EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv1 과정

제 4기

2022. 12. 29

진동민

학습목표 & 6회차 날짜

EDDI
Electronic Design Development Institute

학습목표

- 구조체가 무엇인지 명확히 이해하기
- 구조체를 어셈블리어로 분석하기
- 함수의 인자로 구조체 변수의 포인터와 구조체 변수를 넘겼을 때의 차이점 이해하기
- 함수 포인터를 이용한 인터페이스 이해하기

수업 날짜

2022-10-01 (토) 오후 6시~9시

목차

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

1) struct.c

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

typedef struct _member
{
    char *name;
    int age;
} member;

#define MEMBER_ALLOC_FAIL       false
#define MEMBER_ALLOC_SUCCESS    true

bool init_member(member *account, char *name, int age)
{
    int name_length = strlen(name);
    char *tmp = (char *)malloc(sizeof(char) * (name_length + 1));

    if (tmp == NULL)
    {
        return MEMBER_ALLOC_FAIL;
    }

    account->name = tmp;
    strncpy(account->name, name, name_length);
    account->age = age;

    return MEMBER_ALLOC_SUCCESS;
}
```

```c
void print_member(member account)
{
    printf("이름: %s\n", account.name);
    printf("나이: %d\n", account.age);
}

// 과제
void free_member(member account)
{
    free(account.name);
}

int main(void)
{
    int value;
    struct _member account;

    member my_account = { };

    if (init_member(&my_account, "gogo", 19) == MEMBER_ALLOC_FAIL)
    {
        printf("member 구조체 할당 실패!\n");
        exit(-1);
    }

    print_member(my_account);

    free_member(my_account);
    return 0;
}
```

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

2) 구조체에서 가장 중요한 개념

1. 자료형을 새로 정의한다 -> 커스텀 데이터 타입
2. 재활용 측면으로 사용

한 가지를 더 생각하면
3. 행위의 일관성 측면의 클래스 설계 -> SRP 규칙

SRP 규칙을 오해하면 '하나의 기능만 하게 만들어라'라고 오해할 수 있지만, SRP 규칙은 무엇이냐?!

SRP 규칙: 특정 동작을 수행함에 있어 일관성을 가지게 만들어라!
이 규칙의 핵심: 언제나 일관성 있는 설계를 해서 신뢰성을 주어라!

사실 구조체를 구성할 때는 이런 개념을 가지고 설계하면 좋다

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

3) 구조체

구조체는 데이터 타입을 만드는 작업이다.

```
typedef struct _member
{
    char *name;
    int age;
} member;
```


4) 구조체 변수를 초기화하는 방법

보통은 다음과 같은 방식으로 한다.

```
if (init_member(&my_account, "gogo", 19) == MEMBER_ALLOC_FAIL)
{
    printf("member 구조체 할당 실패!\n");
    exit(-1);
}
```

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

5) bool

bool 타입을 사용하려면 stdbool.h 헤더 파일을 포함해야 한다.

6) malloc이 fail이 발생할 수도 있다

이 예제에서 예외처리는 아래와 같다.

```c
#include <stdbool.h>
#include <stdlib.h>

#define MEMBER_ALLOC_FAIL       false

char *tmp = (char *)malloc(sizeof(char) * (name_length + 1));

if (tmp == NULL)
{
    return MEMBER_ALLOC_FAIL;
}
```

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

7) exit 함수

exit(-1): -1이 나오는 건 보통 굉장한 문제가 생겨서 터진 케이스
malloc이 동작을 안한다는 건 메모리에 심각한 문제가 있는 것이다.

8) 할당하는 함수의 인자로 &가 붙는다면…

보편적으로 잘 설계된 오픈소스 코드들을 보면 어떤 할당하는 함수 코드에 이상하게 &가 붙는 애들이 있을 것이다.
그런 &를 보면, 아! 이 함수에 들어가서 여기에 값을 세팅하려고 하는구나를 의심해봐야 한다.
보편적인 목적이 그렇다.

9) exit 함수와 return의 차이

개발자가 return -1;을 보면 그냥 별생각 없이 -1을 반환하는 구나라고 생각하겠지만,
exit(-1)은 아 프로그램을 종료하는구나라고 이런 차이가 있다.

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

10) struct.c의 main 함수 어셈블리어

```
(gdb) disas
Dump of assembler code for function main:
=> 0x0000555555555323 <+0>:     endbr64
   0x0000555555555327 <+4>:     push   %rbp
   0x0000555555555328 <+5>:     mov    %rsp,%rbp
   0x000055555555532b <+8>:     sub    $0x20,%rsp
   0x000055555555532f <+12>:    mov    %fs:0x28,%rax
   0x0000555555555338 <+21>:    mov    %rax,-0x8(%rbp)
   0x000055555555533c <+25>:    xor    %eax,%eax
   0x000055555555533e <+27>:    movq   $0x0,-0x20(%rbp)
   0x0000555555555346 <+35>:    movq   $0x0,-0x18(%rbp)
   0x000055555555534e <+43>:    lea    -0x20(%rbp),%rax
   0x0000555555555352 <+47>:    mov    $0x13,%edx
   0x0000555555555357 <+52>:    lea    0xcc2(%rip),%rsi        # 0x555555556020
   0x000055555555535e <+59>:    mov    %rax,%rdi
   0x0000555555555361 <+62>:    callq  0x555555555229 <init_member>
   0x0000555555555366 <+67>:    xor    $0x1,%eax
   0x0000555555555369 <+70>:    test   %al,%al
   0x000055555555536b <+72>:    je     0x555555555383 <main+96>
   0x000055555555536d <+74>:    lea    0xcb4(%rip),%rdi        # 0x555555556028
   0x0000555555555374 <+81>:    callq  0x5555555550e0 <puts@plt>
   0x0000555555555379 <+86>:    mov    $0xffffffff,%edi
   0x000055555555537e <+91>:    callq  0x555555555130 <exit@plt>
   0x0000555555555383 <+96>:    mov    -0x20(%rbp),%rdx
   0x0000555555555387 <+100>:   mov    -0x18(%rbp),%rax
   0x000055555555538b <+104>:   mov    %rdx,%rdi
   0x000055555555538e <+107>:   mov    %rax,%rsi
   0x0000555555555391 <+110>:   callq  0x5555555552a9 <print_member>
   0x0000555555555396 <+115>:   mov    -0x20(%rbp),%rdx
   0x000055555555539a <+119>:   mov    -0x18(%rbp),%rax
   0x000055555555539e <+123>:   mov    %rdx,%rdi
   0x00005555555553a1 <+126>:   mov    %rax,%rsi
   0x00005555555553a4 <+129>:   callq  0x5555555552f7 <free_member>
   0x00005555555553a9 <+134>:   mov    $0x0,%eax
   0x00005555555553ae <+139>:   mov    -0x8(%rbp),%rcx
   0x00005555555553b2 <+143>:   xor    %fs:0x28,%rcx
   0x00005555555553bb <+152>:   je     0x5555555553c2 <main+159>
   0x00005555555553bd <+154>:   callq  0x555555555100 <__stack_chk_fail@plt>
   0x00005555555553c2 <+159>:   leaveq
   0x00005555555553c3 <+160>:   retq
End of assembler dump.
```

구조체를 어셈블리어로 분석하기

11) struct.c의 init_member 함수 호출전까지의 main 함수 어셈블리어 분석

```
(gdb) disas
Dump of assembler code for function main:
=> 0x0000555555555323 <+0>:     endbr64
   0x0000555555555327 <+4>:     push   %rbp
   0x0000555555555328 <+5>:     mov    %rsp,%rbp
   0x000055555555532b <+8>:     sub    $0x20,%rsp
   0x000055555555532f <+12>:    mov    %fs:0x28,%rax
   0x0000555555555338 <+21>:    mov    %rax,-0x8(%rbp)
   0x000055555555533c <+25>:    xor    %eax,%eax
   0x000055555555533e <+27>:    movq   $0x0,-0x20(%rbp)
   0x0000555555555346 <+35>:    movq   $0x0,-0x18(%rbp)
   0x000055555555534e <+43>:    lea    -0x20(%rbp),%rax
   0x0000555555555352 <+47>:    mov    $0x13,%edx
   0x0000555555555357 <+52>:    lea    0xcc2(%rip),%rsi        # 0x555555556020
   0x000055555555535e <+59>:    mov    %rax,%rdi
   0x0000555555555361 <+62>:    callq  0x555555555229 <init_member>
```

32바이트 스택 생성

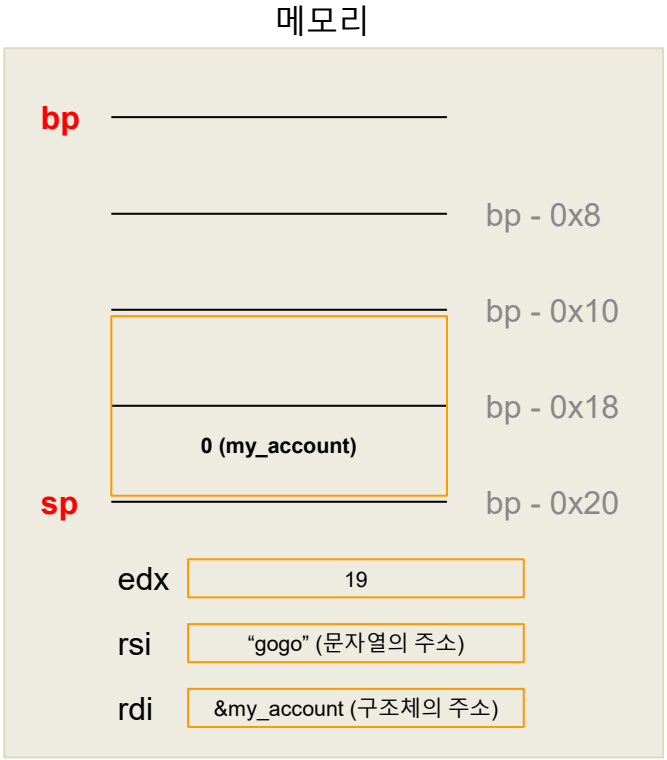
member my_account = { };

init_member(&my_account, “gogo”, 19)

main 함수 스택에 있는 my_account 구조체 변수의 주소값을 init_member 함수의 첫 번째 매개변수로 넘기기 위해 lea 명령어를 사용하여 rdi 레지스터에 복사했다.

구조체를 어셈블리어로 분석하기

11) struct.c의 init_member 함수 호출전까지의 main 함수 메모리



설명

- gdb 프로그램 내에서 아래 명령어를 실행하여 구조체 멤버의 주소를 알아낼 수 있다.
  - p &(my_account.name)
  - p &(my_account.age)

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

12) struct.c의 init_member 함수 어셈블리어

```
(gdb) disas
Dump of assembler code for function init_member:
=> 0x0000555555555229 <+0>:     endbr64
   0x000055555555522d <+4>:     push   %rbp
   0x000055555555522e <+5>:     mov    %rsp,%rbp
   0x0000555555555231 <+8>:     sub    $0x30,%rsp
   0x0000555555555235 <+12>:    mov    %rdi,-0x18(%rbp)
   0x0000555555555239 <+16>:    mov    %rsi,-0x20(%rbp)
   0x000055555555523d <+20>:    mov    %edx,-0x24(%rbp)
   0x0000555555555240 <+23>:    mov    -0x20(%rbp),%rax
   0x0000555555555244 <+27>:    mov    %rax,%rdi
   0x0000555555555247 <+30>:    callq  0x5555555550f0 <strlen@plt>
   0x000055555555524c <+35>:    mov    %eax,-0xc(%rbp)
   0x000055555555524f <+38>:    mov    -0xc(%rbp),%eax
   0x0000555555555252 <+41>:    add    $0x1,%eax
   0x0000555555555255 <+44>:    cltq
   0x0000555555555257 <+46>:    mov    %rax,%rdi
   0x000055555555525a <+49>:    callq  0x555555555120 <malloc@plt>
   0x000055555555525f <+54>:    mov    %rax,-0x8(%rbp)
   0x0000555555555263 <+58>:    cmpq   $0x0,-0x8(%rbp)
   0x0000555555555268 <+63>:    jne    0x555555555271 <init_member+72>
   0x000055555555526a <+65>:    mov    $0x0,%eax
   0x000055555555526f <+70>:    jmp    0x5555555552a7 <init_member+126>
   0x0000555555555271 <+72>:    mov    -0x18(%rbp),%rax
   0x0000555555555275 <+76>:    mov    -0x8(%rbp),%rdx
   0x0000555555555279 <+80>:    mov    %rdx,(%rax)
   0x000055555555527c <+83>:    mov    -0xc(%rbp),%eax
   0x000055555555527f <+86>:    movslq %eax,%rdx
   0x0000555555555282 <+89>:    mov    -0x18(%rbp),%rax
   0x0000555555555286 <+93>:    mov    (%rax),%rax
   0x0000555555555289 <+96>:    mov    -0x20(%rbp),%rcx
   0x000055555555528d <+100>:   mov    %rcx,%rsi
   0x0000555555555290 <+103>:   mov    %rax,%rdi
   0x0000555555555293 <+106>:   callq  0x5555555550d0 <strncpy@plt>
   0x0000555555555298 <+111>:   mov    -0x18(%rbp),%rax
   0x000055555555529c <+115>:   mov    -0x24(%rbp),%edx
   0x000055555555529f <+118>:   mov    %edx,0x8(%rax)
   0x00005555555552a2 <+121>:   mov    $0x1,%eax
   0x00005555555552a7 <+126>:   leaveq
   0x00005555555552a8 <+127>:   retq
End of assembler dump.
```

구조체를 어셈블리어로 분석하기

13) struct.c의 init_member 함수 어셈블리어 분석 - 1

```
(gdb) disas
Dump of assembler code for function init_member:
=> 0x0000555555555229 <+0>:     endbr64
   0x000055555555522d <+4>:     push   %rbp
   0x000055555555522e <+5>:     mov    %rsp,%rbp
   0x0000555555555231 <+8>:     sub    $0x30,%rsp
   0x0000555555555235 <+12>:    mov    %rdi,-0x18(%rbp)
   0x0000555555555239 <+16>:    mov    %rsi,-0x20(%rbp)
   0x000055555555523d <+20>:    mov    %edx,-0x24(%rbp)
   0x0000555555555240 <+23>:    mov    -0x20(%rbp),%rax
   0x0000555555555244 <+27>:    mov    %rax,%rdi
   0x0000555555555247 <+30>:    callq  0x5555555550f0 <strlen@plt>
   0x000055555555524c <+35>:    mov    %eax,-0xc(%rbp)
   0x000055555555524f <+38>:    mov    -0xc(%rbp),%eax
   0x0000555555555252 <+41>:    add    $0x1,%eax
   0x0000555555555255 <+44>:    cltq
   0x0000555555555257 <+46>:    mov    %rax,%rdi
   0x000055555555525a <+49>:    callq  0x555555555120 <malloc@plt>
   0x000055555555525f <+54>:    mov    %rax,-0x8(%rbp)
   0x0000555555555263 <+58>:    cmpq   $0x0,-0x8(%rbp)
   0x0000555555555268 <+63>:    jne    0x555555555271 <init_member+72>
   0x000055555555526a <+65>:    mov    $0x0,%eax
   0x000055555555526f <+70>:    jmp    0x5555555552a7 <init_member+126>
```

48바이트 스택 생성

스택에 매개변수 배치

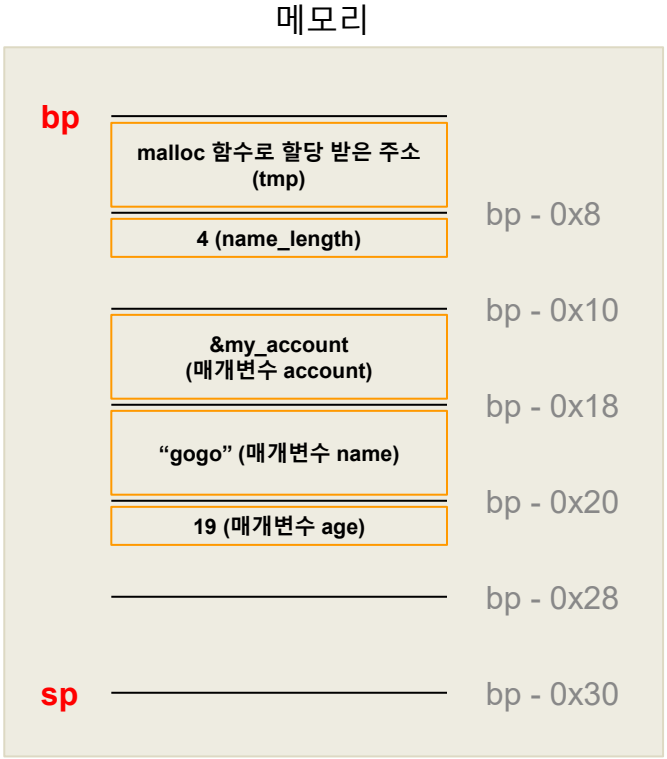
```
int name_length = strlen(name);

char *tmp = (char *)malloc(sizeof(char) *
        (name_length + 1));

if (tmp == NULL)
{
                return
MEMBER_ALLOC_FAIL;
```

구조체를 어셈블리어로 분석하기

13) struct.c의 init_member 함수 메모리



구조체를 어셈블리어로 분석하기

13) struct.c의 init_member 함수 어셈블리어 분석 - 2

```
   0x0000555555555271 <+72>:    mov    -0x18(%rbp),%rax
   0x0000555555555275 <+76>:    mov    -0x8(%rbp),%rdx
   0x0000555555555279 <+80>:    mov    %rdx,(%rax)
   0x000055555555527c <+83>:    mov    -0xc(%rbp),%eax
   0x000055555555527f <+86>:    movslq %eax,%rdx
   0x0000555555555282 <+89>:    mov    -0x18(%rbp),%rax
   0x0000555555555286 <+93>:    mov    (%rax),%rax
   0x0000555555555289 <+96>:    mov    -0x20(%rbp),%rcx
   0x000055555555528d <+100>:   mov    %rcx,%rsi
   0x0000555555555290 <+103>:   mov    %rax,%rdi
   0x0000555555555293 <+106>:   callq  0x5555555550d0 <strncpy@plt>
   0x0000555555555298 <+111>:   mov    -0x18(%rbp),%rax
   0x000055555555529c <+115>:   mov    -0x24(%rbp),%edx
   0x000055555555529f <+118>:   mov    %edx,0x8(%rax)
   0x00005555555552a2 <+121>:   mov    $0x1,%eax
   0x00005555555552a7 <+126>:   leaveq
   0x00005555555552a8 <+127>:   retq
End of assembler dump.
```

account->name = tmp;

strncpy(account->name, name, name_length);

account->age = age;

return MEMBER_ALLOC_SUCCESS;

구조체를 어셈블리어로 분석하기

14) struct.c의 init_member 함수 어셈블리어 중요 포인트 정리

account->name = tmp; 이 코드의 어셈블리어 마지막 코드는 mov %rdx, (%rax) 이다.

이 코드를 실행하기 직전까지를 분석한다면…
rax 레지스터는 매개변수 account 값을 복사했는데 이는 main 함수 스택에 배치된 my_account 구조체 변수의 주소이다.
rdx 레지스터는 매개변수 tmp 값을 복사했는데 이는 malloc 함수로 새로 할당 받은 주소이다.

그래서 마지막 코드인 mov 명령어의 destination이 (%rax)인 이유는, my_account의 첫 멤버변수가 name 이므로 별도의 주소 계산 필요없이 바로 접근한 것이다.

(별도의 주소 계산이 필요한 예시)

- account->name: 0x8(%rax)

구조체를 어셈블리어로 분석하기

15) struct.c의 init_member 함수 실행 후의 main 함수 어셈블리어 분석

```
   0x0000555555555366 <+67>:    xor    $0x1,%eax
   0x0000555555555369 <+70>:    test   %al,%al
   0x000055555555536b <+72>:    je     0x555555555383 <main+96>
   0x000055555555536d <+74>:    lea    0xcb4(%rip),%rdi        # 0x555555556028
   0x0000555555555374 <+81>:    callq  0x5555555550e0 <puts@plt>
   0x0000555555555379 <+86>:    mov    $0xffffffff,%edi
   0x000055555555537e <+91>:    callq  0x555555555130 <exit@plt>
   0x0000555555555383 <+96>:    mov    -0x20(%rbp),%rdx
   0x0000555555555387 <+100>:   mov    -0x18(%rbp),%rax
   0x000055555555538b <+104>:   mov    %rdx,%rdi  ← my_account.name
   0x000055555555538e <+107>:   mov    %rax,%rsi  ← my_account.age
   0x0000555555555391 <+110>:   callq  0x5555555552a9 <print_member>
   0x0000555555555396 <+115>:   mov    -0x20(%rbp),%rdx
   0x000055555555539a <+119>:   mov    -0x18(%rbp),%rax
   0x000055555555539e <+123>:   mov    %rdx,%rdi
   0x00005555555553a1 <+126>:   mov    %rax,%rsi
   0x00005555555553a4 <+129>:   callq  0x5555555552f7 <free_member>
   0x00005555555553a9 <+134>:   mov    $0x0,%eax
   0x00005555555553ae <+139>:   mov    -0x8(%rbp),%rcx
   0x00005555555553b2 <+143>:   xor    %fs:0x28,%rcx
   0x00005555555553bb <+152>:   je     0x5555555553c2 <main+159>
   0x00005555555553bd <+154>:   callq  0x555555555100 <__stack_chk_fail@plt>
   0x00005555555553c2 <+159>:   leaveq
   0x00005555555553c3 <+160>:   retq
End of assembler dump.
```

return 0;

```
if (... == MEMBER_ALLOC_FAIL)
{
        printf("member 구조체 할당 실패
        exit(-1);
}

print_member(my_account);

free_member(my_account);
```

구조체를 어셈블리어로 분석하기

16) struct.c의 print_member 함수를 어셈블리어로 분석하는 이유

- bool init_member(member *account, char *name, int age);
- void print_member(member account);

init_member 함수는 인자로 구조체 변수의 포인터를 넘기지만, print_member 함수는 구조체 변수를 인자로 넘긴다.

위의 방법이 실제 어셈블리어 코드에서 어떤 차이가 있는지 확인하기 위해 print_member 함수의 어셈블리어 코드를 확인해 보겠다.

앞의 슬라이드에서 C 레벨에서 print_member 함수에 넘겨준 인자는 하나로 my_account 구조체 변수를 전달했지만, 어셈블리 레벨에서는 rdi, rsi 레지스터에 값을 넘기고 함수를 호출한 것을 확인할 수 있다.

이는 C 코드 문법으로는 하나로 묶은 구조체이지만, 실제 작동하는 방식인 어셈블리 레벨에서는 구조체를 구성하는 멤버 변수를 하나씩 처리하는 것임을 알 수 있다.

이를 통해 구조체를 인자로 넘겨줄 때, 구조체를 구성하는 변수가 함수 호출시에 스택에 쌓을만큼 많다면 성능 최적화를 위해 포인터로 보내는 것도 방법 또한 생각할 수 있다.

구조체를 어셈블리어로 분석하기

17) struct.c의 print_member 함수 어셈블리어 분석

```
(gdb) disas
Dump of assembler code for function print_member:
=> 0x00005555555552a9 <+0>:     endbr64
   0x00005555555552ad <+4>:     push   %rbp
   0x00005555555552ae <+5>:     mov    %rsp,%rbp
   0x00005555555552b1 <+8>:     sub    $0x10,%rsp
   0x00005555555552b5 <+12>:    mov    %rdi,%rax
   0x00005555555552b8 <+15>:    mov    %rsi,%rcx
   0x00005555555552bb <+18>:    mov    %rcx,%rdx
   0x00005555555552be <+21>:    mov    %rax,-0x10(%rbp)
   0x00005555555552c2 <+25>:    mov    %rdx,-0x8(%rbp)
   0x00005555555552c6 <+29>:    mov    -0x10(%rbp),%rax
   0x00005555555552ca <+33>:    mov    %rax,%rsi
   0x00005555555552cd <+36>:    lea    0xd34(%rip),%rdi        # 0x555555556008
   0x00005555555552d4 <+43>:    mov    $0x0,%eax
   0x00005555555552d9 <+48>:    callq  0x555555555110 <printf@plt>
   0x00005555555552de <+53>:    mov    -0x8(%rbp),%eax
   0x00005555555552e1 <+56>:    mov    %eax,%esi
   0x00005555555552e3 <+58>:    lea    0xd2a(%rip),%rdi        # 0x555555556014
   0x00005555555552ea <+65>:    mov    $0x0,%eax
   0x00005555555552ef <+70>:    callq  0x555555555110 <printf@plt>
   0x00005555555552f4 <+75>:    nop
   0x00005555555552f5 <+76>:    leaveq
   0x00005555555552f6 <+77>:    retq
End of assembler dump.
```

16바이트 스택 생성

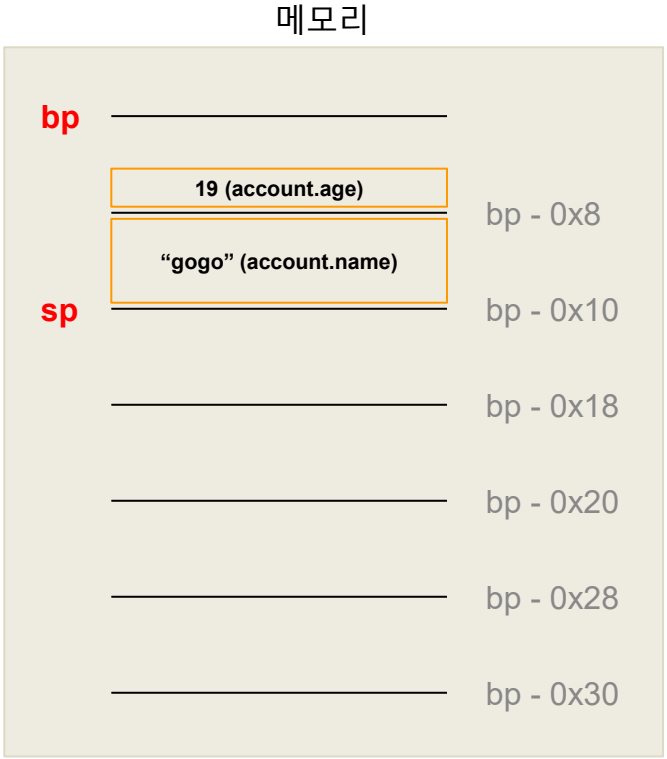
스택에 매개변수 배치

printf("이름: %s\n", account.name);

printf("나이: %d\n", account.age);

구조체를 어셈블리어로 분석하기

17) struct.c의 print_member 함수 어셈블리어 메모리



구조체를 어셈블리어로 분석하기

18) struct.c의 어셈블리어 분석 최종 정리

1. member 구조체의 크기는 16바이트로 잡히기 때문에, my_account 구조체 변수 초기화 또한 16바이트 기준으로 0으로 초기화하였다.

```
(gdb) print sizeof(account.name)
$2 = 8
(gdb) print sizeof(account.age)
$3 = 4
(gdb) print sizeof(account)
$4 = 16
```

```
(gdb) disas
Dump of assembler code for function main:
=> 0x0000555555555323 <+0>:     endbr64
   0x0000555555555327 <+4>:     push   %rbp
   0x0000555555555328 <+5>:     mov    %rsp,%rbp
   0x000055555555532b <+8>:     sub    $0x20,%rsp
   0x000055555555532f <+12>:    mov    %fs:0x28,%rax
   0x0000555555555338 <+21>:    mov    %rax,-0x8(%rbp)
   0x000055555555533c <+25>:    xor    %eax,%eax
   0x000055555555533e <+27>:    movq   $0x0,-0x20(%rbp)
   0x0000555555555346 <+35>:    movq   $0x0,-0x18(%rbp)
   0x000055555555534e <+43>:    lea    -0x20(%rbp),%rax
   0x0000555555555352 <+47>:    mov    $0x13,%edx
   0x0000555555555357 <+52>:    lea    0xcc2(%rip),%rsi        # 0x555555556020
   0x000055555555535e <+59>:    mov    %rax,%rdi
   0x0000555555555361 <+62>:    callq  0x555555555229 <init_member>
```

main 함수 어셈블리어 중 일부

member 구조체를 구성하는 멤버 변수의 총 크기가 12바이트이더라도, 구조체의 크기는 16바이트로 잡힌다.

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

18) struct.c의 어셈블리어 분석 최종 정리

2. 함수의 인자로 '구조체 변수의 포인터' vs '구조체 변수'를 넘길 때의 차이

구조체의 포인터를 함수의 인자로 넘기면 스택에 배치된 구조체의 주소를 함수의 인자로 넘기지만,
C레벨에서 구조체 변수를 함수의 인자로 넘기면, 어셈 레벨에서는 구조체를 구성하는 멤버 변수 하나하나를 넘긴다.

3. 변수 value의 값을 확인할 수 없는 이유

변수가 할당되는 시점은 선언 시점이 아닌 메모리에 값을 배치하는 초기화 시점에 할당된다.

```
44 int main(void)
45 {
46     int value;
47     struct _member account;
48
49     member my_account = { };
50
51     if (init_member(&my_account, "gogo", 19) == MEMBER_ALLOC_FAIL)
52     {
53         printf("member 구조체 할당 실패!\n");
54         exit(-1);
55     }
```

구조체를 어셈블리어로 분석하기

EDDI
Electronic Design Development Institute

18) struct.c의 어셈블리어 분석 최종 정리

4. printf 함수를 puts 함수로 바꿔서 컴파일

```
if (init_member(&my_account, "gogo", 19) == MEMBER_ALLOC_FAIL)
{
    printf("member 구조체 할당 실패!\n");
    exit(-1);
}
```

carbon
carbon.now.sh

```
0x0000555555555361 <+62>:    callq  0x555555555229 <init_member>
0x0000555555555366 <+67>:    xor    $0x1,%eax
0x0000555555555369 <+70>:    test   %al,%al
0x000055555555536b <+72>:    je     0x555555555383 <main+96>
0x000055555555536d <+74>:    lea    0xcb4(%rip),%rdi        # 0x555555556028
0x0000555555555374 <+81>:    callq  0x5555555550e0 <puts@plt>
0x0000555555555379 <+86>:    mov    $0xffffffff,%edi
0x000055555555537e <+91>:    callq  0x555555555130 <exit@plt>
```

5. bool 타입은 init_member의 어셈블리어 코드에서 true는 1, false는 0 으로 확인할 수 있었다.

6. 과제(malloc으로 할당한 member 구조체 멤버 변수 name을 해제하는 함수)

```
38 // 과제
39 void free_member(member account)
40 {
41     free(account.name);
42 }
```

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

1) 함수포인터 인터페이스

강사님 말씀:

특수한 기법들도 추가되어있지만, GitHub에 올려도 상관없을 것 같다.
이미 리눅스 커널에서도 많이 쓰고 있고, 굳이 숨길 필요가 있나 생각이 들음.

(union은 시간관계로 설명 듣지 못했으나, 참고용으로 적어놓았다)
union은 메모리 절약하려고 사용하는데, 요즘은 사실 안쓴다.
굉장히 특수한 기법으로 사용하는게 하나 있다.

파일은 총 4개로, 각 파일 이름은 다음과 같다.

- main.c
- device.c
- device.h
- polymorphism_call.h

이 중에서 main.c와 polymorphism_call.h 파일만 보여주겠다.

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

2) main.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

#include "device.h"

enum polumorphism_protocol {
    CAMERA,
    DCMOTOR,
    BLDC,
    PMSM,
    ACIM,
    LED,
    I2C,
    SPI,
    CAN,
    ECAP,
    END
};

#define POLY_CALL_BUFFER_COUNT  (END)
#define POLY_CALL_BUFFER        ((END) - (1))

typedef void (* polymorphism_call_ptr_t)(void);

void polymorphism_not_impl(void)
{
    printf("아직 구현되지 않은 스펙입니다!\n");
}
```

```c
const polymorphism_call_ptr_t polymorphism_call_table[POLY_CALL_BUFFER_COUNT] = {
    [0 ... POLY_CALL_BUFFER] = polymorphism_not_impl,
    #include "polymorphism_call.h"
};

void recv_command_from_pc(void)
{
    int command;

    printf("명령이 들어온다 가정하고 진행(랜덤으로 만들기)!\n");

    command = rand() % POLY_CALL_BUFFER_COUNT;
    printf("수신된 명령: %d(가정 - 랜덤)\n", command);

    polymorphism_call_table[command]();
}

int main(void)
{
    srand(time(NULL));

    recv_command_from_pc();

    return 0;
}
```

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

3) polymorphism_call.h

```
1  #ifndef ASSEMBLYTEST_POLYMORPHISM_CALL_H
2  #define ASSEMBLYTEST_POLYMORPHISM_CALL_H
3
4  #include "device.h"
5
6  #define __POLYMORPHISM_CALL(nr, sym) [nr] = sym,
7
8  __POLYMORPHISM_CALL(0, proc_camera)
9  __POLYMORPHISM_CALL(1, proc_dc_motor)
10 __POLYMORPHISM_CALL(2, proc_bldc)
11 __POLYMORPHISM_CALL(3, proc_pmsm)
12 __POLYMORPHISM_CALL(4, proc_acim)
13 __POLYMORPHISM_CALL(5, proc_led)
14 __POLYMORPHISM_CALL(6, proc_i2c)
15 __POLYMORPHISM_CALL(7, proc_spi)
16 __POLYMORPHISM_CALL(8, proc_can)
17 __POLYMORPHISM_CALL(9, proc_ecap)
18
19 #endif // ASSEMBLYTEST_POLYMORPHISM_CALL_H
```

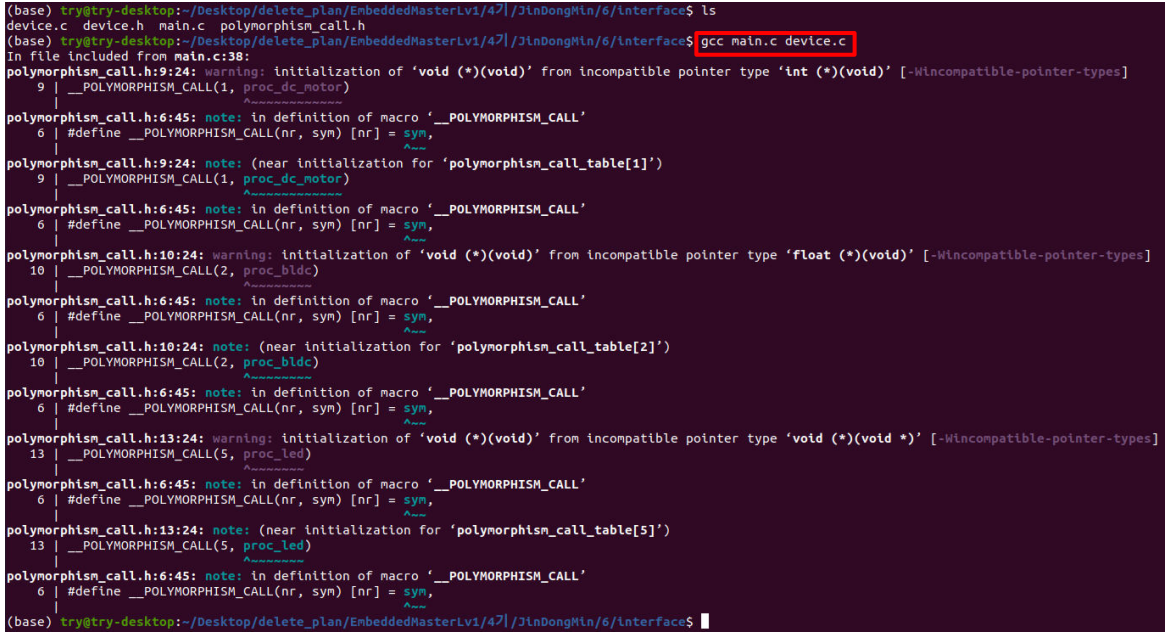
함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

4) 컴파일하는 방법

$ gcc main.c device.c

컴파일 시 warning 메시지가 조금 나오는데 void 포인터를 쓰면 나오는 메시지들이다.



함수 포인터 인터페이스

5) 실행 결과

$ ./a.out

어떤 명령이 들어오는 지에 따라 각각 다른 센서들이 동작함.

```
(base) try@try-desktop:~/Desktop/delete_plan/EmbeddedMasterLv1/4기/JinDongMin/6/interface$ ls
a.out  device.c  device.h  main.c  polymorphism_call.h
(base) try@try-desktop:~/Desktop/delete_plan/EmbeddedMasterLv1/4기/JinDongMin/6/interface$ ./a.out
명령이 들어온다 가정하고 진행(랜덤으로 만들기)!
수신된 명령: 3(가정 - 랜덤)
PMSM Motor Processing
(base) try@try-desktop:~/Desktop/delete_plan/EmbeddedMasterLv1/4기/JinDongMin/6/interface$ ./a.out
명령이 들어온다 가정하고 진행(랜덤으로 만들기)!
수신된 명령: 0(가정 - 랜덤)
Camera Processing
(base) try@try-desktop:~/Desktop/delete_plan/EmbeddedMasterLv1/4기/JinDongMin/6/interface$ ./a.out
명령이 들어온다 가정하고 진행(랜덤으로 만들기)!
수신된 명령: 0(가정 - 랜덤)
Camera Processing
(base) try@try-desktop:~/Desktop/delete_plan/EmbeddedMasterLv1/4기/JinDongMin/6/interface$ ./a.out
명령이 들어온다 가정하고 진행(랜덤으로 만들기)!
수신된 명령: 0(가정 - 랜덤)
Camera Processing
(base) try@try-desktop:~/Desktop/delete_plan/EmbeddedMasterLv1/4기/JinDongMin/6/interface$ ./a.out
명령이 들어온다 가정하고 진행(랜덤으로 만들기)!
수신된 명령: 0(가정 - 랜덤)
Camera Processing
(base) try@try-desktop:~/Desktop/delete_plan/EmbeddedMasterLv1/4기/JinDongMin/6/interface$ ./a.out
명령이 들어온다 가정하고 진행(랜덤으로 만들기)!
수신된 명령: 7(가정 - 랜덤)
25LC010A EEPROM Processing
(base) try@try-desktop:~/Desktop/delete_plan/EmbeddedMasterLv1/4기/JinDongMin/6/interface$
```

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

6) 코드 분석 - 1

```
void recv_command_from_pc(void)
{
    int command;

    printf("명령이 들어온다 가정하고 진행(랜덤으로 만들기)!\n");

    command = rand() % POLY_CALL_BUFFER_COUNT;
    printf("수신된 명령: %d(가정 - 랜덤)\n", command);

    polymorphism_call_table[command]();
}
```

carbon
carbon.now.sh

switch 대신 함수포인터를 사용함.

command(명령어라고 가정)를 받아서 위 사진의 빨간 박스 내에 있는 코드를 실행함

함수 포인터 인터페이스

6) 코드 분석 - 2

```
enum polumorphism_protocol {
    CAMERA,
    DCMOTOR,
    BLDC,
    PMSM,
    ACIM,
    LED,
    I2C,
    SPI,
    CAN,
    ECAP,
    END
};

#define POLY_CALL_BUFFER_COUNT  (END)
#define POLY_CALL_BUFFER        ((END) - (1))
```


enum의 마지막에 END를 배치하여, 프로토콜에 사용되는 개수를 매크로 상수로 선언

배열 선언할 때 배열 길이 명시에서 사용
배열의 전체 요소를 초기화할 때 사용⇒

(자세한 내용은 슬라이드 두 장 뒤로...)

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

6) 코드 분석 - 3

```
typedef void (* polymorphism_call_ptr_t)(void);
```

void (*)(void)라는 함수 포인터 대신 **polymorphism_call_ptr_t**로 사용하겠다라는 typedef 선언

```
void polymorphism_not_impl(void)
{
    printf("아직 구현되지 않은 스펙입니다!\n");
}
```

위의 함수는 구현하지 않은 프로토콜 기능이 있을 경우에 사용할 함수

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

6) 코드 분석 - 4

```
const polymorphism_call_ptr_t polymorphism_call_table[POLY_CALL_BUFFER_COUNT] = {
    [0 ... POLY_CALL_BUFFER] = polymorphism_not_impl,
    #include "polymorphism_call.h"
};
```

carbon
carbon.now.sh

위의 코드는 배열을 선언과 동시에 초기화하고 있다.

배열 타입: void (*) (void)
배열 이름: polymorphism_call_table
배열 길이: POLY_CALL_BUFFER_COUNT (매크로 상수)
초기화 방법

- [0 … POLY_CALL_BUFFER]의 의미: 0 부터 9 까지 구현되지 않았을 경우에 사용하는 함수로 초기화하고 있다.
- 배열의 각 요소를 초기화하는 방법인 #include “polymorphism_call.h”는 다음 장에서 분석하겠다.

함수 포인터 인터페이스

6) 코드 분석 - 5

```
1 #ifndef ASSEMBLYTEST_POLYMORPHISM_CALL_H
2 #define ASSEMBLYTEST_POLYMORPHISM_CALL_H
3
4 #include "device.h"
5
6 #define __POLYMORPHISM_CALL(nr, sym) [nr] = sym,
7
8 __POLYMORPHISM_CALL(0, proc_camera)
9 __POLYMORPHISM_CALL(1, proc_dc_motor)
10 __POLYMORPHISM_CALL(2, proc_bldc)
11 __POLYMORPHISM_CALL(3, proc_pmsm)
12 __POLYMORPHISM_CALL(4, proc_acim)
13 __POLYMORPHISM_CALL(5, proc_led)
14 __POLYMORPHISM_CALL(6, proc_i2c)
15 __POLYMORPHISM_CALL(7, proc_spi)
16 __POLYMORPHISM_CALL(8, proc_can)
17 __POLYMORPHISM_CALL(9, proc_ecap)
18
19 #endif // ASSEMBLYTEST_POLYMORPHISM_CALL_H
```

carbon
carbon.now.sh

polymorphism_call.h

여기서 매크로를 사용하여 각 요소를 초기화하였다.

전처리가 끝난 코드는 다음 슬라이드에서 보면 된다.

```
[0] = proc_camera,
[1] = proc_dc_motor,
[2] = proc_bldc,
[3] = proc_pmsm,
[4] = proc_acim,
[5] = proc_led,
[6] = proc_i2c,
[7] = proc_spi,
[8] = proc_can,
[9] = proc_ecap,
```

carbon
carbon.now.sh

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

6) 코드 분석 - 5

```
const polymorphism_call_ptr_t polymorphism_call_table[POLY_CALL_BUFFER_COUNT] = {
    [0 ... POLY_CALL_BUFFER] = polymorphism_not_impl,
    [0] = proc_camera,
    [1] = proc_dc_motor,
    [2] = proc_bldc,
    [3] = proc_pmsm
    [4] = proc_acim,
    [5] = proc_led,
    [6] = proc_i2c,
    [7] = proc_spi,
    [8] = proc_can,
    [9] = proc_ecap,
};
```


위 코드는 C99 이상인 경우에 사용할 수 있다.
C99 미만의 경우에는 배열을 { func1, func2, func3 } 와 같이 직접 초기화해주어야 한다.

(배열 초기화에 관하여 참고하면 좋은 링크)

- https://en.cppreference.com/w/c/language/array_initialization
- https://www.digitalocean.com/community/tutorials/initialize-an-array-in-c

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

7) 프로토콜 추가에 의해 함수를 추가해야 할 경우

- device.c 파일에 함수를 정의한다.
- device.h 파일에 함수 프로토타입을 작성한다.
- polymorphism_call.h 파일에 새로 작성한 함수를 배열에 추가하기 위해 매크로를 사용하여 작성한다.
- main.c 파일의 polymorphism_protocol 열거형에 새 프로토콜을 추가한다.

8) 이 예제를 사용할 때의 주의사항
배열을 사용하는 파일에서는 함수를 선언한 헤더파일(여기서는 device.h)을 include 해야 한다.

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

9) 추가적인 지식 + 수업 동영상 (https://youtu.be/A35TwcHhYOU)

시간: 1:55:08, 1:58:12

(함수포인터를 더 응용을 한다면…)
응용하면 C로 디자인 패턴을 할 수 있는데, 함수 포인터의 별들의 전쟁이 시작된다고 한다 ㅋㅋㅋ
필요하면 별이 4~5개 붙을 수도 있다.

위와 같이 별이 많이 필요한 경우는 동영상 스트리밍 처리를 예시로 들수있다.

또, 함수 포인터를 반환하고 그 함수 포인터가 함수를 호출하는 방식도 가질 수 있다.
그 상황이 디자인 패턴이다.

함수 포인터 인터페이스

EDDI
Electronic Design Development Institute

9) 추가적인 지식 + 수업 동영상 (https://youtu.be/A35TwcHhYOU)

시간: 2:15:28

if문 돌아갈 때 파이프라인이 깨진다.
파이프라인이 깨질 때 최대의 약점은 캐시를 버린다.

함수 포인터 배열의 경우, 캐시니까 가지고 있는다.
만약에 다른 프로토콜 처리가 들어온다면, 메모리에 접근해서 가져오는 것이 아닌 캐시에서 그냥 읽어버린다.
그러면 CPU 클럭 사이클도 아낄 수 있다.

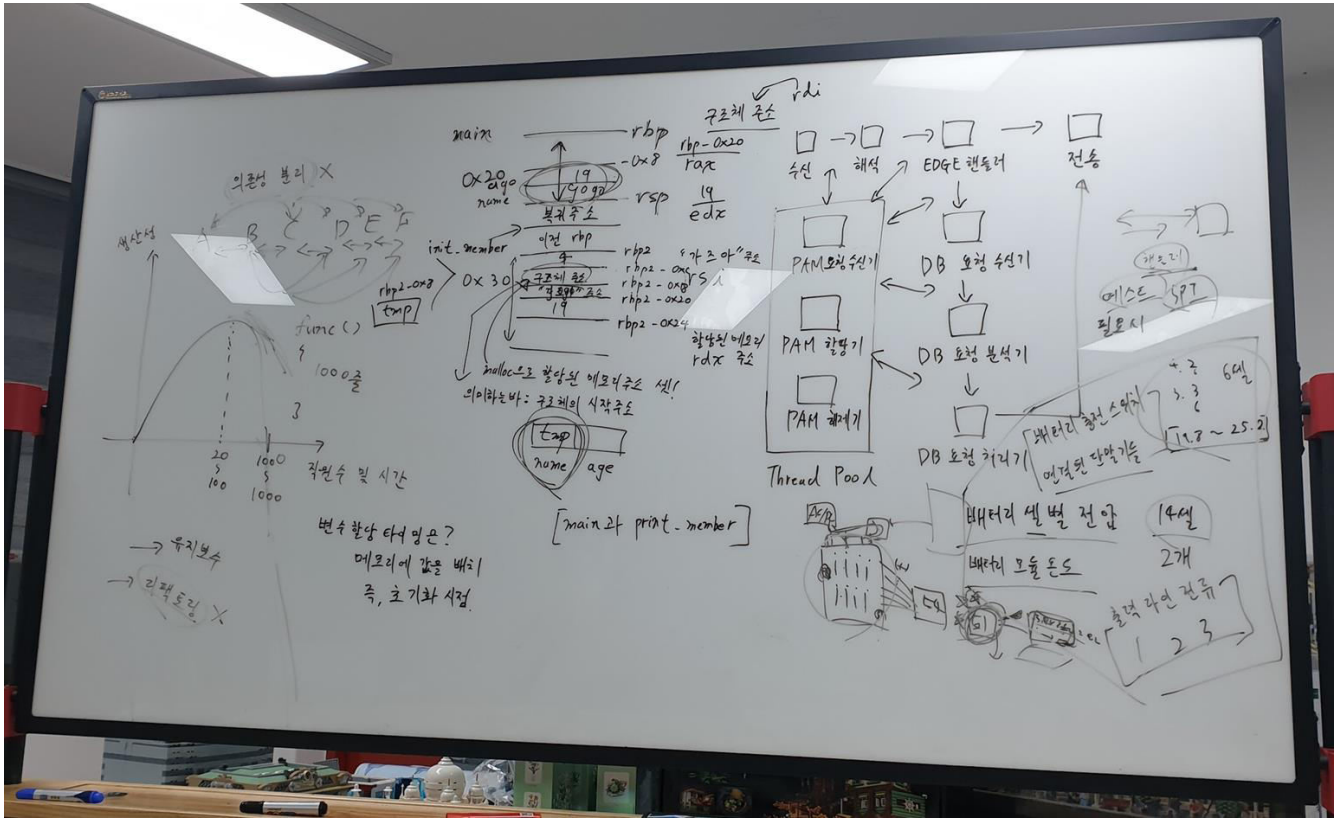
또한, AVR 최신버전은 C98 이라고 한다. 그러므로 배열을 초기화할 때 직접 일일이 함수를 명시해야 한다.

시간: 2:23:10

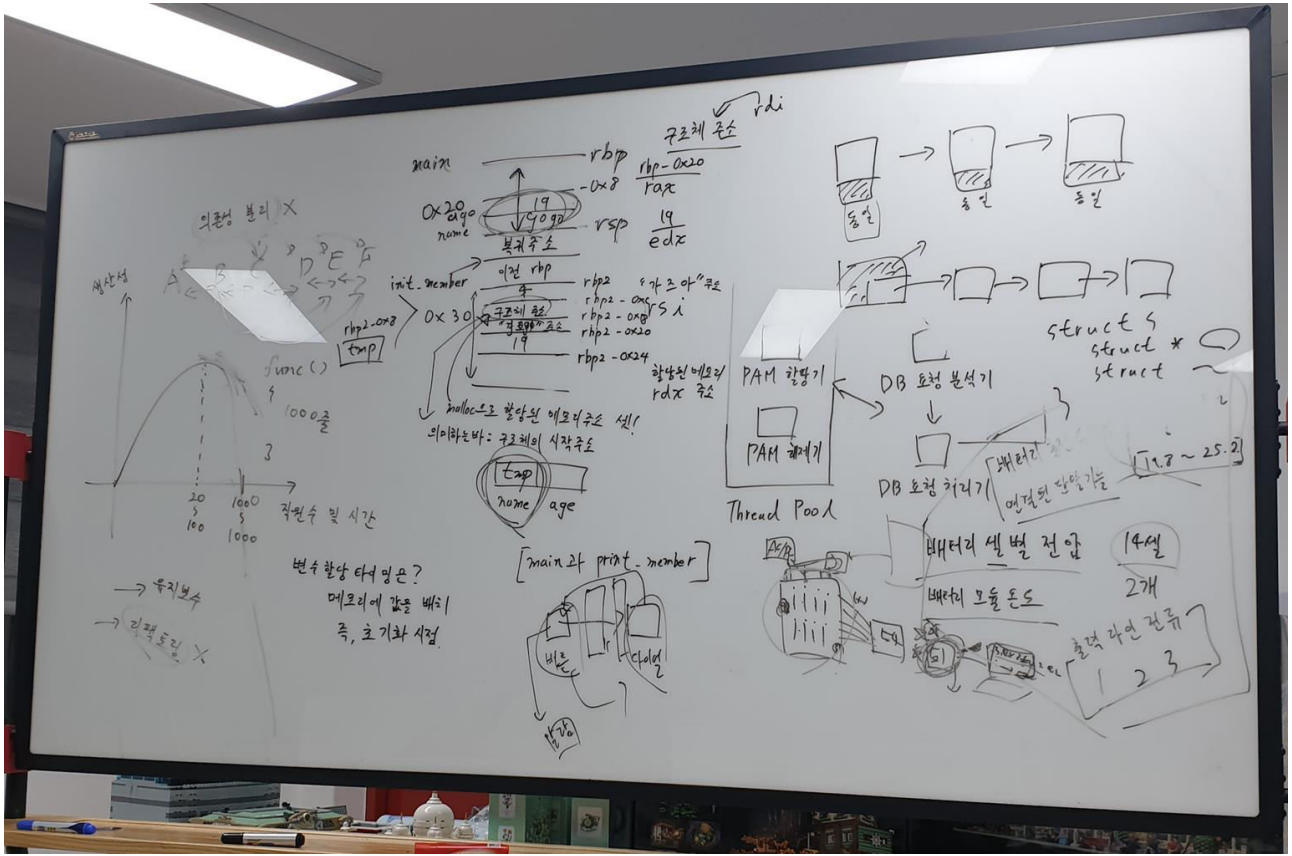
수업에서는 my_account 구조체를 초기화하지 않고 디버깅할 때 애를 먹었는데 { } 로 초기화하니 디버깅할 때 스택에서 구조체 변수의 위치를 찾기 수월했다.

그러므로 여기서 얻은 점은 **향후 디버깅을 위해서 변수를 초기화하는 습관을 들이면 좋을 것 같다!**

수업내용 사진


main
rbp
rsp
rbp-0x20
rax
edx
rdi
구조체 주소
이전 rbp
rbp2
rbp2 - 0x8
rbp2 - 0x20
rbp2 - 0x24
init_member
0x30
malloc으로 할당된 메모리주소 넣기
의미하는바: 구조체의 시작주소
tmp
name
age
main과 print_member
변수 할당 타이밍은?
메모리에 값을 배치
즉, 초기화 시점
생산성
직원수 및 시간
func()
1000줄
수신
해석
EDGE 핸들러
전송
PAM요청 수신기
PAM 할당기
PAM 제거기
Thread Pool
DB 요청 수신기
DB 요청 분석기
DB 요청 처리기
배터리 셀 별 전압
14셀
2개
배터리 모듈 온도
에스드
필요시
EDDI
Electronic Design
Development Institute

수업내용 사진


main()
rbp
rbp-0x20
rax
-0x8
rsp
edx
복귀주소
이전 rbp
rbp2
rbp2 - 0x4
rbp2 - 0x8
rbp2 - 0x20
rbp2 - 0x24
rdi
구조체 주소
rsi
init_member
0x30
할당된 메모리 rdx 주소
malloc으로 할당된 메모리주소 센서
의미하는바: 구조체의 시작주소
tmp
name
age
[main과 print_member
func()
1000줄
생산성
직원수 및 시간
20
100
1000
1000
변수 할당 타입 및 은?
메모리에 값을 빼기
즉, 초기화 시점
유지보수
리팩토링
PAM 합병기
PAM 제거기
Thread Pool
DB 요청 분석기
DB 요청 처리기
배터리
인접된 잘맞는놈
11.8 ~ 25.2
배터리 셀 별 전압
14셀
2개
배터리 모듈 온도
출력 라인 전류
1
2
3
struct
struct *
struct
돌입
돌입
돌입

CONTENTS

EDDI
Electronic Design
Development Institute

1) 형식은 자유롭게~~~

<공부 내용을 적어주세요.>