✓ **6일차 내용 복습**

- malloc() : 데이터를 동적으로 메모리의 heap 영역에 할당함. 얼만큼의 data가 들어오는지 미리 알 수 없는 경우에 사용함.

- malloc() 함수 사용법

int *str_ptr = (char *)malloc(sizeof(char) * 20) // char형 크기 20개 만큼의 메모리를 할당하고 char형의 주소를 반환한다.

- free() : 메모리의 heap 영역에 할당된 data를 해제함. malloc을 하고 free를 계속적으로 하지 않으면 메모리가 부족해지는 경우가 발생함.

- memmove() : 메모리의 값을 복사할 때 사용함.

- memmove(목적지,원본,길이) 로 사용하며 원본의 데이터가 길이만큼 목적지로 복사됨.

- strcmp() : 2 개의 문자열이 서로 같은지 비교하는 함수

- 문자열 str1 과 str2 이 같다면 strcmp(str1,str2) = 0 이고, 다르다면 strcmp(str1,str2) = 1 이다.

- strncmp() : 2 개의 문자열을 일정 길이만큼 비교하는 함수.

- strncmp(str1,str2,8) 은 문자열 str1, str2 를 8 바이트 만큼 비교함.

- 구조체 : 문자열, 문자, 숫자등을 하나로 묶어서 새로운 자료형을 만들고자 할 때 사용함. (커스텀 데이터 타입)

- typedef : 자료형에 새로운 이름을 부여하고자 할 떄 사용함.(주로 구조체나 함수 포인터에 사용함.)

- ex)

1. typedef int * PINT;

2. typedef int INT[5];

3. typedef struct __id_card{char name[NAME_LEN]; char id[ID_LEN]; unsigned int age; }id_card;

   char name[NAME_LEN], char id[ID_LEN], unsigned int age 의 변수를 가지는 구조체 id_card 라는 자료형을 선언함. 구조체 변수를 선언할 때는 id_card arr[2]로 선언함.(id_card 형의 구조체를 2 개 갖는 배열) 구조체 안의 변수에 접근할 때는 arr[i].name, arr[i].id, arr[i].age 의 형식으로 접근함.

- 구조체 안의 구조체

- ex) : typedef struct __id_card{ char name[30]; char id[15]; unsigned int age; } id_card;

typedef struct __city{ id_card card; char city[30]; } city;

city 형의 구조체 안에 id_card 형의 구조체 card 가 들어가 있다.

- 함수 포인터의 기본기

- 함수 프로토타입 : 리턴, 함수명, 인자에 대한 기술서이다.

- ex)

1. int (*p)[2] -> int (*)[2] p -> int 형 2 개에 대한 포인터 p

2. void (* signal(int signum, void(* handler)(int)))(int) -> void (*)(int) signal(int signum, void(* handler)(int)) : void 형을 리턴하고 int 형을 인자로 갖는 함수에 대한 포인터를 리턴하고, int 형 signum 과 void 를 리턴하고 int 형을 인자로 갖는 함수에 대한 포인터 handler 를 인자로 갖는 함수 signal

1) 리턴 : void (*)(int) : void 형을 리턴하고 int 형을 인자로 갖는 함수에 대한 포인터

2) 함수명 : signal

3) 인자 : int signum, void(* handler)(int)

✓ **7일차 내용 복습**

■ **함수 포인터**

- 함수 포인터를 사용하는 이유 : 비동기 처리를 위해서 사용한다. 비동기 처리란 언제 어떤 이벤트가 발생할지 모르고, 서로 간에 합의가 되어있지 않은 상황에서 예기치 않게 일어나는 일을 처리하는 것을 말한다. 인터럽트, 시스템 콜이 비동기 처리에 해당된다. 갑작스럽게 이벤트가 발생했을 때 함수 포인터를 이용해서 해당 이벤트에 맞게 대응하도록 한다.

- 함수 포인터 해석 방법

* 함수 포인터를 해석할 때는 맨 앞에 있는 포인터부터 괄호로 묶고 맨 뒤에 있는 인자를 맨 앞의 포인터 뒤로 옮긴다. 그 다음 포인터를 괄호로 묶고 맨 뒤에 있는 인자를 방금 괄호로 묶은 포인터 뒤로 옮긴다. 이 작업을 반복한다.

1) int (*p) (char) : int형을 반환하고, char형을 인자로 갖는 함수에 대한 포인터

2) void bbb(void(*p)(void))

void를 반환하고, void를 반환하고 void를 인자로 갖는 함수에 대한 포인터를 인자로 갖는 함수

3) void (*bbb(void))(void)

➔ void (*)(void) bbb(void)

➔ void를 반환하고 void를 인자로 갖는 함수의 포인터를 반환하고, void를 인자로 갖는 함수

4) int (*(*bbb(void))(void))[2]

➔ int (*)[2](*bbb(void))(void)

➔ int (*)[2] (*)(void) bbb(void)

➔ int형 2개 묶음의 배열의 포인터를 반환하고 void를 인자로 갖는 함수의 포인터를 반환하고, void를 인자로 갖는 함수

5) void (*bbb(void(*p)(void)))(void)

➔ void (*)(void) bbb(void(*p)(void))

➔ void를 반환하고 void를 인자로 갖는 함수의 포인터를 반환하고, void를 반환하고 void를 인자로 갖는 함수의 포인터를 인자로 갖는 함수

- 함수 포인터 작성 방법 : 함수 포인터를 해석할 때의 방법을 거꾸로 하면서 작성한다.

■ **스 택 (Stack)**

- Push를 하면 메모리의 stack 영역에 데이터가 쌓이게 되고, pop을 하면 stack영역의 가장 아래부터 데이터가 없어지게 된다.