코벤져스's PICK

빨간색으로 강조된 부분이 라이브에 언급될 부분입니다.

미션1

12-6팀

1. 배열 선언 : int arr[6][5] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {11, 12, 13, 14, 15}, {16, 17, 18, 19, 20}, {21, 22, 23, 24, 25}, {26, 27, 28, 29, 30}};

2. 배열 포인터로 가리키기 : int (*p)[5] = arr;

3. 중첩 루프를 이용해서 순차적으로 출력 :

```
for (i = 0; i <6 ; i++) {
        for (j = 0; j < 5 ; j++) {
        }
}
```

<Seul 님이 프로그래밍한 코드>

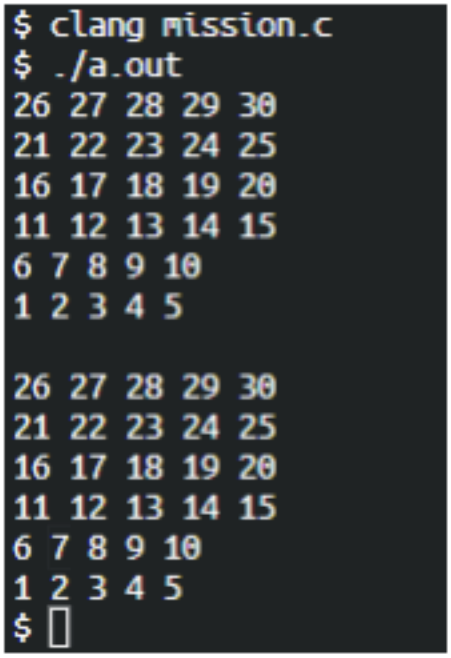
```c
#include <stdio.h>

int main()
{
    int arr[6][5] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {11, 12, 13, 14, 15},
    {16, 17, 18, 19, 20}, {21, 22, 23, 24, 25}, {26, 27, 28, 29, 30}};

    int (*p)[5] = arr;
    int i, j;

    for (i = 0; i <6 ; i++) {
        for (j = 0; j < 5 ; j++) {
            printf("%d\t", (*(p+i))[j]);
            }
        printf("\n");
    }
    return 0 ;
}
```

<최종 코드 – 배열 접근과 이중 포인터로 배열 출력하기>

```c
#include <stdio.h>

int main(){

    int  arr[6][5] = {{1, 2, 3, 4, 5}, {6, 7, 8, 9, 10}, {11, 12, 13, 14, 15},
                      {16, 17, 18, 19, 20}, {21, 22, 23, 24, 25}, {26, 27, 28, 29, 30}};

    // 행의 크기
    int row = sizeof(arr)/sizeof(arr[0]);
    // 열의 크기
    int col = sizeof(arr[0])/sizeof(int);

    //이중 for문 배열 접근해서 푸는 방법
    for(int i=row-1;i>-1;i--){
        for(int j =0; j < col; j++){
            printf("%d ", arr[i][j]);
        }
        printf("\n");
    }
    printf("\n");

    //이중 포인터로 접근하는 방법
    for(int i=row-1;i>-1;i--){
        for(int j =0; j < col; j++){
            printf("%d ", *( *(arr+i) + j));
        }
        printf("\n");
    }
    return 0;
}
```

1) row 와 col 의 실제 값을 모르는 경우를 생각한다면 sizeof()로 직접 계산해서 구할 수 있다. 행은
전체 이차원 배열로 한 행의 크기로 나눈 값으로 구하고 열은 한 행을 int 로 나눠 크기를 구할
수 있다.
2) 배열을 접근할 때 먼저 int (*p)[5] = arr; 한 것에서, 한 번 더 포인터로 가리킬 수 있다. = 이중
포인터로 이차원 배열 접근 -> * (*(arr + i) + j)
3) arr[i] 대신, *(arr + i) 포인터로 출력할 수 있다.

<출력 결과>


$ clang mission.c
$ ./a.out
26 27 28 29 30
21 22 23 24 25
16 17 18 19 20
11 12 13 14 15
6 7 8 9 10
1 2 3 4 5
26 27 28 29 30
21 22 23 24 25
16 17 18 19 20
11 12 13 14 15
6 7 8 9 10
1 2 3 4 5
$

미션2

13-8팀

1) 수업에서 언급되었던 Heap overflow 와 stack overflow 에 대해서 어떤 경우에 발생이 되는지 서술해주세요.

heap overflow

heap 영역의 버퍼가 넘쳐 인접한 영역을 침범하는 경우 발생

예) 동적 할당 malloc() 함수를 해제하지 않고 계속해서 사용하는 경우

stack overflow

프로그램이 호출 스택에서 이용 가능한 공간 이상을 사용하려고 시도할 때 발생

예) 탈출 조건이 없어 자기자신을 무한히 호출하는 재귀함수를 사용하는 경우

2) Strcpy 와 strncpy 의 차이점을 서술해보세요. (어떤 것을 추천하는지와 그 이유에 대해서 서술해주세요.)

char* strcpy(char* destination, const char* source);

char* strncpy(char* destination, const char* source, size_t num);

strcpy 는 source 의 널문자(\0)까지 포함한 전체 문자열을 destination 으로 복사한다. 복사 받으려는 배열의 크기가 원본 배열의 크기보다 작으면 오버 플로우가 발생한다.

strncpy 는 size_t 인자를 갖고 있어서 source 의 num byte 만큼만 destination 으로 복사한다. 이때 널문자가 포함되지 않을 수도 있으므로 사용한 뒤에 destination[num] = '\0'를 추가해줘야 한다. 하지만 오버 플로우를 예방할 수 있기 때문에 안정성 측면에서 이 함수를 쓰는 것을 추천한다.

3) 메모리 초기화, 복사, 이동, 비교와 같은 함수가 라이브러리에 있습니다. 사용방법을 숙지하고, 간단하게 코드로 구현 후 정상적으로 동작이 되는지 확인해보세요.

- **메모리 초기화(memset)**

void* memset(void* ptr, int value, size_t num);

ptr 로 시작하는 메모리 주소로부터 num byte 만큼 value 값으로 채운다.

초기화 전

```
#include <stdio.h>

int main(void)
{
```

```
    int arr[5];

    for (int i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

출력 결과

```
4334640 0 4205136 0 -1058801440
```

초기화 후

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

int main(void)
{
    int arr[5];

    memset(arr, 0, sizeof(arr));

    for (int i = 0; i < 5; i++)
        printf("%d ", arr[i]);
    printf("\n");
}
```

출력 결과

```
0 0 0 0 0
```

- **메모리 복사(memcpy)**

void* memcpy(void* destination, const void* source, size_t num);
source 주소에 있는 데이터를 destination 주소로 num byte 만큼 복사

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

int main(void)
{
    char dest[] = "0000";
    char source[] = "1111";

    memcpy(dest, source, sizeof(dest));

    printf("%s\n", dest);
}
```

출력 결과

```
1111
```

- **메모리 이동(memmove)**

void* memmove(void* destination, const void* source, size_t num);
source 주소에 있는 데이터를 destination 주소로 num byte 만큼 복사
(기능적으로 memcpy 와 동일)

<memcpy 와 memmove 차이점>

- memcpy 는 src 에서 dest 로 메모리를 바로 복사
- memmov 는 버퍼에 저장했다가 버퍼에서 dest 로 복사
- 속도는 memcpy 가 더 빠름
- 안정성은 memmove 가 더 높음

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

int main(void)
{
    char dest[] = "0000";
    char source[] = "1111";

    memmove(dest, source, sizeof(dest));

    printf("%s\n", dest);
}
```

출력 결과

```
1111
```

- **메모리 비교(memcmp)**

int memcmp(const void* ptr1, const void* ptr2, size_t num);

ptr1 과 ptr2 부터 시작하는 메모리 공간을 num byte 만큼 비교해서 같으면 0 아니면 다른 값 리턴

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

int main(void)
{
    char *str1 = "this is apple";
    char *str2 = "this is orange";

    if (memcmp(str1, str2, strlen(str1)) == 0){
        printf("same string\n");
    }else{
        printf("different string\n");
    }
}
```

출력 결과

```
different string
```

미션3

5-2팀

```
#include <stdio.h>

//함수 선언
void sort(int number, int *array);
//포인터를 사용해서 함수 자체가 반환하는 값이 없으므로 반환값 형식자는 void

int main()
{
    int n = 7; //배열의 길이
    //지정되어 있지 않으면 sizeof(arr)/sizeof(int)로 계산해야 함
    int arr[7] = { 0, 25, 10, 17, 6, 12, 9 };
    sort(n, arr[1]); //배열 이름은 주소(배열의 첫번째 칸 메모리 주소)랑 같음

    return 0;
}

void sort(int n, int *array){
    //3 번 간단한 버블 정렬 코드를 배열이 아닌 포인터를 활용
    //버블정렬: 처음부터 2 개의 숫자씩 짝지어 가며 정렬
    int temp, i, j, k;
    for(i = 0; i < n; i++){
        //최악의 상황을 뜻하는 big O 가 n^2 이기 때문에 중첩 for 문을 사용
        for(j = 0; j < n - 1; j++){ //값 연산자(*)
            if(*(array + j) > *(array + j + 1)){
                temp = *(array + j);
                *(array + j) = *(array + j + 1);
                *(array + j + 1) = temp;
            }
        }
    }
    for(k = 0; k < n - 1; k++)
        printf("%d, ", *(array + k));
    printf("%d\n", *(array+6));

    /* 조건문을 활용하는 방법
     for(k = 0; ; k++){
         printf("%d", *(array + k));
         if(k == n - 2)     break; //반복문을 탈출, for 문에 조건문 삭제
         printf(", ");
     }
     printf("\n");
     */
}
```