫자 : 5
10^0 째 자리 숫자 : 6

16진수 : 표현할 수 있는 숫자가 16개 0 ~ 9, A, B, C, D, E, F
0740 // 16^3*0+16^2*7+16^1*4+16^0*0 => 계산하면 1856
16^3 째 자리 숫자 : 0
16^2 째 자리 숫자 : 7
16^1 째 자리 숫자 : 4
16^0 째 자리 숫자 : 0

```
yeo@yeo-15Z980-GA50K:~/EmbeddedMasterLv1/3기/GYY/c/ch2$ ./16jinsu
정수를 입력하시오
221
10진수 출력 : 221
 8진수 출력 : 0335
16진수 출력 : 0xdd
```

진수들은 왜 사용할까?
컴퓨터는 진수의 표현을 전기적인 신호로 판단, 2진수 체계로 신호
있거나(1), 없거나(0) 둘 중 하나로 오류없이 판단할 수 있음

16진수는 16이 2의 4승 이기 때문에 2진수와 4대 1로 변환이 가능하다는 점에서 효율이 좋다
즉 메모리의 영역 부담을 줄일 수가 있음

xor을 이용한 대소문자 변환 방식

*ASCII 표 참조

A~Z : 0b 0100 0001 ~ 0b 0101 1010
a~z : 0b 0110 0001 ~ 0b 0111 1010
대문자와 소문자 간에는 10진수로 32, 즉 2진수로 0b0010 0000 만큼의 차이가 있다

왼쪽 그림에서 변수 c에 c^0x20을 하여 출력 시, // ^는 XOR 연산자로서 비트 단위가 같으면 0 아니면 1
0x20 : 0b 0010 0000

결과적으로 소문자에서는 32를 빼고,
대문자에서는 32를 더하면서
대소문자 변환을 가능하게 한다.

```c
#include <stdio.h>

int main(void)

{
        char c = '\0';

        printf("문자를 입력:");
        scanf("%c",&c);

        if((c>96&&c<123)||(c>64&&c<91))
        {
                printf("모든지 ok\n");
                printf("입력값 = %c, 변환값 = %c\n",c,c^0x20);
        }
        return 0;
}
```

Memory 구조(stack, heap, data, text)

[Text]
코드 영역, 실행할 프로그램의 코드가 저장되는 영역
CPU는 코드 영역에 저장된 명령어를 하나씩 가져가서 처리하게 됨
제어문, 함수, 상수 등이 이 영역에 저장됨

[Data]
데이터 영역은 작성한 코드에서 전역변수, 정적변수 등이 저장되는 공간
보통 메인(main)함수 전(프로그램 실행 전)에 선언되어 프로그램이 끝날 때 까지
메모리에 남아있는 변수로 프로그램이 종료되면 소멸함
초기화 된 변수 영역과 초기화되지 않는 변수 영역으로 나뉨

[Heap]
사용자에 의해 관리되는 영역, 동적으로 할당 할 변수들이 여기에 저장됨
힙 영역은 메모리의 낮은 주소에서 높은 주소의 방향으로 할당됨

[Stack]
스택 영역은 함수를 호출 할 때 지역변수, 매개변수들이 저장되는 공간
메인(main) 함수 안에서의 변수도 포함.
그리고 함수가 종료되면 해당 함수에 할당된 변수들을 메모리에서 해제시킴
Stack 영역은 Heap 영역과 반대로 높은주소에서 낮은주소로 메모리에 할당됨

Memory 구조(stack, heap, data, text)

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

const int constval = 30;        //상수

int uninitial;  //초기화되지 않는 전역변수
int initial = 30;       //초기화된 전역변수

static int staticval = 70;      //정적변수

int function(){
        return 0;
}

int main(void){
        int localval1 = 30;     //지역변수 1
        int localval2 = 20;     //지역변수 2

        printf(" 상수 Memory Address : \t\t %p \n", &constval);
        printf(" 비초기화 변수 Memory Address : \t %p \n", &uninitial);
        printf(" 초기화 변수 Memory Address : \t %p \n", &initial);
        printf(" 정적 변수 Memory Address : \t %p \n", &staticval);
        printf(" 함수 Memory Address : \t\t %p \n", &function);
        printf(" 지역변수1 Memory Address : \t %p \n", &localval1);
        printf(" 지역변수2 Memory Address : \t %p \n", &localval2);

        return 0;
}// 실행 후, 각 변수가 메모리에 할당되는 주소(위치)의 차이를 확인할 수 있음
```

```
상수 Memory Address :              0x55e269731008
비초기화 변수 Memory Address :     0x55e26973301c
초기화 변수 Memory Address :       0x55e269733010
정적 변수 Memory Address :         0x55e269733014
함수 Memory Address :              0x55e269730169
지역변수1 Memory Address :         0x7fff08fc4c10
지역변수2 Memory Address :         0x7fff08fc4c14
```

변수의 주소값과 실제 메모리값의 일치성 확인(디버깅)

```
(gdb) r
Starting program: /home/yeo/EmbeddedMasterLv1/3기/CYY/c/men_adder
var=7
var mem addr = 0xffffdf94
[Inferior 1 (process 8154) exited normally]
(gdb) b main
Breakpoint 1 at 0x555555555189: file men_adder.c, line 4.
(gdb) r
Starting program: /home/yeo/EmbeddedMasterLv1/3기/CYY/c/men_adder

Breakpoint 1, main () at men_adder.c:4
4       {
(gdb) n
5               int var = 7;
(gdb) n
6               printf("var=%d\n", var);
(gdb) disas main
Dump of assembler code for function main:
   0x0000555555555189 <+0>:     endbr64
   0x000055555555518d <+4>:     push   rbp
   0x000055555555518e <+5>:     mov    rbp,rsp
   0x0000555555555191 <+8>:     sub    rsp,0x10
   0x0000555555555195 <+12>:    mov    rax,QWORD PTR fs:0x28
   0x000055555555519e <+21>:    mov    QWORD PTR [rbp-0x8],rax
   0x00005555555551a2 <+25>:    xor    eax,eax
   0x00005555555551a4 <+27>:    mov    DWORD PTR [rbp-0xc],0x7
=> 0x00005555555551ab <+34>:    mov    eax,DWORD PTR [rbp-0xc]
   0x00005555555551ae <+37>:    mov    esi,eax
   0x00005555555551b0 <+39>:    lea    rdi,[rip+0xe6d]        # 0x555555556004
   0x00005555555551b7 <+46>:    mov    eax,0x0
   0x00005555555551bc <+51>:    call   0x5555555550b0 <printf@plt>
   0x00005555555551c1 <+56>:    lea    rax,[rbp-0xc]
   0x00005555555551c5 <+60>:    mov    rsi,rax
   0x00005555555551c8 <+63>:    lea    rdi,[rip+0xe5d]        # 0x55555555602c
   0x00005555555551cf <+70>:    mov    eax,0x0
   0x00005555555551d4 <+75>:    call   0x5555555550b0 <printf@plt>
   0x00005555555551d9 <+80>:    mov    eax,0x0
   0x00005555555551de <+85>:    mov    rdx,QWORD PTR [rbp-0x8]
   0x00005555555551e2 <+89>:    xor    rdx,QWORD PTR fs:0x28
   0x00005555555551eb <+98>:    je     0x5555555551f2 <main+105>
   0x00005555555551ed <+100>:   call   0x5555555550a0 <__stack_chk_fail@plt>
   0x00005555555551f2 <+105>:   leave
   0x00005555555551f3 <+106>:   ret
End of assembler dump.
(gdb) p/x &var
$1 = 0x7fffffffdf94
(gdb) p/x $rbp-0xc
$2 = 0x7fffffffdf94
(gdb) x/x $rbp-0xc
0x7fffffffdf94: 0x00000007
```

1. gdb 실행 후, r (프로그램 구동) 하여
   var 변수의 메모리 주소값 확인
   변수 var의 주소값은 0xffffdf94

2. 중단점 지정 후, 한 줄 씩 n(소스 코드 단위 실행) 하며
   disas(기계어 코드 확인) 하며 현재 위치 확인

3. 간단히 기계어 코드를 봤을 때,
   <+4> 스택의 시작점(rbp)를 스택에 넣음
   <+5> 스택의 최상단 지점(rsp)에 시작점(rbp) 값을 복사하
   <+8> 스택의 최상단 지점(rsp) 0x10만큼 빼줌
   <+27>스택의 시작점(rbp)에서 -0xc의 위치에 0x7 값을 복사

4. 해당 위치에서 var의 메모리 주소 값과
   스택의 시작점(rbp)에서 -0xc, 즉 $rbp-0xc의 메모리 주소 값을
   출력하고 그 값을 1번에서 출력한
   변수 var의 주소값 0xffffdf94와 일치성 확인

두 개의 변수를 입력받아 대소 비교

```c
#include <stdio.h>

int main(void){

        int a=0;
        int b=0;

        printf("1~100 사이의 정수만 입력하시오.\n");
        printf("a의 값을 입력하시오 : \n");
        scanf("%d", &a);
        printf("b의 값을 입력하시오 : \n");
        scanf("%d", &b);

        if(a>b)
        {
                printf("%d > %d, a가 b보다 더 큰 수 입니다.\n", a, b);
        }
        else if(a=b)
        {

                printf("%d > %d, a가 b의 값이 동일 합니다.\n", a, b);
        }
        else
        {

                printf("%d > %d, b가 a보다 더 큰 수 입니다.\n", b, a);
        }
        return 0;
}
```

```
1~100 사이의 정수만 입력하시오.
a의 값을 입력하시오 :
50
b의 값을 입력하시오 :
30
50 > 30, a가 b보다 더 큰 수 입니다.
```

피보나치 수열 프로그래밍

```c
#include <stdio.h>

int main(void)
{
        int n_0=0, n_1=1;
        int a, temp, i;

        printf("피보나치 수열의 몇 번째 열까지 출력할까요?\n");
        scanf("%d", &a);

        printf("피보나치 수열의 %d번째 열까지 출력하겠습니다.\n", a);

        for(i=0;i<a;i++)
        {
                printf("\t%7d", n_1);
                temp=n_1;
                n_1=n_1+n_0;
                n_0=temp;

                if((i+1)%5==0)
                {
                        printf("\n");
                }
        }
        return 0;
}
```

```
피보나치 수열의 몇 번째 열까지 출력할까요?
15
피보나치 수열의 15번째 열까지 출력하겠습니다.
              1       1       2       3       5
              8      13      21      34      55
             89     144     233     377     610
```

주사위 게임

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

int main(void)
{
        int num;
        int mynum;
        srand(time(NULL));

        printf("당신의 주사위 숫자는 몇 입니까?\n");
        scanf("%d", &mynum);

        num=rand()%6+1;

        if(num>mynum)
        {
                printf("당신은 졌습니다.\n");
        }
        else if(num==mynum)
        {
                printf("비겼습니다.\n");
        }
        else
        {
                printf("당신이 이겼습니다.\n");
        }
return 0;
}
```

```
당신의 주사위 숫자는 몇 입니까?
5
당신이 이겼습니다.
yeo@yeo-15Z980-GA50K:~/EmbeddedMaste
당신의 주사위 숫자는 몇 입니까?
1
당신은 졌습니다.
yeo@yeo-15Z980-GA50K:~/EmbeddedMaste
당신의 주사위 숫자는 몇 입니까?
3
당신은 졌습니다.
yeo@yeo-15Z980-GA50K:~/EmbeddedMaste
당신의 주사위 숫자는 몇 입니까?
4
비겼습니다.
```

학습내용 정리

[논리연산자]
1. '&&' : AND 연산자로 두 수를 비교하여 둘다 참이면 1 아니면 0
2. '||' : OR 연산자로 두 수를 비교해서 둘 중 하나라도 참이면 1 아니면 0
3. '!' : NOT 연산자로 참이면 거짓, 거짓이면 참으로 값의 반전

[비트연산자]
1. '&' : AND 연산자로 두 수를 비교하여 같은 자리의 비트 단위가 서로 1이면 1, 하나라도 다르면 0
2. '|' : OR 연산자로 두 수를 비교하여 둘 중 하나라도 비트 단위로 1이면 1 아니면 0
3. '^' : XOR 연산자로 두 수를 비교하여 비트 단위가 같으면 0 서로 다르면 1
4. '~' : NOT 연산자로 비트 단위를 반전하는 연산자로 0이면1, 1이면 0으로 반전
5. '<<' : 비트를 왼쪽으로 쉬프트 하는 연산자로 원하는 만큼 왼쪽으로 비트 이동
6. '>>' : 비트를 오른쪽으로 쉬프트 하는 연산자로 원하는 만큼 오른쪽으로 비트 이동