2018. 02. 27 화 <5 회차>

과정 : TI, TSP, MCU 및 Xilinx Zynq FPGA 프로그래밍 전문가 과정

1. 재귀함수 복습
2. 배열 – Array
 2.1 Character type Array
 2.2 High Dimension Array
 2.3 배열 초기화
3. 포인트 – Point

1. 재귀함수 복습
 if, while, goto 등을 사용할 시에는 jmp 를 사용

Ex> 피보나치수열 : 재귀호출을 while 로 바꿈

```
#include <stdio.h>

//while 로 재귀호출을 대신한 형태
// \n : 한 줄을 내린다.

int fib(int num)
{
   int first = 1;
   int second = 1;
   int tmp = 0;

   if(num == 1 || num == 2)
     return 1;

   while(num-- > 2)
   {
     tmp = first + second;
     first = second;
     second = tmp;
    }
    return tmp;
}

int main(void)
{
        int result, final_val;
        printf("피보나치 수열의 항의 개수를 입력하시오:");
        scanf("%d", &final_val);
        result = fib(final_val);
        printf("%d 번째 항의 수는 = %d\n", final_val, result);

        return 0;
}
```

Ex> 팩토리얼 : 재귀호출

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

```
int fact(int num)
{

   if(num == 0)
     return 1;
   else
     return num * fact(num - 1);
}

int main(void)
{
   int result, fact_val;
   printf("계산할 팩토리얼을 입력하시오:");
   scanf("%d", &fact_val);
   result = fact(fact_val);
   printf("%d 번째 항의 수는 = %d\n", fact_val, result);
   return 0;
}
```

Ex>팩토리얼 : 재귀호출을 while 로 바꿈

```
#include <stdio.h>

//while 을 통하여 재귀호출을 대신한 형태

int fact(int num)
{
   int first = 1;
   while(num > 0)
     first = first*num--;
   return first;
}

int main(void)
{
   int result, fact_val;
   printf("계산할 팩토리얼을 입력하시오:");
   scanf("%d", &fact_val);
   result = fact(fact_val);
   printf("%d 번째 항의 수는 = %d\n", fact_val, result);
   return 0;
}
```

2. 배열 – Array

배열이 필요한 이유 : 변수가 많은 경우

다수의 변수를 사용할 경우 for 문을 이용하여 간단히 작업가능

배열 선언 방법(형태) : data type - filename – data 갯수 = 초기화 ; (index 는 0 부터 시작)

Ex> int sensor_data [ ] = {0};

참고. data type : int, character, float, double 등

배열은 메모리상에서 순차적으로 배열이 되어있다.

그러므로 2 차원, 3 차원배열보다는 이중, 삼중배열이 옳은 표현

num	0	1	2	3	4	5	6
Index	0	1	2	3	4	5	6

Ex> 배열생성예제

```
#include <stdio.h>

//배열예제.
//

int main(void)
{
  int i;
  int num[7];

  for(i = 0; i < 7; i ++)
  {
    num[i] = i + 1;
    printf("num[%d] = %d\n", i, num[i]);
  }
  return 0;
}
```

Ex>

```
#include <stdio.h>

//arr[]에서 원소가 5 개이므로 자동으로 5 가 채워진다.




int main(void)
{
   int i;
   int num1_arr[] = {1, 2, 3, 4, 5};
   int num2_arr[3] = {1, 2, 3};

   int len1 = sizeof(num1_arr)/sizeof(int);     //배열에 몇 개의 data 가 있는지 의미 : 배열전체의 사이즈를 하나의 데이터타입의 사이즈로 나눔
   int len2 = sizeof(num2_arr)/sizeof(int);

   printf("num1_arr length = %d\n", len1);
   printf("num2_arr length = %d\n", len2);

   for(i = 0; i < len1; i++)
   {
     printf("num1_arr[%d] = %d\n", i, num1_arr[i]);
   }

   for(i = 0; i < len2; i++)
   {
     printf("num2_arr[%d] = %d\n", i, num2_arr[i]);
```

```
    }
    return 0;
}
```

Ex> 배열의 값을 전부 초기화시켜주지 않은 경우 : 0 으로 인식

```
#include <stdio.h>

// 배열의 갯수를 7 개로 설정했으나, 실제 3 개만 주었을 경우에는 나머지는 전부 0(초기화)이 된다.

int main(void)
{
    int i;
    int num1_arr[7] = {1, 2, 3};

    for(i = 0; i < 7; i++)
    {
        printf("num1_arr[%d] = %d\n", i, num1_arr[i]);
    }

    return 0;
}
```

2.1 Character type Array

Character type Array 가 필요한 이유

- string 인 문자열 “I’m Marth Kim”은 변경이 불가능
  char 형 배열(1byte)은 내부데이터변경이 가능하다.
  마지막 data 에 Null character(\0) 필요함, 없으면 어디까지인지 몰라 인식이 불가능(But, 최근에는 compiler 가 알아서 해석)
  Null character : 문자열의 마지막을 의미

아래 두 코드는 동일한 형태

```
  int d[10] = {     };
  char str[32] = “    “;
```

Ex>

```
#include <stdio.h>

/* 두가지 포인트
   1. 기존에는 str 3 에 \0 가 없어서 끝이 어딘지 몰라서 제대로 출력이 안나왔지만, 현재는 컴파일러가 똑똑해져서 알아서 마무리
   2. 마지막에서는 char 는 원하는 문자를 바꿀수 있다는 이야기
   추가. "" 와 {}에서 []안에는 +1 만큼 작성해주어야함
        {} : 캐릭터하나씩, "" : 문자열
int main(void)
{
    char str1[5] = "AAA";
    char str2[]  = "BBB";
    char str3[]  = {'A', 'B', 'C'};
    char str4[]  = {'A', 'B', 'C', '\0'};

    printf("str1 = %s\n", str1);
```

```
    printf("str2 = %s\n", str2);
    printf("str3 = %s\n", str3);
    printf("str4 = %s\n", str4);

    str1[0] = 'E';
    str1[1] = 'H';
    printf("str1 = %s\n", str1);

    return 0;
}
```

2.2 High Dimension Array
다중배열
이중배열에서는 행령의 표현이 용이
삼중배열에서는 텐서의 표현이 용이

그림추가

Ex>

```
#include <stdio.h>

// printf("\n'); 한 줄 밑으로 가라는 의미

int main(void)
{
    int arr[4][4];
    int i, j;

    for(i = 0; i < 4; i++)
    {
        for(j = 0; j < 4; j++)
        {
            if(i == j)
                arr[i][j] = 1;
            else
                arr[i][j] = 0;
        }
    }
    for(i = 0; i < 4; i++)
    {
        for(j = 0; j < 4; j++)
        {
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

3.3 배열의 초기화

초기화 : 배열에 들어가는 값을 정해주는 행위

Ex> 다중배열의 초기화

```
/* 다중배열의 경우 초기화방법    -     초기화 : stack 에 처음값을 지정해주는 행위
arr[2][2][3] = {
              {{1,2,3}, {1,2,3}},
              {{1,2,3}, {1,2,3}}
             };
   처음 코드만 보고 이해가 가지 않을 경우, 그림을 그려서 이해
*/

#include <stdio.h>

int main(void)
{
   int arr[2][2]  =  {{10, 20}, {30, 40}};
   int i, j;

   for(i = 0; i < 2; i++)
   {
      for(j = 0; j < 2; j++)
      {
         printf("arr[%d][%d] = %d\n", i, j, arr[i][j]);
      }
   }

   return 0;
}
```

Ex> 배열의 이름은 주소 : 그 주소는 첫 번째 배열의 주소

```
// '배열의 이름은 주소'임을 보여주는 예제(참고로 지금까지 나오는 모든 주소는 가짜 주소)  -  더 공부해보고싶으면, 가상메모리, 페이징(paging)을 볼 것
// 주소값들의 차이가 4 - 배열이 int 형이므로 4byte 씩 나므로 일렬로 나열되있음
// 배열의 대표는 배열의 이름 : 그러므로 맨 앞의 주소가  배열의 대표
//%p 는 주소값 꾸릴때 사용
// & - 주소를 가르키는 값

#include <stdio.h>

int main(void)
{
   int arr[4] = {10, 20, 30, 40};
   int i;

   printf("address = %p\n", arr);

   for(i = 0; i < 4; i++)
   {
      printf("address = %p, arr[$d] = %d\n", &arr[i], i, arr[i]);
   }
```

```
    return 0;
}
```

배열의 결론

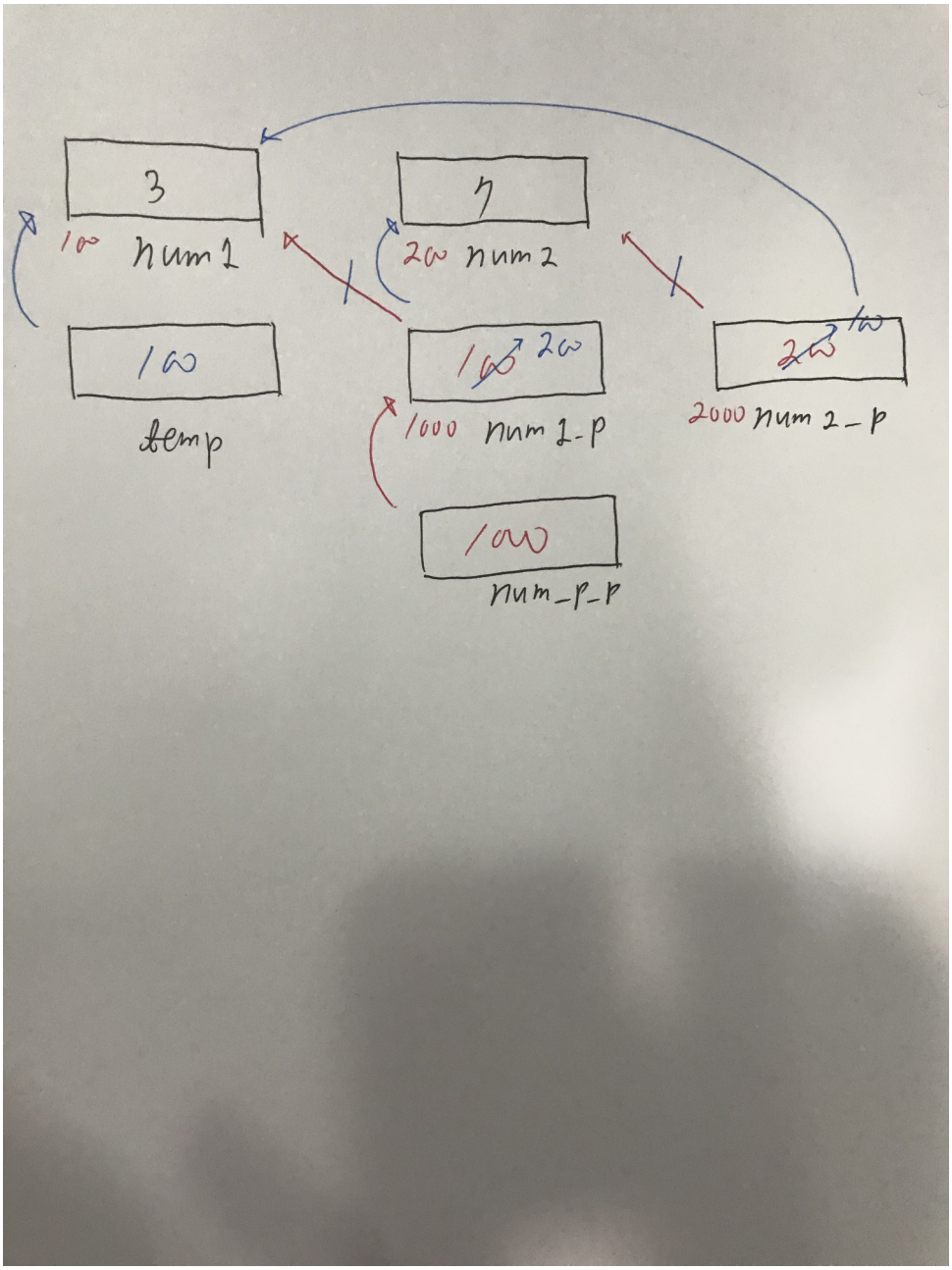
배열 = 주소

참고
기본 배열과 이중배열의 관계 : 이중배열은 기본배열을 포함한 것

3. 포인터
정의 : 주소를 저장할 수 있는 공간
사용하는 이유 : 어떤 위치에 존재하는 주소를 불러오기 위하여 사용
포인트 선언 방법(형태)
int num = 7;
int *p = #
int 와 *이 합쳐져 하나의 data type 을 이룬다.(* 과 p 가 하나의 느낌이 아님)

Ex> 주소값을 중간에 수정해주는 포인트예제
// 꽤 재미있는 예제

```
#include <stdio.h>

int main(void)
{
  int num1 = 3, num2 = 7;
  int *temp = NULL;
  int *num1_p = &num1;
  int *num2_p = &num2;
  int **num_p_p = &num1_p;

  printf("num1_p = %d\n", *num1_p);
  printf("num2_p = %d\n", *num2_p);

  temp = *num_p_p;
  *num_p_p = num2_p;
  num2_p = temp;

  printf("*num1_p = %d\n", *num1_p);
  printf("*num2_p = %d\n", *num2_p);

  return 0;
}
```


3
10
num1
7
20
num2
10
temp
10
20
1000
num1_p
20
2000
num2_p
10
num_p_p