배열과 포인터

배열

- 연속된 변수들을 쉽게 사용하기 위해 고안된 개념
- 변수명[배열의 크기]와 같은 형식으로 사용됨
- 앞서 배운 for문과 함께 편리하게 사용할 수 있음
- 1차원으로 사용되는 일반적인 배열부터, 축과 크기를 추가하여 변수명[x축 크기][y축 크기]... 과 같이 다차원 배열로 사용할 수 있음
- 문자열
  - char형의 변수의 배열이라 볼 수 있음
  - NULL문자 ('\0') 가 오면 문자열의 끝으로 간주함
  - 관련 함수로 strlen( ), strcmp( ) 등이 있음
    - strlen ( ) : strlen(문자열) 형식으로 사용하여 문자열에서 NULL이 오기 전까지의 길이를 반환함
    - strcmp ( ) : strcmp(문자열 1, 문자열 2) 형식으로 사용하여 두 문자열이 동일한지 여부를 반환함
      - 두 문자열의 ASCII 코드값을 비교하여 같을 경우엔 0, 1번 인자의 값이 크면 1, 2번 인자의 값이 크면 -1을 반환함

```
#include <stdio.h>

int main(){
    int i_arr[10], n = 10;

    //배열은 i_arr[0]부터, i_arr[9]까지 구성됨
    for(int i = 0; i < n; i++){
        i_arr[i] = i;
    }

    //배열의 byte는 자료형 byte * 배열의 크기
    printf("배열 byte : %lu\n", sizeof(i_arr));

    //자료형을 알고 크기가 정해지지 않은 배열을 이용할 때, sizeof를 이용하면 간단함
    for(int i = 0; i < sizeof(i_arr) / sizeof(int); i++){
        printf("%d ", i_arr[i]);
    }
    printf("\n");

    //배열의 활용, 역순으로 출력
    for(int i = sizeof(i_arr) / sizeof(int) - 1; i >= 0; i--){
        printf("%d ", i_arr[i]);
    }
    printf("\n");

    //배열의 활용, 최댓값 출력
    int i_max = 0;
    for(int i = 0; i < sizeof(i_arr) / sizeof(int); i++){
        if(i_arr[i] > i_max){
            i_max = i_arr[i];
        }
    }
    printf("최댓값 : %d\n", i_max);

    //배열의 활용, 홀수의 개수
```

```
    int i_cnt = 0;
    for(int i = 0; i < sizeof(i_arr) / sizeof(int); i++){
        if(i_arr[i] % 2 != 0){
            i_cnt++;
        }
    }
    printf("홀수의 개수 : %d\n", i_cnt);

    //다차원 배열, 파스칼의 삼각형
    int i_2darr[10][10] = {0};
    for(int i = 0; i < 10; i++){
        for (int j = 0; j <= i; j++) {
            if(j == 0 || j == i){
                i_2darr[i][j] = 1;
            }
            else{
                i_2darr[i][j] = i_2darr[i - 1][j - 1] + i_2darr[i - 1][j];
            }
            printf("%d ", i_2darr[i][j]);
        }
        printf("\n");
    }

    return 0;
}
```

```
실행 결과

배열 byte : 40
0 1 2 3 4 5 6 7 8 9
9 8 7 6 5 4 3 2 1 0
최댓값 : 9
홀수의 개수 : 5
1
1 1
1 2 1
1 3 3 1
1 4 6 4 1
1 5 10 10 5 1
1 6 15 20 15 6 1
1 7 21 35 35 21 7 1
1 8 28 56 70 56 28 8 1
1 9 36 84 126 126 84 36 9 1
```

포인터

- Pointer : 변수의 주소를 저장하는 변수
  - 선언 : 일반적인 변수 선언 방법과 같으나, 변수명 앞에 *를 붙여서 선언 *ex) int *p_pointer;*
  - 변수의 주소를 저장하므로, 초기화 할 때도 변수의 주소를 반환하는 &와 함께 사용됨 *ex) p_pointer = &i_a;*
  - 선언이 아닌 경우 변수를 사용할 때, 포인터 변수에 *을 붙여 사용하면 포인터 변수가 가지고 있는 주소가 가지고 있는 변수의 값을 반환함 *ex) result = *p_pointer // result = i_a*
  - 포인터 (*)을 중첩해서 사용하면 포인터를 가리키는 포인터 변수 또한 만들 수 있으나, 잘 사용하지 않음.
  - 포인터의 연산
    - 포인터에 +1을 하게 되면, 포인터가 가지고 있는 주소값 + 1이 될거라 생각되지만 그렇지 않음

        - 포인터에 +1을 하게 되면, 포인터가 가지고 있는 주소값 + sizeof(포인터가 가리키는 변수) * 1만큼이 더해져, 현재 가지고 있는 변수의 다음 주소를 갖게 됨 (배열로 생각하면 배열을 한 주소를 가리키는 포인터 + 1은 배열의 다음 요소를 가리키게 된다는 것)
    - 배열 포인터
        - 배열을 가리키는 포인터 변수를 만들면, 배열과 포인터는 완벽히 동일하게 사용할 수 있음 *ex) int i_arr[3]; *p_ptr = arr; 이라면, p_ptr과 arr은 동일*
        - 컴퓨터가 배열을 처리하는 방식
            - 우리가 배열을 i_arr[i]와 같이 사용한다면, 컴퓨터는 이 식을 *(i_arr + i)로 변환시켜 사용함
            - 앞에서 말한 배열 포인터의 성질 때문에, *(i_arr + i) == *(p_ptr + i) == *(i + p_ptr) == *(i + i_arr)이 성립함
            - 이 식을 다시 우리가 배열을 사용하는 형식으로 바꾸면, i_arr[i] == i[i_arr]이 성립함
        - 배열을 가리키는 포인터는 배열의 [0]의 주소값을 갖지만, 그 배열 자체를 가리키는 것으로 작동함
            - 따라서 이 포인터에 + 1을 하면, 배열 하나를 변수처럼 생각해 배열 끝의 다음 주소를 가리킴
        - 2차원 배열에서의 배열 포인터
            - 2차원 배열은 어떻게 보면 배열 안의 배열이라 생각할 수 있음
            - i_arr[y] [x]는, i_arr[x]를 하나의 변수로 가지는 [y] 크기의 배열과 같음
            - 따라서 배열 포인터를 사용하여, i_arr을 가리키는 포인터 *p_ptr를 만들면, p_ptr + 1 은 i_arr[y][1]을 가리키게 될 것
    - 포인터 배열
        - 포인터들을 배열로 묶어놓은 것
        - 배열 포인터와 선언 방법이 비슷하여 혼동하지 않는 것이 중요함

```
#include <stdio.h>

int main(){
    int i_a = 1;

    //포인터 변수의 선언
    int *p_ptr = &i_a;

    //포인터 변수의 값
    printf("포인터가 저장하는 값 : %d, 포인터가 가리키는 변수의 값 : %d\n", p_ptr, *p_ptr);

    //더블 포인터
    int **pp_ptr = &p_ptr;
    printf("**pp_ptr : %d, *p_ptr : %d, i_a : %d\n", pp_ptr, *pp_ptr, **pp_ptr);

    //포인터의 연산
    int i_arr[10];
    for(int i = 0; i < 10; i++){
        i_arr[i] = i;
    }

    printf("&i_arr[2] : %d, &i_arr[3] : %d, &i_arr[2] + 1 : %d\n", &i_arr[2], &i_arr[3], &i_arr[2] + 1);

    //배열 포인터
    int (*p_ptarr)[10] = &i_arr;
    printf("배열 포인터 *p_ptarr : ");
```

```
    for (int i = 0; i < 10; i++) {
        printf("%d ", (*p_ptarr)[i]);
    }
    printf("\n");

    //2차원 배열과 배열 포인터
    int i_2darr[2][3];
    for(int i = 0; i < 2; i++){
        for (int j = 0; j < 3; j++) {
            i_2darr[i][j] = i * 3 + j;
        }
    }
    int (*p_pt2darr)[3] = i_2darr;

    for(int i = 0; i < 2; i++){
        for (int j = 0; j < 3; j++){
            printf("%d ", p_pt2darr[i][j]);
        }
        printf("\n");
    }

    //포인터 배열
    char c_arr[4][10] = {"What", "FXXX", "Pointer", "\\_/"};
    char *p_str[4];

    for(int i = 0; i < 4; i++){
        p_str[i] = c_arr[i];
    }
    for(int i = 0; i < 4; i++){
        printf("%s\n", p_str[i]);
    }

    return 0;
}
```

```
실행 결과

포인터가 저장하는 값 : -283403944, 포인터가 가리키는 변수의 값 : 1
**pp_ptr : -283403952, *p_ptr : -283403944, i_a : 1
&i_arr[2] : -283403816, &i_arr[3] : -283403812, &i_arr[2] + 1 : -283403812
배열 포인터 *p_ptarr : 0 1 2 3 4 5 6 7 8 9
0 1 2
3 4 5
What
FXXX
Pointer
\_/
```

Section3. 배열과 포인터 종합문제

- 100개 이하의 정수를 입력받아 첫 줄에 짝수 번째 숫자들을 순서대로 출력하고, 다음 줄에 홀수 번째 숫자들을 순서대로 출력하는 프로그램을 만들어 보세요

```
#include <stdio.h>

int main(){
    int n = 0;
    int i_num[100];

    scanf("%d", &n);

    for(int i = 0; i < n; i++){
```

```
        scanf("%d", &i_num[i]);
    }

    for (int i = 0; i < n; i++) {
        if(i % 2 != 0)
        {
            printf("%d ", i_num[i]);
        }
    }
    printf("\n");

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

    return 0;
}
```

```
입력

7
3 1 4 1 5 9 2

실행 결과

1 1 9
3 4 5 2
```

- 코드를 보고 출력값을 예상해 보세요 _ 1

```
#include <stdio.h>

int main() {
    int a = 10;
    int b = 20;

    int *ptr;

    ptr = &a;
    *ptr = 30;  // a에는 30이 대입

    ptr = &b;
    *ptr = 10;  // b에는 10이 대입

    printf("%d\n", a); // 30
    printf("%d\n", b); // 10
    printf("%d\n", *ptr); // b를 가리키고 있는 상태
}
```

- 코드를 보고 출력값을 예상해 보세요 _ 2

```
#include <stdio.h>

int main() {
    int arr[10] = { 3, 1, 4, 1, 5, 9, 2, 6, 5, 3 };

    printf("%d\n", arr); // arr 주소
    for (int i = 3; i < 7; i++) {
        printf("%d %d\n", arr + i, *(arr + i));
```

```
        // 3 * sizeof(int) = 12
        // 따라서 arr주소 + 12 만큼이 출력되고
        // 그 주소의 값인 1이 출력
        // 같은 방식으로 반복될 것이다
    }
}
```

- 10 * 10 이하의 정수형 이차원 배열을 입력받아 그 배열의 각 행의 합을 출력하는 프로그램을 만들어 보세요

```
#include <stdio.h>

int main() {
    int i_arr[10][10];

    int x, y, i_cnt;
    scanf("%d %d", &x, &y);

    for(int i = 0; i < x; i++){
        for(int j = 0; j < y; j++){
            scanf("%d", &i_arr[i][j]);
        }
    }

    for(int i = 0; i < x; i++){
        i_cnt = 0;
        for(int j = 0; j < y; j++){
            i_cnt += i_arr[i][j];
        }
        printf("%d\n", i_cnt);
    }
    return 0;
}
```

```
입력

3 4
4 2 6 3
7 9 3 4
5 1 2 1

실행 결과

15
23
9
```

- 코드를 보고 출력값을 예상해 보세요 _ 3

```
#include <stdio.h>

int main() {
  int arr[3][3] = { 0 }; // 배열 초기화

  printf("%d\n", &arr); // 2차원 배열 전체 주소인 &arr[0][0]
  printf("%d\n", arr); // 2차원 배열 중 0행의 주소인 &arr[0][0]
  printf("%d\n", *arr); // 2차원 배열 중 0행의 주소인 &arr[0][0]

  printf("%d\n", &arr[0]); // arr과 동일
  printf("%d\n", arr[0]); // 2차원 배열 중 0행의 주소인 &arr[0][0]
  printf("%d\n", *arr[0]); // arr[0][0]의 값 : 0

  printf("%d\n", &arr[0][0]); // 2차원 배열 중 arr[0][0]의 주소인 &arr[0][0]
```

```
  printf("%d\n", arr[0][0]); // arr[0][0]의 값 : 0
}
```

- 코드를 보고 출력값을 예상해 보세요 _ 4

```
#include <stdio.h>

int main() {
  int arr[3][3] = { 0 };

  printf("%d\n", &arr[0][0]); // arr[0][0]의 주소

  printf("%d\n", arr[0] + 1); // arr[0] -> arr[0][0]의 주소, + 1 -> arr[0][1]의 주소
  printf("%d\n", &arr[0] + 1); // &arr[0] : 2차원 배열 첫번째 행을 가리키는 포인터
  printf("%d\n", arr + 1); // &arr[0] == arr
  printf("%d\n", &arr + 1); // &arr: 2차원 배열 전체를 가리키는 포인터
}
```

- 코드에서 틀린 부분을 예상해 보세요

```
#include <stdio.h>

int main() {
  int arr[3][4] = {
    { 1, 2, 3, 4 },
    { 5, 6, 7, 8 },
    { 9, 10, 11, 12 }
  };

  int(*ptr)[4] = arr;

  for (int i = 0; i < 3; i++) {
    for (int j = 0; j < 4; j++) {
      printf("%d ", *ptr[i][j]); //포인터 자체를 배열로 사용 가능하므로, *는 필요가 없다
    }
    printf("\n");
  }
}
```