

Xilinx Zynq FPGA, TI DSP, MCU 기반의 프로그래밍 및 회로 설계 전문가 과정

강사 - Innova Lee(이상훈)
gcccompil3r@gmail.com

학생 - hoseong Lee(이호성)
hslee00001@naver.com

목차

1. typedef 사용법
2. malloc() 과 free() 사용법
3. calloc() 사용법
4. stack 메모리와 Heap 메모리
5. 구조체(커스텀 데이터 타입)
6. 구조체 포인터
7. 구조체 참조 연산자.
8. enum(열거형)의 유용성
9. 구조체 배열
10. 함수 포인터

1. typedef 는 무엇인가?

자료형에 새로운 이름을 부여해서 간편하게 사용하고자 할 때 사용 (주로 구조체나 함수 포인터)

ex1)

```
#include <stdio.h>

typedef int INT;
typedef int *PINT;

int main(void){
    INT num = 3;
    PINT ptr = &num;

    printf("num = %d\n", *ptr);
    return 0;
}
```

run:

A terminal window showing the output of the first example program. The text 'num = 3' is displayed in a monospaced font on a dark background.

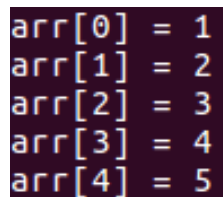
ex2)

```
#include <stdio.h>

typedef int INT[5];
int main(void){
    int i;
    INT arr = {1, 2, 3, 4, 5};
    for(i = 0; i < 5; i++)
        printf("arr[%d] = %d\n", i, arr[i]);

    return 0;
}
```

run:

A terminal window showing the output of the second example program. The text displays the contents of an array: 'arr[0] = 1', 'arr[1] = 2', 'arr[2] = 3', 'arr[3] = 4', and 'arr[4] = 5', each on a new line.

→ typedef 를 붙여주고 구조체를 정의한다. 구조체 별칭도 보인다.

2. malloc()은 무엇을 하는가?

메모리를 사용하려면 malloc 함수로 사용할 메모리 공간을 확보해야 합니다(memory allocation). 이때 필요한 메모리 크기는 바이트 단위로 지정합니다

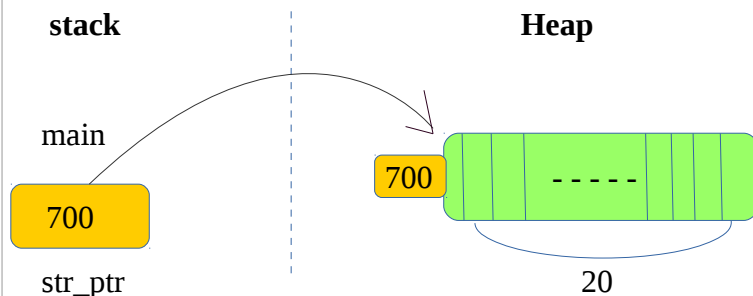
Memory 구조상 heap 에 data 를 할당함. data 가 계속해서 들어올 경우, 얼마만큼의 data 가 들어오는지 알 수 없음. 들어 올 때마다 동적으로 할당할 필요성이 있음.

포인터변수 = malloc(크기);

→ void *malloc(size_t_size); , 성공하면 메모리 주소를 반환, 실패하면 NULL 을 반환

→ 게임에서: 접속자 수가 몇명일지 모른다. 이 때, 접속자가 많아졌을 경우 서버가 폭주할 수 가 있으므로 이때 사용하는 malloc 이다.

ex) alloc() :



Heap 에서 할당한 메모리는 반드시 해제 해야한다.

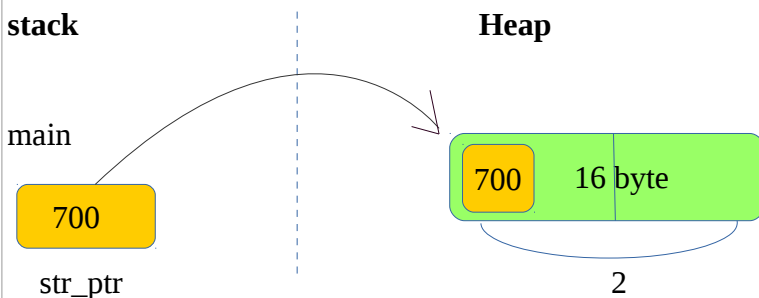
→ free(포인터변수)

3. calloc()은 무엇을 하는가?

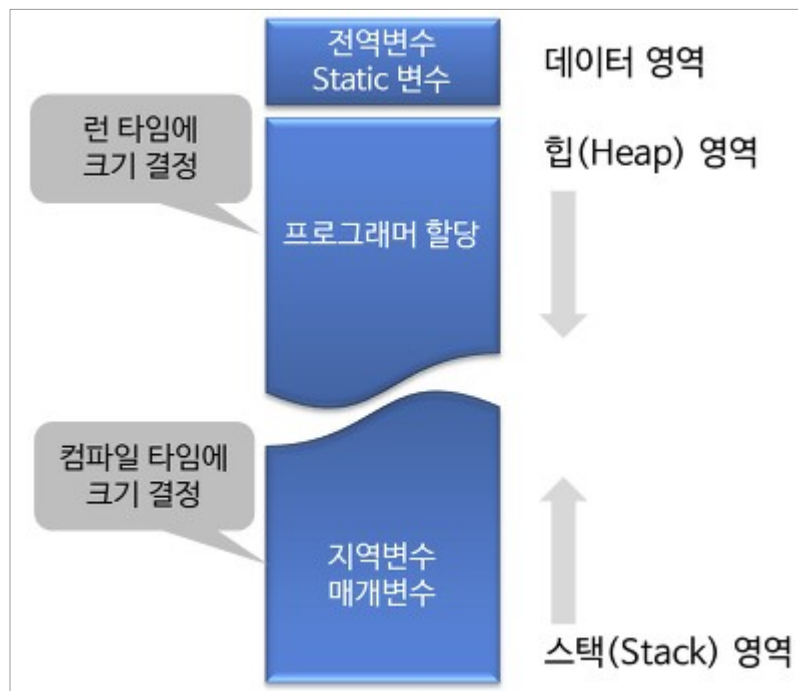
malloc()과 완전히 동일함. 허나 사용 방법이 다르다. 1 번째 인자는 할당할 개수, 2 번째 인자는 할당할 크기 즉, calloc(2,sizeof(int)) 는 8byte 공간을 할당

calloc(할당개수,할당할 크기)

ex) calloc(2,sizeof(int))



4. Stack 메모리와 Heap 메모리



: 운영체제는 우리가 실행시킨 프로그램을 위해 메모리 공간을 할당해주는데 할당되는 메모리공간은 크게 스택(Stack), 힙(Heap), 데이터(Data)영역으로 나뉘어진다.

4.1 데이터

- 전역변수와 **static 변수**가 할당되는 영역
- 프로그램의 시작과 동시에 할당되고, 프로그램이 종료되어야 메모리에서 소멸됨.

4.2 Stack

- 함수 호출시 생성되는 **지역변수**와 **매개변수**가 저장되는 영역
- 함수 호출이 완료되면 사라짐.

4.3 Heap

- 프로그래머가 할당한다.

```
ex) // 비 정상적인 배열선언
int i = 0;
scanf("%d", &i);
int arr[i];
```

→ i 크기가 4byte 라는 것은 알 수 있으나, arr 이라는 배열의 크기는 알 수 없다.

- 할당해야 할 메모리의 크기를 프로그램이 실행되는 동안 결정해야 하는 경우(런 타임때) 유용하게 사용되는 공간

5. 구조체는 왜 사용하는가?

자료를 처리하다보니 하나로 묶어야 편함
문자열과, 숫자를 한 번에 묶어서 관리하고 싶을때 등

name, age, address 변수에는 한 사람의 정보만 저장할 수 있다. 여러명의 정보를 저장하려면 name1, name2 처럼 변수이름을 바꿔서 계속 추가해야한다. 복잡하고 비효율적이기 때문에 자료를 체계적으로 관리하기 위해 구조체를 사용한다.

5.1 구조체 사용하기

구조체 정의

```
struct Person {           // 구조체 정의
    char name[20];         // 구조체 멤버 1
    int age;               // 구조체 멤버 2
    char address[100];     // 구조체 멤버 3
};
```

→ main 문에 사용할 구조체 변수 선언과 값 할당

```
struct Person p1 = { "홍길동", 30, "서울시 어쩌고"};
```

출력

```
printf("이름: %s\n", p1.name);    // 이름: 홍길동
printf("나이: %d\n", p1.age);    // 나이: 30
printf("주소: %s\n", p1.address); // 주소: 서울시 용산구 한남동
```

5.2 구조체 사용하기

```
struct Person {           // 구조체 정의
    char name[20];         // 구조체 멤버 1
    int age;               // 구조체 멤버 2
    char address[100];     // 구조체 멤버 3
} p1;
```

6. 구조체 배열과 포인터

6.1 구조체 포인터

구조체는 멤버 변수가 여러개 들어 있어서 크기가 큰 편

구조체 변수끼리 할당하면 모든 멤버를 복사하게됨.

구조체 변수를 일일이 선언해서 사용하는 것은 비효율적이다.

다른 자료형과 마찬가지로 구조체도 포인터를 선언할 수 있으며, 구조체 포인터에는 (malloc) 함수를 사용하여 동적 메모리를 할당할 수 있다.

```
struct 구조체이름 *포인터이름 = malloc(sizeof(struct 구조체이름));
```

```
struct Person {    // 구조체 정의
    char name[20];    // 구조체 멤버 1
    int age;          // 구조체 멤버 2
    char address[100]; // 구조체 멤버 3
};

int main()
{
    struct Person *p1 = malloc(sizeof(struct Person));    // 구조체 포인터 선언, 메모리 할당

    // 화살표 연산자로 구조체 멤버에 접근하여 값 할당
    strcpy(p1->name, "홍길동");
    p1->age = 30;
    strcpy(p1->address, "서울시 용산구 한남동");

    // 화살표 연산자로 구조체 멤버에 접근하여 값 출력
    printf("이름: %s\n", p1->name);    // 홍길동
    printf("나이: %d\n", p1->age);    // 30
    printf("주소: %s\n", p1->address); // 서울시 용산구 한남동

    free(p1);    // 동적 메모리 해제
```

구조체 이름 앞에는 반드시 struct 키워드를 붙여야 한다는 점만 기억하면 쉽다. 즉, 포인터를 선언할 때도, sizeof로 크기를 구할 때도 struct 키워드를 넣어준다.

7. 구조체 참조 연산자.

7.1 화살표 연산자

```
// 화살표 연산자로 구조체 멤버에 접근하여 값 할당
strcpy(p1->name, "홍길동");
p1->age = 30;
strcpy(p1->address, "서울시 용산구 한남동");

// 화살표 연산자로 구조체 멤버에 접근하여 값 출력
printf("이름: %s\n", p1->name);      // 홍길동
printf("나이: %d\n", p1->age);      // 30
printf("주소: %s\n", p1->address);  // 서울시 용산구 한남동
```

: 구조체의 멤버에 접근하는 방법

- $p1 \rightarrow age = 30$: 구조체 포인터의 멤버에 접근한 뒤 값을 할당한 뒤
- $p1 \rightarrow age$: 값을 가져온다.

문자열 멤버는 할당연산자로 문자열을 저장할 수 없으므로 strcpy 함수를 사용한다.
마지막으로 free(p1) 으로 할당한 메모리를 해제해준다.

7.2 괄호와 역참조를 사용하면 (점)으로

```
p1->age;      // 화살표 연산자로 멤버에 접근
(*p1).age;    // 구조체 포인터를 역참조한 뒤 .으로 멤버에 접근
```

: 구조체 포인터에서 .으로 멤버에 접근하기

→ (*p1).age 와 같이 구조체 포인터를 역참조하면 pointer to struct Person 에서 pointer to 가 제거되어 struct Person 이 된다. 따라서 .으로 접근 가능하다.

8. enum(열거형)의 유용성

열거형을 사용하면 정수형 상수를 좀 더 편하게 정의

```
enum DayOfWeek {    // 열거형 정의
    Sunday = 0,      // 초깃값 할당
    Monday,
    Tuesday,
    Wednesday,
    Thursday,
    Friday,
    Saturday
};

int main()
{
    enum DayOfWeek week;    // 열거형 변수 선언

    week = Tuesday;        // 열거형 값 할당

    printf("%d\n", week);   // 2: Tuesday의 값 출력

    return 0;
}
```

9. 구조체 배열

8.1 구조체 배열 선언하는 방법 : struct 구조체이름 변수이름[크기]

```
struct Point2D {
    int x;
    int y;
};

int main()
{
    struct Point2D p[3];    // 크기가 3인 구조체 배열 생성

    p[0].x = 10;    // 인덱스로 요소에 접근한 뒤 점으로 멤버에 접근
    p[0].y = 20;
    p[1].x = 30;
    p[1].y = 40;
    p[2].x = 50;
    p[2].y = 60;

    printf("%d %d\n", p[0].x, p[0].y);    // 10 20
    printf("%d %d\n", p[1].x, p[1].y);    // 30 40
    printf("%d %d\n", p[2].x, p[2].y);    // 50 60
}
```

→ 구조체 배열에서 각 요소에 접근하려면 배열 뒤에 대괄호를 사용하며 대괄호 안에 인덱스를 지정해주면 됩니다. 이 상태에서 다시 멤버에 접근하려면 (점)을 사용합니다. 즉, p[0].x 는 구조체 배열의 첫 번째 요소에서 멤버 x에 접근한다는 뜻입니다.

8.2 구조체 배열 선언하는 동시에 초기화하기.

```
struct Point2D {
    int x;
    int y;
};

int main()
{
    // 구조체 배열을 선언하면서 초기화
    struct Point2D p1[3] = { { .x = 10, .y = 20 }, { .x = 30, .y = 40 }, { .x = 50, .y = 60 } };

    printf("%d %d\n", p1[0].x, p1[0].y);    // 10 20
    printf("%d %d\n", p1[1].x, p1[1].y);    // 30 40
    printf("%d %d\n", p1[2].x, p1[2].y);    // 50 60

    // 구조체 배열을 선언하면서 초기화
    struct Point2D p2[3] = { { 10, 20 }, { 30, 40 }, { 50, 60 } };

    printf("%d %d\n", p2[0].x, p2[0].y);    // 10 20
    printf("%d %d\n", p2[1].x, p2[1].y);    // 30 40
    printf("%d %d\n", p2[2].x, p2[2].y);    // 50 60
}
```

10. 구조체 포인터 배열

구조체 요소가 한꺼번에 뭉쳐져 있는 배열이 아닌 요소마다 메모리를 할당하고 싶을 수도 있다. 이때는 구조체 포인터 배열을 만들고, malloc 함수로 각 요소에 메모리를 할당하면 된다.

선언하는 방법: struct 구조체이름 *포인터이름[크기];

```
struct Point2D {
    int x;
    int y;
};

int main()
{
    struct Point2D *p[3];    // 크기가 3인 구조체 포인터 배열 선언

    // 구조체 포인터 배열 전체 크기에서 요소(구조체 포인터)의 크기로 나눠서 요소 개수를 구함
    for (int i = 0; i < sizeof(p) / sizeof(struct Point2D *); i++)    // 요소 개수만큼 반복
    {
        p[i] = malloc(sizeof(struct Point2D));    // 각 요소에 구조체 크기만큼 메모리 할당
    }

    p[0]->x = 10;    // 인덱스로 요소에 접근한 뒤 화살표 연산자로 멤버에 접근
    p[0]->y = 20;
    p[1]->x = 30;
    p[1]->y = 40;
    p[2]->x = 50;
    p[2]->y = 60;

    printf("%d %d\n", p[0]->x, p[0]->y);    // 10 20
    printf("%d %d\n", p[1]->x, p[1]->y);    // 30 40
    printf("%d %d\n", p[2]->x, p[2]->y);    // 50 60

    for (int i = 0; i < sizeof(p) / sizeof(struct Point2D *); i++)    // 요소 개수만큼 반복
    {
        free(p[i]);    // 각 요소의 동적 메모리 해제
    }
}
```

→ 구조체 포인터를 선언하고, 배열도 선언했다. 메모리 할당을 해주어야하므로 배열 크기(요소개수)만큼 반복하면서 각 요소에 구조체 크기만큼 메모리를 할당해준다. 이 때 구조체포인터 배열에는 포인터가 들어 있으므로 요소 개수를 구하려면 구조체 포인터 배열의 전체 크기에서 구조체 포인터의 크기로 나눠주면된다.

*** sizeof(struct Point2D)**는 구조체가 차지하는 크기, **sizeof(struct Point2D *)**는 구조체 포인터의 크기

11. 함수 포인터

```
void(* signal(int signum, void (* handler)(int)))(int);
```

*함수 프로토타입이란 ?

리턴, 함수명, 인자에 대한 기술서

그렇다면 위 함수에 대한 프로토타입은 뭘까?

이전에 배웠던 `int (*p)[2];` → `int (*)[2] p`

리턴: `void (*)`(int)

함수명: `signal`

인자 : `int signum` 과 `void (*handle)(int)`

```
void (*p)(void)
```

`void` 를 리턴하고 `void` 를 인자로 취하는 함수의 주소값을 저장할 수 있는 변수 `p`

```
int aaa(int,int);
```

`int (*p)(int,int)` → 이것이 함수 프로토타입

* 함수 포인터를 쓰는 이유는 ??

- a. 비동기 처리
- b. HW 개발 관점에서 인터럽트
- c. 시스템콜 (유일한 SW 인터럽트임)

여기서 인터럽트들 (SW,HW) 은 사실상 모두 비동기 동작에 해당한다 . 결국 1 번 (비동기 처리) 가 핵심이라는 의미다 .

* 그렇다면 비동기 처리라는 것은 무엇일까?

기본적으로 동기 처리라는 것은 송신하는쪽 수신하는쪽이 쌍방 합의하에만 달성된다 . (휴대폰 전화 통화 등등)

반면 비동기 처리는 (이메일 , 카톡등의 메신저)

그래서 그냥 던져 놓으면 상대방이 바쁠때는 못보겠지만 그다지 바쁘지 않은 상황이라면 메시지를 보고 답변을 줄 것이다 . 이와 같이 언제 어떤 이벤트가 발생할지 알 수 없는 것들을 다루는 녀석이 바로 함수 포인터다 .

사람이 이런데서는 임기응변을 잘해야 하듯이 컴퓨터 관점에서 임기응변을 잘 하도록 만들어주는 것이 바로 함수 포인터다 . 또한 `c` 를 자바처럼 쓸 수있음

선언하는 방법 : 반환값자료형 (*함수포인터이름)();

```
void hi()
{
    printf~~
}
```

```
void (*fp)();
fp = hi;
fp();
```

→ 반환값과 매개변수가 없는 함수 포인터 fp 선언

```
int main(void){
bbb();
ccc(aaa);
ddd();
return 0;
```

```
void (* bbb(void))(void)
```

→ 리턴이 함수 포인터임.

리턴 : void (*)(void)

이름 : bbb

인자: void

```
void ccc(void(*p)(void)){
```

→ 인자가 함수 포인터임

리턴: void

이름: ccc

인자: void(*p)(void)

```
int (* ddd(void))(void)
```

→ 리턴이 함수 포인터임.

리턴: int (*)(void)

이름: ddd

인자: void

int *p[] = 가로크기가 []인 배열을 가리키는 포인터 즉 {a,b,c,d..}를 가리키는 포인터(=포인터 배열)

int (*p)[] = int 형 포인터 []개를 담을수 있는 배열, (=배열포인터)

```
void (* bbb(void(*p)(void)))(void)
```

→ 리턴, 인자 모두 함수 포인터!

리턴: void (*)(void)

이름: bbb

인자: void(*p)(void)

memmove()

memcpy → 속도가 빠르므로 이게 더 좋긴한데 메모리 보호가 안된다. 먼저 사용해보고 오류가나면 무브로 바꾸면된다.

String Transfer to main()

strlen 함수언제 사용하나?

주로 strcpy()등의 함수와 함께 사용하는편

문자열의 길이를 구하는데 사용함

strlen(“문자열”);처럼 사용함

strcpy() 함수는 언제 사용하나?

문자열을 복사하고 싶을 경우 사용한다.

strcpy(dst,src), strncpy(dst,src,length) 로 사용.

→ strncpy 는 몇개를 복사할 것인지 추가 인자를 넣는다.

Strncmp(), strncmp() 함수는 언제 사용하나?

문자열이 서로 같은지 비교하고 싶을때, 서로같은 경우 0 을 반환하게 된다. 중요..

strcmp(str1,str2), strncmp(str1,str2,len) 처럼

→ strncmp 는 마찬가지로 몇개가 같은지 추가 인자를 넣는다.

int (* aaa(void))[2]

→ int (*)[2] aaa(void)

배열 2 개짜리 묶음의 주소를 반환하고 인자로 void 를 취하는 함수 aaa

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

→ int (*)[2](*) (void) bbb(void)

배열 2 개짜리 묶음의 주소를 반환하고

인자로 void 를 취하는 함수 포인터를 반환하며

인자로 void 를 취하는 함수 bbb

int ((*(*p[][2])(void))(void))[2]

int (*)[2] (*) (void) (*) (void) p[][2]

배열 2 개짜리 묶음의 주소를 반환하고,

인자로 void 를 취하는 함수 포인터를 반환하며,

다시 인자도 void 를 취하는 함수 포인터를 배열형태로 가짐.

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

```
#define EMPTY 0
```

```
struct node{
    int data;
    struct node *link;
};
typedef struct node Stack;
```

```
Stack *get_node()
{
    Stack *tmp;
    tmp=(Stack *)malloc(sizeof(Stack));
    tmp->link=EMPTY;
    return tmp;
}
```

```
void push(Stack **top, int data)
{
    Stack *tmp;
    tmp = *top;
    *top = get_node();
    (*top)->data = data;
    (*top)->link = tmp;
}
```

Stack

```
int pop(Stack **top)
{
```

```
    Stack *tmp;
    int num;
    tmp = *top;
    if(*top == EMPTY)
    {
        printf("Stack is empty!!!\n");
        return 0;
    }
    num = tmp->data;
    *top = (*top)->link;
    free(tmp);
    return num;
}
```

```
int main(void)
{
```

```
    Stack *top = EMPTY;
    push(&top, 10);
    push(&top, 20);
    push(&top, 30);
    printf("%d\n", pop(&top));
    printf("%d\n", pop(&top));
    printf("%d\n", pop(&top));
    printf("%d\n", pop(&top));
    return 0;
}
```

