Test 1

임베디드스쿨2기

Lv1과정

2021. 04. 30

박태인

1. Q1)

1) C언어는 함수를 호출 할 때 마다 무엇을 생성하는가?
어째서 재귀호출이 일반적인 Loop 보다 성능이 떨어지는 것인가?
이에 대해 상세히 기술하시오.

나의답 :

C언어는 함수를 호출 할 때마다, 이전 함수 복귀 주소를 Main 또는 이전 함수의의 rsp-8 위치에 복귀 주소를 push 하고 호출 하는 함수의 새로운 스택으로 jump 이동하게 된다.

재귀 호출이 일반적인 Loop 보다 성능이 떨어지는 이유?

시간이 많이 걸리고 공간을 많이 차지 하기 때문 입니다.
재귀함수를 사용 했을 경우 호출한 함수에서 다시 또 자신의 함수를 호출하는 구조 입니다.

함수는 기본적으로 새로운 스택을 만들어 내야 하고 계속해서 새로운 메모리 공간이 필요하게 됩니다.
물론 결론이 나는 함수(= 더이상 재귀하지 않는) 메모리의 경우 결론을 도출하고 사라지지만 기본적으로 한번에 많은 함수가 호출이 되어 메모리 공간이 더욱 많이 필요하게 됩니다.

이러한 원리로 재귀 호출이 일반적인 Loop 보다 성능이 떨어 질 수 있다고 생각합니다.

Answer :

C언어는 함수를 호출 할 때마다 Stack을 생성한다.
함수 호출의 경우 어셈블리어를 보면 Stack Frame을 만들고 해제하는 코드가 들어간다.
이 부분이 쓸대 없이 지속적으로 반복 되며 call이 발생하므로
오히려 mov cmp jmp인 Loop 보다 효율이 좋지 못하다.
(파이프라인은 둘 다 깨진다)

종합 :

나의 답에서는 **함수 호출이 반복 됨으로써 사용하는 메모리 공간이 늘어남으로써** 재귀 호출이 성능이 떨어 질 수 있다는 것을 설명하였고,
답에서 이 부분이 어**셈블리 상에서 stack을 생성하고 해제하는 코드가 추가로 들어감에 따라 이것이 반복**되는 재귀호출이 일반적인 루프인 **mov cmp jmp 보다 성능이 떨어 질 수 있다는 판단**이 됩니다.

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

2) 회원이름, 나이, 전화번호, 거주지를 입력 받도록 한다. (나의 답)

```c
int main(void)
{
        char name[10];
        char address[20];
        int age;
        int phone;

        printf("이름을 입력 하세요. : ");
        scanf("%s", name);

        printf("거주지를 입력 하세요. : ");
        scanf("%s", address);

        printf("나이를 입력 하세요. : ");
        scanf("%d", &age);

        printf("전화번호를 입력 하세요. ");
        scanf("%d", &phone);

//      printf("당신의 이름은 %s \n 거주지는 %s \n 나이는 %d \n 전화번호는 0%d 입니다.\n",name, address, age, phone);
```

```
이름을 입력 하세요. : 박태인
거주지를 입력 하세요. :  수원
나이를 입력 하세요. : 33
전화번호를 입력 하세요. 01090761365
당신의 이름은 박태인
 거주지는 수원
 나이는 33
 전화번호는 01090761365 입니다.
```

나의 답 :
- 이름, 주소를 정보를 받을 수 있게끔 캐릭터형 배열 생성

.

- 나이, 전화번호를 받기 위해 int 형 배열 생성.
- sacnf 와 printf 를 통해 각각의 배열에 정보를 입력 받음

.

- 문제점 : 함수로 표현하진 못함.

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

2) 회원이름, 나이, 전화번호, 거주지를 입력 받도록 한다. (Answer)

```
int main(void)
{
        int age;
        char name[32] = { 0 };
        char phone[32] = { 0 };
        char city[32] = { 0 };
        char street[128] = { 0 };
        char detail[128] = { 0 };

        input_info(name, &age, phone, city, street, detail);

        printf("이름: %s, 나이: %d, 전화번호: %s\n거주지: %s시 %s %s\n",
                name, age, phone, city, street, detail);
}
```

우선 main 문을 살펴 보자.
- 나이를 제외한 이름, 전화번호, 도시, 거리, 상세를 전부 char형 배열로 { 0 }; 초기화 하였습니다.
- input_info 라는 함수를 통해서 위의 정보들을 받습니다.
  (여기서 는 &age로 인자를 받습니다. 나머지는 배열이고 **배열 이름이 곧 배열의 첫 주소를 가르키기 때문**입니다.)

- 그리고 호출 된 함수가 완성 되면 각각의 변수의정보를 print 합니다.
- 자 이제 상세히 iput_info 함수로 넘어가 보자.

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

2) 회원이름, 나이, 전화번호, 거주지를 입력 받도록 한다. (Answer)

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

void input_info(char *name, int *age, char *phone, char *city, char *street, char *detail)
{
        printf("회원 정보를 입력해주세요.\n이름: ");
        scanf("%s", name);

        printf("나이: ");
        scanf("%d", age);

        printf("전화 번호: ");
        scanf("%s", phone);

        printf("도시: ");
        scanf("%s", city);

        printf("도로명: ");
        scanf("%s", street);

        printf("상세 주소: ");
        scanf("%s", detail);
}
```

- input_info 함수를 분석해 보자.
  - ㄴ 우선 함수의 인자 변수는 메인 함수의 변수와는 다른 것이다.
  - ㄴ 이는 어셈블리 분석을 통해서도 알 수 있었다.
  - ㄴ 여기서 모든 인자는 주소값으로 받기 위해 인자에 *를 붙여 준다.
    (scanf를 활용하기 위함 이다. scanf는 두 번째 값에 주소 값을 넣어야 한다.

- 이 함수의 경우 이렇듯 함수의 정보를 인자로 받고, 각 변수를 통해 받은 값이 main으로 전달 된다.

```
회원 정보를 입력해주세요.
이름: 박태인
나이: 33
전화 번호: 01090761365
도시: 수원
도로명: 평동로
상세 주소: 114-1
이름: 박태인, 나이: 33, 전화번호: 01090761365
거주지: 수원시 평동로 114-1
```

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

3) 적당히 여러 회원을 입력한 이후 거주지가 같은 사람들만 출력해 본다. (Answer)

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

typedef struct _member_info member_info;

struct _member_info
{
        int age;
        char name[32];
        char phone[32];
        char city[32];
        char street[128];
        char detail[128];
};

int total_member;

member_info *init_member_info(void)
{
        member_info *tmp = (member_info *)malloc(sizeof(member_info) * 64);
}
```

구조체 선언을 한 부분이다.

Typedef 을 통해서 struct member_info를 **member_info로 단축 정의** 합니다.

기본적으로_member_info구조체는
Int 형 나이를 제외하고는
이름, 전화번호, 도시, 거리, 상세는 모두 캐릭터형 배열로 선언 되어 있습니다.

그리고 int형 total_member;라는 전역 변수를 선언 하였다.
ㄴ 추후에 사용 될 것이다.

그리고 회원들의 정보를 저장하기 위한 메모리를 동적할당 하기 위해
malloc함수를 사용하게 됩니다. **Malloc 함수는 성공하면 메모리 주소를 반환**하기 때문에
member_info **구조체에 포인터(*) 반환형**의 init_member_info 함수를 생성 합니다.

이 때, 반환형 포인터(*) malloc(크기) 가 malloc의 원형이므로,

(member_info *)malloc(sizeof(member_info) * 64);로 사용이 됩니다.
ㄴ 구조체 포인터 ㄴ구조체 크기 x 64

이렇게 생성된 메모리는 member_info *tmp
ㄴ 구조체 포인터형인 tmp에 저장됩니다.
ㄴ **즉, 회원 정보가 들어 있는 구조체의 주소가 저장됨.**

추가 조사

- 동적할당 malloc 사용

→ malloc 사용법에 대해 알아 보자.
메모리를 사용하기 위해 Malloc 함수로 사용할 메모리의 공간을 확보해야 합니다.
- 헤더는 stdlib.h 에 선언되어 있다.
- 원하는 시점에 원하는 메모리를 할당 할 수 있다고 해서 '동적 할당' 방법이라고도 한다.

- 포인터 = malloc(크기);
  Void *malloc(size_t_Size);
  성공하면 메모리 주소를 반환, 실패하면 NULL을 반환함.

- 변수는 Stack에 메모리를 생성하지만 malloc 함수를 Heap 영역에 메모를 생성한다.
  ㄴ 영역의 예시로 우측의 그림을 참고 하면 되지만 시스템 마다 다를 수 있다.
- Stack 과 Heap의 가장 큰 차이는 Stack 함수를 사용한 뒤에 따로 처리를 하지 않아도 되지만, Malloc 함수로 heap에 할당한 메모리는 반드시 메모리 해를 해주어야 한다.(필수사항이다)

- 후처리(free)
  ㄴ free(포인터)
  → void free(void *_Block);

35-2 스택과 힙(Windows x86/32비트)

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

3) 적당히 여러 회원을 입력한 이후 거주지가 같은 사람들만 출력해 본다. (Answer)

```
void input_info(member_info *mi)
{
        bool run = true;
        int num;

        while (run)
        {
                printf("회원 정보를 계속 입력하시겠습니까 ? 1(yes), 0(no)\n");
                scanf("%d", &num);

                switch (num)
                {
                        case 1:
                                printf("회원 정보를 입력해주세요.\n이름: ");
                                scanf("%s", mi[total_member].name);

                                printf("나이: ");
                                scanf("%d", &mi[total_member].age);

                                printf("전화 번호: ");
                                scanf("%s", mi[total_member].phone);

                                printf("도시: ");
                                scanf("%s", mi[total_member].city);

                                printf("도로명: ");
                                scanf("%s", mi[total_member].street);

                                printf("상세 주소: ");
                                scanf("%s", mi[total_member].detail);

                                total_member++;

                                break;

                        case 0:
                                printf("회원 입력을 종료합니다.\n");
                                run = false;
                                break;
                }
        }
}
```

input_info 함수는 구조체 포인터형 mi를 인자로 받아 정보를 기입하는 함수이다.

여기서 while문 조건에 **Bool 변수형을 사용**하는데,
Bool 변수형을 사용하기 위해서는
#include <stdbool.h> 를 선언하고,이것은 **bool, true, false가 정의된 것**이다.

While문 조건이 run 인데, run이 초기 값이 true 이므로 함수가 실행되면 초기부터 실행이 된다.

회원정보를 입력 하겠느냐 안하겠느냐의 조건은

Switch 문으로 선택을 한다 1 (yes) 0 (no)
여기서 회원들의 정보는 **구조체 mi[멤버 번호].name** 등 의 방식으로 받는다.
여기서 age의 경우 구조체에서 배열이 아닌 int 형 이므로
Scanf 사용시 & 를 붙여 줘야 한다.

이렇게 회원정보는 받고, 만약 사용자가 더이상의 회원정보 입력을 원하지 않으면 Case 0 이 되어 switch 문을 빠져 나오게 된다.

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

3) 적당히 여러 회원을 입력한 이후 거주지가 같은 사람들만 출력해 본다. (Answer)

```
void print_member_info(member_info *mi)
{
        int i;

        if (mi->name)
        {
                for (i = 0; i < total_member; i++)
                {
                        printf("이름: %s, 나이: %d, 전화번호: %s\n거주지: %s시 %s %s\n",
                                mi[i].name, mi[i].age, mi[i].phone,
                                mi[i].city, mi[i].street, mi[i].detail);
                }
        }
}
```

ㄴ 입력 받은 멤버들의 정보를 출력하는 함수 이다.
여기에서 **if 조건문은 값이 있으면을 의미 하는 것이다!!** 따라서, Mi→age, mi→phone 등의 조건을 써도 마찬가지 인 것이다.

```
int main(void)
{
        member_info *mi;

        mi = init_member_info();
        input_info(mi);

        print_member_info(mi);

        free(mi);
}
```

→ 최종적으로 main 문 실행 절차를 살펴 보자.
- member_info * 구조체 포인터 형태인 mi 를 생성한다.
- Mi = init_member_info()를 통해서 구조체에 동적할당을 실시한다.
- input_info(mi) 구조체 인자를 받아서 scanf로 회원들의 정보를 입력한다.
- 입력 받은 구조체의 정보들을 출력한다.
- 동적 할당 하였던 구조체의 메모리들을 free 시켜 준다.

프로그램 실행 예시 →

```
회원 정보를 계속 입력하시겠습니까 ? 1(yes), 0(no)
1
회원 정보를 입력해주세요.
이름: 박태인
나이: 33
전화 번호: 010
도시: 수원
도로명: 평동로
상세 주소: 114-1
회원 정보를 계속 입력하시겠습니까 ? 1(yes), 0(no)
1
회원 정보를 입력해주세요.
이름: 박찬석
나이: 30
전화 번호: 010
도시: 부산
도로명: 하신중앙로
상세 주소: 60
회원 정보를 계속 입력하시겠습니까 ? 1(yes), 0(no)
1
회원 정보를 입력해주세요.
이름: 노정숙
나이: 58
전화 번호: 010
도시: 부산
도로명: 하신중앙로
상세 주소: 60
회원 정보를 계속 입력하시겠습니까 ? 1(yes), 0(no)
0
회원 입력을 종료합니다.
이름: 박태인, 나이: 33, 전화번호: 010
거주지: 수원시 평동로 114-1
이름: 박찬석, 나이: 30, 전화번호: 010
거주지: 부산시 하신중앙로 60
이름: 노정숙, 나이: 58, 전화번호: 010
거주지: 부산시 하신중앙로 60
```

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

4) 10대, 20대, 30대 별로 출력해보도록 한다. (Answer)

```
typedef struct _member_info member_info;

struct _member_info
{
        int age;
        char name[32];
        char phone[32];
        char city[32];
        char street[128];
        char detail[128];
};

int total_member;

int age[64] = { 10, 11, 12, 15, 16,
                                18, 20, 21, 24, 27,
                                33, 34, 32, 36, 38, 39 };
char name[64][32] = { "김윤환", "김택용", "이영호", "조기석", "임요환",
                                "마주작", "진영수", "염보성", "김명운", "임홍구",
                                "박경수", "이영수", "김성환", "이재호", "김창수", "도재욱" };
char phone[64][32] = { "010-1111-1234", "010-1111-1234", "010-1111-1234", "010-1111-1234", "010-1111-1234",
                                "010-1111-1234", "010-1111-1234", "010-1111-1234", "010-1111-1234", "010-1111-1234",
                                "010-1111-1234", "010-1111-1234", "010-1111-1234", "010-1111-1234", "010-1111-1234", "010-1111-1234" };
char city[64][32] = { "서울", "서울", "서울", "부산", "서울",
                                "대전", "대전", "대구", "전주", "대구",
                                "전주", "대전", "서울", "서울", "대전", "서울" };
char street[64][128] = { 0 };
char detail[64][128] = { 0 };
```

구조체 member_info를 member_info로 typedef 정의 합니다.
구조체 내용은 앞선 내용과 동일 합니다.

앞서 방식과 다른점은 회원의 정보를 이중배열을 통해 나타 냅니다.
이중 배열 구조의 예를 들면 아래와 같은 구조이다.

하나더 하면 int **arr[2][4]**

[0]				[1]			
[0][0]	[0][1]	[0][2]	[0][3]	[1][0]	[1][1]	[1][2]	[1][3]

따라서 아래 것 중 하나를 예를 들어 이름 정보의 name[64][32] 는
[64] : 전체 종류 , [32] : 각각의 개수 즉, 64개의 정보를 32byte씩.

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

4) 10대, 20대, 30대 별로 출력해보도록 한다. (Answer)

```
member_info *init_member_info(void)
{
        member_info *tmp = (member_info *)malloc(sizeof(member_info) * 64);
}

void input_info(member_info *mi)
{
        bool run = true;
        int num;
        int i;

        for (i = 0; i < 16; i++)
        {
                mi[i].age = age[i];

                strcpy(mi[i].name, name[i]);
                strcpy(mi[i].phone, phone[i]);
                strcpy(mi[i].city, city[i]);
                strcpy(mi[i].street, street[i]);
                strcpy(mi[i].detail, detail[i]);
        }

        total_member = 16;
}
```

→ malloc 함수를 통해서 구조체의 동적 할당을 실행하는 함수

→ input_info 함수를 통해 회원들의 정보를 정리한다.
16명 회원의 정보를 동적 할당된 구조체 메모리에 할당합니다.

for문을 통해서

mi[i].age : I 번째 구조체의 age에 age[i] 정보를 입력.
ㄴ **나이의 경우 숫자 값 이기에 바로 대입.**

문자열 값의 경우 strcpy 함수를 사용하여 값을 복사한다.
아래는 strcpy의 함수 원형이며
예를 들어 **Strcpy(s1, s2) → s2의 문자열을 s1로 복사 한다는 의미**.

- strcpy(대상문자열, 원본문자열);
  - char *strcpy(char *_Dest, char const *_Source);
  - 대상문자열의 포인터를 반환

즉, 하나 문장을 예로 들어보면
strcpy(mi[i].phone, phone[i]);
ㄴ **phone배열의 값을 mi 구조체 I 번째의 phone 배열에 복사.**

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

4) 10대, 20대, 30대 별로 출력해보도록 한다. (Answer)

```
void print_condition_member_info(member_info *mi, int num)
{
        int i;

        if (mi->name)
        {
                for (i = 0; i < total_member; i++)
                {
                        if ((mi[i].age / num) == 1)
                        {
                                printf("이름: %s, 나이: %d, 전화번호: %s\n거주지: %s시 %s %s\n",
                                        mi[i].name, mi[i].age, mi[i].phone,
                                        mi[i].city, mi[i].street, mi[i].detail);
                        }
                        else
                        {
                                continue;
                        }
                }
        }
}

void print_member_info(member_info *mi)
{
        int i;

        if (mi->name)
        {
                for (i = 0; i < total_member; i++)
                {
                        printf("이름: %s, 나이: %d, 전화번호: %s\n거주지: %s시 %s %s\n",
                                mi[i].name, mi[i].age, mi[i].phone,
                                mi[i].city, mi[i].street, mi[i].detail);
                }
        }
}
```

→ print condition 함수는 구조체 정보와 숫자 num을 받아
원하는 나이대의 정보를 출력하기 위한 함수 입니다.

여기서 int num 인자에 10을 받으면 10대 사람만 출력하게 되는 것 입니다.

여기서도 if 문 조건에 mi → name 이라는 값이 있으면이라는 의미가
적용 되었고, toal member의 숫자 만큼

for문을 반복한다.

두번째 if 문의 조건은
(mi[i].age / num) == 1 은 구조체 나이 값을 10으로 나누었을 때
몫이 1 인경우만 10대 이므로 **10대의 조건을 구하기 위함**이다.

→ print_member_info 함수는 10대의 조건 뿐만 아닌
구조체에 있는 전체의 값을 출력하는 함수이다.

1. Q2) 2 ~ 4

- 콘솔창에서 회원가입을 시키고자 한다.

4) 10대, 20대, 30대 별로 출력해보도록 한다. (Answer)

```
int main(void)
{

        member_info *mi;

        mi = init_member_info();
        input_info(mi);

        printf("10대만 출력\n");
        print_condition_member_info(mi, 10);

        free(mi);
}
```

→ 최종적으로 main 문 동작을 다시 한번 살펴 보자.

- member_info * mi 구조체 포인터형 mi 를 선언
- mi 구조체에 동적메모리 할당
- input_info를 통해 구조체에 정보 입력
- 10대만 출력하는 출력 함수 사용
- 동적 할당 한 함수 free

```
10대만 출력
이름: 김윤환, 나이: 10, 전화번호: 010-1111-1234
거주지: 서울시
이름: 김택용, 나이: 11, 전화번호: 010-1111-1234
거주지: 서울시
이름: 이영호, 나이: 12, 전화번호: 010-1111-1234
거주지: 서울시
이름: 조기석, 나이: 15, 전화번호: 010-1111-1234
거주지: 부산시
이름: 임요환, 나이: 16, 전화번호: 010-1111-1234
거주지: 서울시
이름: 마주작, 나이: 18, 전화번호: 010-1111-1234
거주지: 대전시
```

→ 결과물 예시

Q5)

구조체를 사용 하는 이유?

배열이 동일한 형태(int, char 같은)의 정보를 여러개 가지고 있을 때,
구조체는 다양한 변수를 가질 때 사용 합니다.

예를 들어 학생의 신상정보를 다루고 싶을 때,
학생이라는 구조체에 나이,이름, 성별을 넣는 것 입니다.

즉, 이렇게 다양한 정보를 하나의 객체에 담기 위해 구조체를 사용합니다.

→ 나의 답

구조체를 사용하는 이유는 앞선 문제에서와 같이
여러개의 데이터를 한 번에 묶어서 처리하고자 할 때 유용하다.
일일히 데이터를 모두 파라미터로 넘기는 것은 너무 불편하고 관리하기에도 힘들다.
반면 구조체로 관리하면 어떤 목적으로 사용하는지가 보다 명시적이 된다.

→ Answer

Q6)

1 6 2 3 -3 -1 -4 -7 -6 -10
위 숫자 나열에서 20번째 숫자를 구하시오.

```c
#include <stdio.h>

int arr[30] = { 1, 6, 2 };

int find_series(int num)
{
        int i;

        for (i = 3; i < num; i++)
        {
                if (i % 2)
                {
                        arr[i] = arr[i - 1] + arr[i - 3];
                        printf("홀 ");
                }
                else
                {
                        arr[i] = arr[i - 1] - arr[i - 3];
                        printf("짝 ");
                }

                printf("검토용 arr[%d] = %d\n", i, arr[i]);
        }

        return arr[num - 1];
}

int main(void)
{
        int res = find_series(20);

        printf("res = %d\n", res);
}
```

→ 우선 규칙부터 찾아 보자.

먼저 첫번째 세 항 1,6,2를 제외 하고 나면 점화식은 아래와 같다.

홀수항(배열 번호 기준) => **n = (n-1) + (n-3)**
[ex : 4번째항(배열3번)은 '3' 의 값을 가지는데
이는 3번째항의 '2' 와 1번째 항의 '1'의 합과 같다]

짝수항(배열 번호 기준) => **n = (n-1) – (n-3)**
[ex : 5번째항(배열4번)은 '-3'의 값을 가지는데
이는 4번째항의 '3' 에서 2번째항의 '6'을 뺀 값과 같다]

→ 코드해석

- Arr배열 int형의 30개의 자리를 선언
- **Find_series 함수는 원하는 항의 값 숫자를 받기 위한 함수**
  -> for문을 통해 배열의 [3]번에서 부터 값을 넣도록
  -> **홀, 짝 항의 계산 방법이 다르므로 if문을 통해 홀짝의 여부를 판단**
  (I % 2 인경우 2로 나누었을 때, 나머지가 없으면 짝수이기에 내용은 '홀 ' 로 판단
  -> **return 시에 num – 1은 배열이 0번부터 시작 되기 때문**

```
홀 검토용 arr[3] = 3
짝 검토용 arr[4] = -3
홀 검토용 arr[5] = -1
짝 검토용 arr[6] = -4
홀 검토용 arr[7] = -7
짝 검토용 arr[8] = -6
홀 검토용 arr[9] = -10
짝 검토용 arr[10] = -3
홀 검토용 arr[11] = -9
짝 검토용 arr[12] = 1
홀 검토용 arr[13] = -2
짝 검토용 arr[14] = 7
홀 검토용 arr[15] = 8
짝 검토용 arr[16] = 10
홀 검토용 arr[17] = 17
짝 검토용 arr[18] = 9
홀 검토용 arr[19] = 19
res = 19
```

→ 실행 결과 이며, 20번째 항의 값은 '19' 가 된다.

Q7)

1 ~ 10 사이의 숫자 30개를 생성하시오.

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

#define MAX                     30

int rand_arr[MAX];

void init_rand_num(void)
{
        int i;

        for (i = 0; i < MAX; i++)
        {
                rand_arr[i] = rand() % 10 + 1;
        }
}

void print_rand_arr(void)
{
        int i;

        for (i = 0; i < MAX; i++)
        {
                printf("rand_arr[%d] = %d\n", i, rand_arr[i]);
        }
}

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

        init_rand_num();

        print_rand_arr();

        return 0;
}
```

→ 코드해석

- 30개의 숫자를 생성하기 위함 이므로, 코드 가독성을 위해 MAX 값을 30으로 define
- Rand_arr[MAX] 랜덤 값 배열 생성

- Init_rand_num 함수
  -> for문을 통해서 30개의 난수를 생성하고 rand 배열에 배치 합니다.
  난수에 rand % 10 + 1 은 1~10의 숫자를 의미 합니다.

- Print_rand_arr 함수
  -> 랜덤으로 만든 숫자 배열을 **출력하는 함수이다**.

- Main 문
  -> 전체적인 동작을 살펴 보면 srand(time(NULL))을 통해 1초마다 값이 다르게 초기 설정합니다. Rand()함수는 srand 함수에 귀속되어 있기에 반드시 선언하여 주어야 합니다.
(**이 때문에 파일을 실행 할 때마다 값이 다르게 출력 될 수 있습니다**.)
  -> init 함수를 통해 랜덤 숫자를 생성합니다.
  -> print 함수를 통해 생성한 랜덤 숫자를 출력 합니다.

- 아래는 프로그램 실행 결과물 입니다.

```
rand_arr[0] = 3
rand_arr[1] = 9
rand_arr[2] = 6
rand_arr[3] = 5
rand_arr[4] = 10
rand_arr[5] = 4
rand_arr[6] = 8
rand_arr[7] = 5
rand_arr[8] = 4
rand_arr[9] = 6
rand_arr[10] = 8
rand_arr[11] = 2
rand_arr[12] = 3
rand_arr[13] = 1
rand_arr[14] = 1
rand_arr[15] = 10
```

```
rand_arr[16] = 1
rand_arr[17] = 10
rand_arr[18] = 3
rand_arr[19] = 6
rand_arr[20] = 4
rand_arr[21] = 1
rand_arr[22] = 3
rand_arr[23] = 8
rand_arr[24] = 2
rand_arr[25] = 3
rand_arr[26] = 2
rand_arr[27] = 5
rand_arr[28] = 7
rand_arr[29] = 10
```

Q8) -1

1 ~ 10 사이의 숫자를 **중복되지 않게** 10개 생성 하시오.

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

#define MAX                     10

int rand_arr[MAX];

bool dup_check(int num)
{
        int i;

        for (i = 0; i < num; i++)
        {
                if (rand_arr[i] == rand_arr[num])
                {
                        return true;
                }
        }

        return false;
}

void init_rand_num(void)
{
        int i;

        for (i = 0; i < MAX; i++)
        {
redo:
                rand_arr[i] = rand() % 10 + 1;

                if (dup_check(i))
                {
                        goto redo;
                }
        }
}
```

→ 코드해석

- 앞서 했던 1~10 까지의 숫자를 10번 생성하는 코드에 수정사항이 추가된 코드 입니다.

- 이 코드의 핵심 코드는 10개의 숫자중 중복되는 것이 없는 것이 중요 합니다.

- Dup_check 함수는 생성된 랜덤 숫자의 중복을 체크하는 함수 입니다.
  -> 인자로 num을 받아서 몇번째 숫자인지를 체크합니다.
  -> for문을 인자로 받은 num 만큼 실행 시켜줍니다.
  -> 그리고 가장 중요한 부분인 if문 조건인데
  if(rand_arr[i] == rand_arr[num]
  즉, 예를 들어 **num이 5라면 for문을 통해 배열 0~4항 까지 값이 같은지 확인하는 작업을 하는 것 입니다.**
  [만약 이렇게 확인 하는 도중 **if문 조건이 참이 되면(같은 숫자가 있으면) true 값을 반환** 합니다.
  for문을 다 돌았는대도 **if가 거짓이면(같은 숫자가 없으면) false 값을 리턴합니다**.
  참고로 함수의 반환형은 bool 입니다.

- init_rand_num 함수는 정수를 랜덤으로 생성하여 주는 함수인데 여기에서 봐야할 포인트가 있습니다.

- 앞서 중복되는 숫자를 판단하는 함수를 활용하는 부분인데,
  rand 함수를 통해 1~10의 숫자를 생성하면서
  if 만약 중복되는 값이 있어서 조건이 참이면
  goto redo : 를 실행하여 rnad() 함수를 통해 새로운 랜덤 숫자를 생성하는 것 입니다.
  여기서 **주목해야 할 부분은 goto로 rnad()함수로 넘어 갔기 때문에 fot문의 조건으로 가서 i++이 실행 되는 것은 아닙니다!!**

  생각에 만약 continue와 같은 개념을 사용 했다면 아래 코드는 실행 되지 않되 for문 조건문으로 갔을 것 입니다.

Q8) - 2

1 ~ 10 사이의 숫자를 중복되지 않게 10개 생성 하시오.

```
void print_rand_arr(void)
{
        int i;

        for (i = 0; i < MAX; i++)
        {
                printf("rand_arr[%d] = %d\n", i, rand_arr[i]);
        }
}

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

        init_rand_num();

        print_rand_arr();

        return 0;
}
```

→ 코드해석

- 앞서 함수들을 통해 rand () 랜덤 상수가 배열에 다 채워 지게 되면
  print_rand_arr 함수를 통해 배열의 값을 출력 합니다.

- Main 문
  -> srand(time(NULL)) 을 통해 난수 생성을 준비
  -> init 함수를 통해 난수를 생성하고, 중복되는 것이 없는지 확인하는 절차를 거침
  -> print 함수를 통해 마지막으로 생성된 배열들을 출력하게 됨.

```
rand_arr[0] = 10
rand_arr[1] = 9
rand_arr[2] = 6
rand_arr[3] = 7
rand_arr[4] = 3
rand_arr[5] = 4
rand_arr[6] = 8
rand_arr[7] = 1
rand_arr[8] = 2
rand_arr[9] = 5
```

→ 코드의 실행 결과물 입니다.

- 배열에 서로 겹치지 않는 숫자들이 출력 된 것을 확인 할 수 있습니다.

Q9)

주사위를 굴려서 나온 숫자를 출력 해보자.

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

int run_dice(void)
{
        return rand() % 6 + 1;
}

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

        int dice = run_dice();

        printf("dice = %d\n", dice);

        return 0;
}
```

→ 코드해석

- 주사위를 굴려 랜덤적으로 나오는 숫자를 출력해 본다.

- Run_dice 함수
  -> **rand() %6 + 1 을 통해서 1~6의 숫자를 랜덤으로 생성**합니다.

- Main 문
  -> srand 함수를 통해 난수 생성을 준비 합니다.
  -> **int형 dice 변수에 run 함수에서 생성한 난수를 리턴받아 저장** 합니다.
  -> int 형 변수의 값을 printf 함수를 통해 출력 합니다.

```
dice = 5
taein@tae
dice = 2
```

-> 좌측은 코드를 실행 하였을 때의 모습이며,
랜덤적으로 주사위 값이 생성 됩니다.

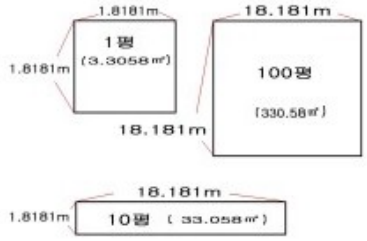
Q10) -1

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

/* 물류센터에 짐이 들어온다.
이 물류 센터 3000평 정도의 면적을 가지고 있다.
여기에 각 물건을 박스화하여 배치한다.
박스의 크기는 28평정도이며 박스간의 이격 간격은 4평이다.
물류를 배치하는 가장 효율적인 방법을 프로그래밍하여 구현하시오. */

float calc_line(int area)
{
        return sqrt(area);
}

int get_num(float total_line, float sub_line)
{
        return total_line / sub_line;
}

int main(void)
{
        // 먼저 정육면체의 길이를 구하도록 한다.
        float row, col, box_row, box_col;
        int num;

        row = col = calc_line(3000);
        box_row = box_col = calc_line(32);

        num = get_num(row, box_row);

        printf("row = col = %f\n", row);
        printf("box_row = box_col = %f\n", box_row);
        printf("3000평에 배치된 박스는 모두 %d개다.\n", num * num);

        return 0;
}
```

물류센터에 짐이 들어온다.
이 물류 센터 3000평 정도의 면적을 가지고 있다.
여기에 각 물건을 박스화 하여 배치한다.
박스의 크기는 28평정도 이며 박스간의 이격 간격은 4평이다.
물류를 배치하는 가장 효율적인 방법을 프로그래밍하여 구현하시오.

코드 해석을 하기 전에 한가지 알고 가자.
여기서 평수단위가 나오고, 이것을 이용해 계산하여 박스의 개수를 구하는 것이므로
이것 간의 관계가 어떻게 되는 지 보면

$1m^2$ = 0.3025 평
1평 = $3.3058\ m^2$
1평 = 가로 ; 1.8181 m
세로 ; 1.8181 m



(m^2) 의 가로,세로의 길이는(정사각형일경우) ; √ (루트)하면 길이가 나옴,

" 제곱미터 = m2 " 는 가로 1 m 곱하기 세로 1 m 입니다.
위와 같은 제곱미터의 면적이 3.305785 개가 모여 한평이 되어집니다.

면적의 단위인 1 평은 = 3.305785 m2 입니다.
정사각형 형태의 1 평의 한변의 길이를 구할경우에는 루트3.305785 = 1.8182 m 입니다.

즉, 1평의 한변의 길이를 구하기 위해서는 평수에 루트를 계산하여 주면 되는 것 입니다.!

Q10) -2

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

/* 물류센터에 짐이 들어온다.
이 물류 센터 3000평 정도의 면적을 가지고 있다.
여기에 각 물건을 박스화하여 배치한다.
박스의 크기는 28평정도이며 박스간의 이격 간격은 4평이다.
물류를 배치하는 가장 효율적인 방법을 프로그래밍하여 구현하시오. */

float calc_line(int area)
{
        return sqrt(area);
}

int get_num(float total_line, float sub_line)
{
        return total_line / sub_line;
}

int main(void)
{
        // 먼저 정육면체의 길이를 구하도록 한다.
        float row, col, box_row, box_col;
        int num;

        row = col = calc_line(3000);
        box_row = box_col = calc_line(32);

        num = get_num(row, box_row);

        printf("row = col = %f\n", row);
        printf("box_row = box_col = %f\n", box_row);
        printf("3000평에 배치된 박스는 모두 %d개다.\n", num * num);

        return 0;
}
```

→ 코드해석
자 이제 코드 해석을 해봅니다.

- calc_line 함수는 정육면체, **공간 평수의 길이**를 구하기 위함 입니다.
(길이는 **평수에 루트 계산**을 하여 주면 됩니다.)

- Get_num 함수는 total line 즉, 3000평 전체의 라인과
박스가 하나씩 차지 하게 될 32평의 공간이 얼마나 들어 갈지에 대한 계산을 하기 위함 입니다.

- Main 문
-> float 변수의 전체 평수의 row,col
박스 평수의 box_row, box_col 를 선언하여 줍니다.
-> 각 변수에 calc_line에서 계산한 값을 리턴을 받습니다.

-> 이 값들을 printf 함수를 통해 나타내어 주기도 하고,
최종적으로 문제에서 원했던 3000평에 배치 될 수 있는 박스가 몇 개인지를
출력하여 줍니다.

↑ 아래는 최종적으로 코드를 실행 시켰을 때 화면이며, 여기서 **주의 해야 할 점은 math 헤더의 루트 계산을 위해서는 컴파일시 마지막에 -lm 을 추가 시켜주어야 합니다**. 그리고 아래 그림과 같이 마지막에 옵션을 추가 시켜주어야 제대로 컴파일이 완성 됩니다.

```
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/first$ gcc -o an10 an10.c -lm
taein@taein-Lenovo-ideapad-700-15ISK:~/proj/es02/Lv01-02/TaeinPark/first$ ./an10
row = col = 54.772255
box_row = box_col = 5.656854
3000평에 배치된 박스는 모두 81개다.
```

Q11)

if문 어셈블리의 특성을 상세히 서술 하시오.

if문 어셈블리의 특성

if문의 경우

if(num1 > num2) 등과 같은
조건식이 들어 가므로

어셈블리어의 값을 비교 하는 cmp가 사용된다.

cmp는 보통 jump 명령어와 주로 사용하게 되는데,

위 조건식의 경우 num2가 num1 보다 작거나 같게 되면

if의 조건으로 맞지 않게 되어 다음으로 넘어가게 되는 것이다.

따라서 jle (jump less equal) 가 cmp 와 함께 사용 된다.

→ 나의 답

if 문의 경우엔 mov로 비교할 대상을 가져와서 cmp를 통해 비교를 수행한다.
이후 EFLAGS 레지스터나 비교 연산의 결과를 가지고 분기를 결정하는 형식으로 구동된다.
결국 mov, cmp, jmp 형식으로 구성된다.

→ Answer

Q12)

배열 작성시 나타나는 어셈블리의 특성을 상세히 서술 하시오.

배열 작성시 나타나는 어셈블리 특징??

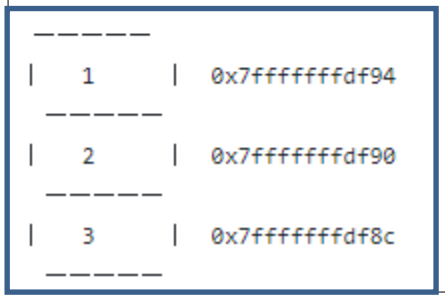
배열은 '특정한 타입의 데이터 메모리'가 배열의 크기, 갯수

만큼 생성 되기 때문에

예를 들어 int arr[] = {1, 2, 3}

이라는 배열이 있다면 각각의 값을 int형(4바이트) 공간에
배열의 값이 나란이 나열 됩니다.

구조적으로는 아래와 같은 모습 입니다.



→ 나의 답

```
   0x0000555555555184 <+27>:    movl   $0x1,-0x14(%rbp)
   0x000055555555518b <+34>:    movl   $0x2,-0x10(%rbp)
   0x0000555555555192 <+41>:    mov    -0x10(%rbp),%edx
   0x0000555555555195 <+44>:    mov    -0x14(%rbp),%eax
   0x0000555555555198 <+47>:    mov    %edx,%r8d
   0x000055555555519b <+50>:    mov    $0x1,%ecx
   0x00005555555551a0 <+55>:    mov    %eax,%edx
   0x00005555555551a2 <+57>:    mov    $0x0,%esi
   0x00005555555551a7 <+62>:    lea    0xe56(%rip),%rdi        # 0x555555556004
   0x00005555555551ae <+69>:    mov    $0x0,%eax
   0x00005555555551b3 <+74>:    callq  0x555555555070 <printf@plt>
```

배열의 시작 주소를 edx에 배치한다.
r8d의 경우엔 r8 레지스터의 32비트 표현이다.
int형 자료이므로 순차적으로 4바이트씩 공간에 배치하는 것을 볼 수 있다.
데이터 자체에 접근할 때도 edx를 기준으로 활용함을 볼 수 있다.
즉, 배열 사용시에는 dx 레지스터가 활용됨을 볼 수 있다.

→ Answer

Q13)

반복문 작성시 나타나는 어셈블리의 특성을 상세히 서술 하시오.

반복문 작성시 나타나는 어셈의 특징

반복문 for 를 사용 했을 때 어셈블리의 특징이라고 하면

예를 들어 for(i=0 i<10; i++) 의 경우

반복문 조건의 i가 1씩 증가되면서 10보다 작을 경우 참이 되어 반복문의 내용을

실행하게 되는데

이것은 어셈으로 분석하여 보면

처음에 i 값이 한 스택 메모리에 할당이 되고

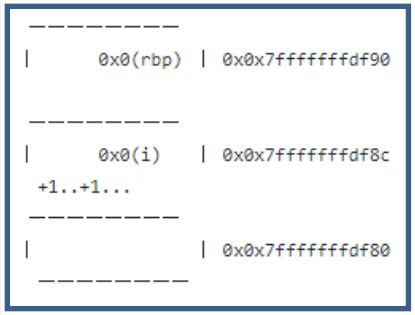
if문과 마찬가지로 조건을 cmp 비교 한 뒤

참이면 for 문 안의 스택 주소로 jump 하고,

반복이 계속 진행 되어 조건이 거짓이 되면 for 문을 빠져나가

어셈블리 명령어 상에서 jump 되지 않고

바로 아래 줄로 실행 되는 절차를 가지게 됩니다.



그림으로 보면 for문 조건문의 i 값이 다음 조건 비교로 넘어 가기전
+1 씩 되는 것 입니다.

→ 나의 답

반복문 작성시 if 문과 마찬가지로 mov, cmp, jmp로 구성된다.
다만 반복을 하기 위해 다시 초기 위치로 되돌리는 jmp와
반복을 탈출하기 위한 jmp로 구성된다.
if와의 차이점이라면 이와 같이 jmp가 두 개 존재한다는 것이다.

→ Answer

Q14)

함수를 호출할 때 발생하는 어셈블리의 특성을 상세히 서술 하시오.

함수 호출 할 때 발생하는 어셈블리 특성??

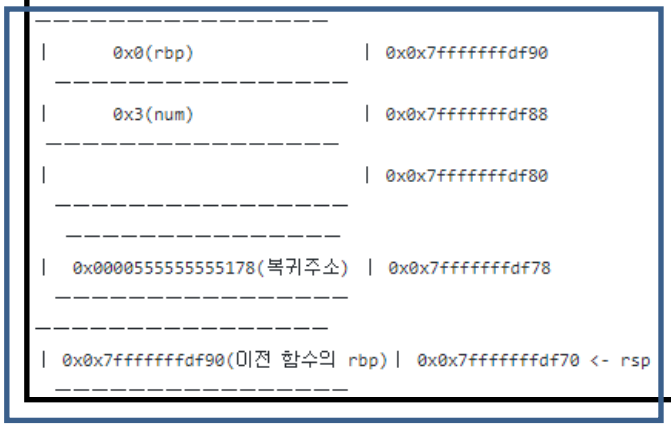
함수 호출에서 가장 중요하다고 생각 되는 놈은 역시

callq 인 것 같다.

callq는 함수 호출을 하기전에 전 스택의 rsp-8 위치에
복귀주소(호출 했던 함수에서 돌아 올 때 넘어가야 할 다음 메모리 주소)
를 push 한 뒤 jump 하는 명령어 이다.

그리고 돌아올 이전 함수의 rbp가 복귀주소의 아래에 저장된다.

예를 들면 아래와 같음 모습.



아직까지는 rbp가 바뀌지 않았다.

그러나, 새로운 함수를 호출하게 되면
rsp -> rbp, 즉, 새로운 rbp가 생성된다. 이것이 새로운 함수의 스택 기준점이 된다.
그리고 새로운 함수의 메모리가 새롭게 할당 되어 새로운 스택을 (아래로)쌓게 된다.
이 곳만의 rbp, rsp가 새로이 생성 되는 것이다.

다시 원래의 함수로 돌아올 때에는
pop %rbp 명령어가 사용 되어
rsp에서 값을 빼서 rbp로 값을 넘겨 준다.
리턴 값이 이런식으로 넘어 간다.

빼내는 방식은 rsp를 위로 올라 가게 만들고
70 -> 78로 가게 한다.

그리고 retq 가 실행 되면
retq는 pop rip 이라는 의미로 rsp에서 값을 빼서 rip에 주는 것이다.

rip은 다음에 실행하게 될 명령이므로
rip에 복귀 주소가 들어가게 됨으로써 저장 되어 있던
복귀 주소로 함수는 복귀 하게 된다.

→ 나의 답

함수를 호출할 때 나타나는 특성은 call이다.
이 call의 경우엔 push + jmp를 수행하므로
push에서 Stack에 복귀 주소를 저장하며 jmp에서는 특정 주소로 이동을 하게 된다.
뿐만 아니라 함수 호출시에는 반드시 스택 프레임을 형성하기 위해
push와 mov가 함께 세트로 움직인다.
또한 마지막에 스택을 해제하기 위해 pop과 ret가 동작하게 되어 있다.

→ Answer

Q15)

프로그래머가 반드시 알아야 하는 가상메모리 섹션 4가지를 서술하고
각각의 섹션 4가지에 대해 상세히 기술 하시오.

스택(Stack): 지역 변수가 배치된다.
힙(Heap): malloc, calloc등의 동적할당된 데이터가 배치된다.
데이터(Data): 전역 변수 및 static 변수가 배치된다.
텍스트(Text): 기계어 및 함수가 배치된다.

Q16)

배열에 대문자로 작성된 문장을 소문자로 변경하여 출력해보세요.

```c
#include <stdio.h>

void conversion_big_to_small(char *str)
{
        int i;

        for (i = 0; str[i]; i++)
        {
                if (str[i] > 64 && str[i] < 92)
                {
                        str[i] ^= 0x20;
                }
        }
}

int main(void)
{
        char str[] = "WhErE ArE YOu FROM ?";

        printf("default: %s\n", str);

        conversion_big_to_small(str);

        printf("conversion: %s\n", str);
}
```

→ 코드해석

- Conversion 함수는 대문자를 소문자로 바꾸어 주는 함수를 작성 한 것이다.
  -> 이 함수에서 **살펴볼 점은 for 의 조건문 표현 방식**이다.

=> str[i]가 조건문인데 이것의 의미는
만약에 i 가 0 이면, str[0]은 W 이고,
i++이 진행되다가 **str[마지막]이 문장이므로 NULL을 반환해서 for문 동작이 멈추게 된다**고 보면 된다!

- If 조건문의 경우 str[i] 의 값이 아스키 값이 64 ~ 92 가 영어 대문자의 값이므로
  대문자 일 경우 조건에 맞게 되어 그 값에 0x20을 XOR 계산 하면 대소문자가 변경이 됩니다.
  (반대로 소문자의 값을 받고 0x20 을 XOR 계산해도 대문자 값이 출력됨)

- Main 문
  -> char 형 배열로 대소문자가 섞인 문장을 저장.
  -> 변환 전의 문장을 출력
  -> conversion 함수를 통해 대문자를 소문자로 변환
  -> 전부 소문자로 변경된 문장을 출력.

```
taein@taein-Lenovo-ideapad-700-15IS
default: WhErE ArE YOu FROM ?
conversion: where are you from ?
```

<- 좌측은 프로그램 실행 했을 때 모습이며, 대문자가 모두 소문자로 변환 된 것을 볼 수 있다.

Q17)

프로그래머가 반드시 알아야 하는 가상메모리 섹션 4가지를 서술하고
각각의 섹션 4가지에 대해 상세히 기술 하시오.

```
ls: 현재 디렉토리 위치에서 파일 리스트를 보는 것
pwd: 현재 디렉토리 위치 보기
mkdir: 디렉토리 만들기
cp: 복사
mv: 이름 바꾸기 및 위치 옮기기
rm: 삭제
gcc: 컴파일러
vi: 편집기
```

Q18) -1

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

typedef struct _employee emp;

struct _employee
{
        char name[32];
        int pay;
};

char name[5][32] = { "이경수", "박지완", "강지훈", "김지환", "조민현" };

emp *init_employee(void)
{
        int i;

        emp *tmp = (emp *)malloc(sizeof(emp) * 5);

        for (i = 0; i < 5; i++)
        {
                strcpy(tmp[i].name, name[i]);
                tmp[i].pay = rand() % 501 + 3000;
        }

        return tmp;
}
```

사원이 5명 있다. (이름은 적당히 작성한다)
5명의 사원의 초봉은 3000 ~ 3500이다.
이들에게 연마다 1 ~ 10%의 랜덤한 임금 상승폭을 적용한다.
10년후 가장 연봉이 높은 사원의 임금과 이름을 출력하시오.

→ 코드해석

- 직원정보를 입력하기 위해 employee 라는 구조체를 만들고, typedef으로 emp로 명칭 설정.
- **이중 배열을 사용해 5명의 정보를 32byte씩 메모리 공간 확보**

- Init_employee 함수는 **malloc 함수를 통해 메모리를 동적할당** 하고,
  이름 정보는 strcpy 함수를 사용해 각 이름 배열에 저장한다.
  연봉은 숫자 값이기에 바로 적용 가능 한데,
  여기서 볼 부분은 **rand() % 501 : 0 ~ 500 의 숫자**에
  +3000을 한 부분이다. 이는 **기본적으로 3000의 초봉에 차이가 0 ~ 500 이 날수 있음을 의미** 한다.

- 그리고 마지막으로 동적 할당한 메모리에 정보가 들어간 tmp 구조체를
  구조체 포인터 형으로 반환한다.

Q18) -2

```c
void print_employee(emp *e)
{
        int i;

        for (i = 0; i < 5; i++)
        {
                printf("name = %s, pay = %d\n", e[i].name, e[i].pay);
        }
}

void years_ago(emp *e, int year)
{
        int i, j;

        for (i = 0; i < year; i++)
        {
                printf("%d 년차\n", i + 1);

                for (j = 0; j < 5; j++)
                {
                        e[j].pay = e[j].pay + e[j].pay * (((rand() % 10) + 1) / 100.0);
                }

                print_employee(e);
        }
}
```

→ 코드해석

- Print_employee 함수는 직원의 이름과 페이를 출력해 주는 함수 이다.

- Years ago 함수는
  년차 만큼 for문을 돌리는데, for문이 돌아가면서, 이율 (1~10%) 만큼을
  증액 시켜 주게 된다.

  여기에서 핵심 코드의 의미는
  어떤 직원의 임금에서 랜덤하게 1~10% 이율이 정해지고
  그 **이율만큼 원금에 곱해진 금액이 이전 금액에 합쳐져**
  증액된 연봉으로 측정되는 것이다.

Q18) -3

```
emp find_max(emp *e)
{
        int i, max_idx, max = 0;

        for (i = 0; i < 5; i++)
        {
                if (max < e[i].pay)
                {
                        max = e[i].pay;
                        max_idx = i;
                }
        }

        return e[max_idx];
}

int main(void)
{
        emp max;

        srand(time(NULL));

        emp *e = init_employee();

        print_employee(e);

        years_ago(e, 10);

        max = find_max(e);
        printf("10년후 가장 연봉이 높은 사원은 %s, %d\n", max.name, max.pay);

        return 0;
}
```

→ 코드해석

- Find_max 함수는 직원들 정보의 구조체 인자를 받아서
  직원들 중 최고의 연봉을 찾아 내는 것이다.

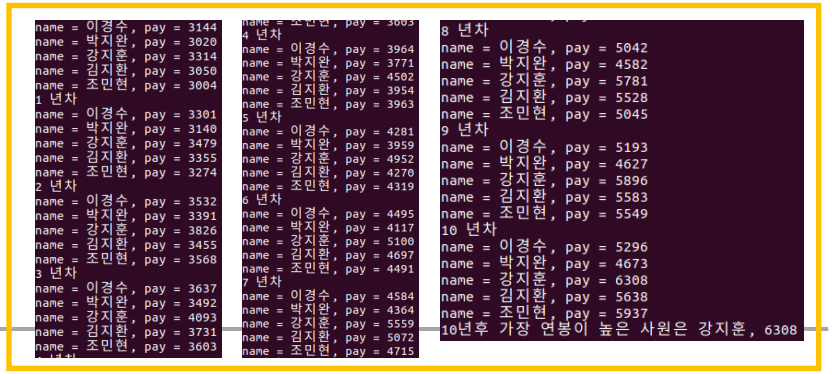
  5명의 직원이므로 for문을 통해 5번 반복 하고,
  if문의 조건문이 핵심이다.

  max 값으로 선정된 금액 보다 새로운 금액이 더 크다면 if 조건문이 참이다.
  max 값은 초기에 0 이기 때문에 한번은 참이 되게 되어 있다.
  그리고 max 값을 초기화 해놓지 않으면 어떤 값이 **max 변수에 들어 있을지 알 수 없기에 반드시 초기화 시켜주어야 한다.**

  여튼, max값이 설정이 되면 **max_idx 변수에 i 값이 몇 번째 값인지** 들어가게 되고, 이 값을 구조체의 인덱스로 사용해서
  누구의 연봉이 제일 높은 것인지 알 수 있게 된다.

- Main 문
  -> 직원 구조체 max 를 선언.
  -> 난수 사용을 위한 설정
  -> init 함수를 통해 동적 메모리 할당 하고 기본적인 연봉 측정
  -> yeas_ago를 통해 10년간 랜덤 적인 이율을 적용
  -> find_max 함수를 통해 최고의 연봉과 사람을 찾고
  그 구조체 값을 max로 반환
  -> printf로 연봉이 가장 높은 사원이 누구인지, 얼마인지를 출력.

프로그램
실행결과 →



Q19) -1

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

typedef struct _employee emp;

struct _employee
{
        char name[32];
        int pay;
};

typedef struct _statistics stat;

struct _statistics
{
        float mean;
        float std_dev;
};

char name[5][32] = { "이경수", "박지완", "강지훈", "김지환", "조민현" };

emp *init_employee(void)
{
        int i;

        emp *tmp = (emp *)malloc(sizeof(emp) * 5);

        for (i = 0; i < 5; i++)
        {
                strcpy(tmp[i].name, name[i]);
                tmp[i].pay = rand() % 501 + 3000;
        }

        return tmp;
}
```

사원이 5명 있다. (이름은 적당히 작성한다)
5명의 사원의 초봉은 3000 ~ 3500이다.
이들에게 연마다 1 ~ 10%의 랜덤한 임금 상승폭을 적용한다
5명의 연도별 평균과 표준 편차를 구하시오.

→ 문제해석 및 코드해석

- 이번에는 18번 문제에 추가로 응용해서 5명의 연도별 평균과 표준편차를 구하는 것이다.

  옆의 코드는 앞서 설명한 바와 같고 추가된 부분만 설명하자면

  statistics 통계라는 구조체가 만들어져 있고,

  내용에는 float 형으로 mean(평균) 과 std_dev(표준편차)가 구성되어 있다.

  여기서 잠깐 표준편차 구하는 방법이 생각이 안남으로 아래를 참고합니다.

- **표준편차**란?
  => **자료의 값이 평균으로부터 얼마나 떨어져 있는지, 흩어져 있는지 나타내는 값**.
  -> 편차를 구하려면 일단 평균값 부터 살펴 봐야 합니다.

$$V(X) = E(x^2) - \{E(X)\}^2$$

분산 제곱의평균 평균의제곱

$$\sqrt{\text{(분산)}}$$

  -> 분산 값의 루트 계산을 하면 표준편차 값이 나오는데,
  이 분산 값은 **[제곱의 평균 - 평균의 제곱]** 값 입니다.

Q19) -2

```c
void print_employee(emp *e)
{
        int i;

        for (i = 0; i < 5; i++)
        {
                printf("name = %s, pay = %d\n", e[i].name, e[i].pay);
        }
}

float calc_mean(emp *e, int num)
{
        int i;
        float sum = 0;

        for (i = 0; i < num; i++)
        {
                sum += e[i].pay;
        }

        return sum / (float)num;
}

float calc_std_dev(emp *e, int num, float mean)
{
        int i;
        float sum = 0;

        for (i = 0; i < num; i++)
        {
                sum += pow((e[i].pay - mean), 2);
        }

        return sum / num;
}
```

→ 코드해석

- print_employee 함수는 5명의 직원들의 이름과 연봉을 출력하여 주는 함수ㅋㅋㅌ.

- **Clac_mean함수는 평균을 계산하는 함수**입니다.
  평균을 구하는 방법은 흔히 아는 방식인 개체들(연봉들)의 합을 더하고
  더한 항의 개수 n 만큼 나누면 되는 방식입니다.
  -> 여기 함수에서는 인자로 구조체에 직원의 정보가 있으므로 구조체로 정보를 받고
  개체수인 num을 입력 받아 그만큼
  for문을 돌려서 float sum 변수에 총 합을 구한뒤
  총 합 나누기 num의 값을 리턴 하여 줍니다.

- Calc_std_dev는 표준편차를 계산하여 주는 함수 입니다.
  인자로 sum 값을 구할 float sum을 선언하여 주고,

  여기서는 pow 라는 C언어 함수가 사용 되었습니다.'

```c
res = pow( 2.0, 3.0 );
printf(" 2 ^ 3 = %.0f \n", res );
```

  위는 pow 함수를 사용한 한 예로 pow(2,3)이면 2 ^ 3 = 8 의 값이 출력 되는 것입니다.

  따라서 **우리 코드의 pow(e[i].pay – mean), 2);**
  는 구조체의 금액 값에 평균을 빼고(평균과의 차이 겠죠) 그것을 제곱 하여 준 것 입니다.

  이것을 각각 구조체 마다(직원마다) 의 값을 sum에 차곡차곡 더하게 됩니다.

  최종적으로 이 sum의 값을 직원 수 num 만큼 나눈 값을 retrun 한 것이

  '표준편차' 값을 구하는 것이 이용되게 됩니다.

Q19) -3

```c
void record_statistics(emp *e, int idx, stat *s)
{
        int i;
        float mean = calc_mean(e, 5);

        s[idx].mean = mean;
        s[idx].std_dev = sqrt(calc_std_dev(e, 5, mean));
}

void print_statistics(stat *s, int idx)
{
        printf("평균 = %f, 표준편차 = %f\n", s[idx].mean, s[idx].std_dev);
}

void years_ago(emp *e, int year, stat *s)
{
        int i, j;

        for (i = 0; i < year; i++)
        {
                printf("%d 년차\n", i + 1);

                for (j = 0; j < 5; j++)
                {
                        e[j].pay = e[j].pay + e[j].pay * (((rand() % 10) + 1) / 100.0);
                }

                record_statistics(e, i, s);

                print_employee(e);
                print_statistics(s, i);
        }
}
```

→ 코드해석

- 앞선 18번 문제의 코드와 달라진 부분은
  통계치를 record 기록하는 함수가 생겼고,

  record_statistics 함수를 통해서 평균과 표준편차를 계산합니다.

- 그리고 계산된 통계를 print_statistics 함수를 통해 출력하게 됩니다.

- 연간 이율을 추가 연봉에 더했던 years_ago 함수에는
  위에서 만든 통계를 구하는 함수를 활용해서
  계산을 하고 출력합니다.

Q19) -4

```c
emp find_max(emp *e)
{
        int i, max_idx, max = 0;

        for (i = 0; i < 5; i++)
        {
                if (max < e[i].pay)
                {
                        max = e[i].pay;
                        max_idx = i;
                }
        }

        return e[max_idx];
}

stat *init_statistics(void)
{
        stat *tmp = (stat *)malloc(sizeof(stat) * 10);

        return tmp;
}

int main(void)
{
        stat *s;
        emp max;

        srand(time(NULL));

        s = init_statistics();
        emp *e = init_employee();

        print_employee(e);

        years_ago(e, 10, s);

        max = find_max(e);
        printf("10년후 가장 연봉이 높은 사원은 %s, %d\n", max.name, max.pay);

        return 0;
}
```

→ 코드해석

- 이부분은 앞선 18번 문제의 코드와 크게 달라진 부분이라고 하면
  통계계산을 위한 동적 메모리 할당 함수 init_stats..가 추가 되었다는 것입니다.

- Main 문
  -> 통계, 직원의 구조체를 가르킬 구조체 포인터 변수 선언
  -> 난수 활용을 설정 하기 위한 srand
  -> s : 통계 구조체 주소 값을 리턴 받을 변수
  -> emp : 직원들의 이름 및 연봉 정보를 리턴 받을 변수
  -> 이율 만큼 연봉 상승 적용 (10년치), 그리고 10년치 평균과 표준편차 계산 및 출력
  -> 최대 연봉자 찾기
  -> 가장 높은 연봉 자의 이름 및 금액 출력.

Q19) -5

컴파일

```
$ gcc -o an19 an19.c -lm
$ ./an19
```

→ gcc 컴파일 과정
math.h 의 루트 계산을 하기 때문에
여기서도 마찬가지로 컴파일시에 –lm 옵션을
추가해야 합니다.

실행 결과

```
name = 이경수, pay = 3472
name = 박지완, pay = 3361
name = 강지훈, pay = 3496
name = 김지환, pay = 3477
name = 조민현, pay = 3341
1 년차
name = 이경수, pay = 3506
name = 박지완, pay = 3394
name = 강지훈, pay = 3530
name = 김지환, pay = 3824
name = 조민현, pay = 3608
평균 = 3572.399902, 표준편차 = 143.250259
2 년차
name = 이경수, pay = 3751
name = 박지완, pay = 3631
name = 강지훈, pay = 3777
name = 김지환, pay = 4015
name = 조민현, pay = 3896
평균 = 3814.000000, 표준편차 = 131.112167
3 년차
name = 이경수, pay = 3938
name = 박지완, pay = 3994
name = 강지훈, pay = 3965
name = 김지환, pay = 4336
name = 조민현, pay = 3934
평균 = 4033.399902, 표준편차 = 152.826172
4 년차
name = 이경수, pay = 3977
name = 박지완, pay = 4233
name = 강지훈, pay = 4361
name = 김지환, pay = 4596
name = 조민현, pay = 4170
평균 = 4267.399902, 표준편차 = 205.786880
5 년차
name = 이경수, pay = 4215
name = 박지완, pay = 4486
name = 강지훈, pay = 4491
name = 김지환, pay = 5055
name = 조민현, pay = 4587
평균 = 4566.799805, 표준편차 = 273.782684
6 년차
name = 이경수, pay = 4552
name = 박지완, pay = 4575
name = 강지훈, pay = 4625
name = 김지환, pay = 5105
name = 조민현, pay = 4770
평균 = 4725.399902, 표준편차 = 204.378662
7 년차
name = 이경수, pay = 4597
name = 박지완, pay = 4620
name = 강지훈, pay = 4856
name = 김지환, pay = 5156
name = 조민현, pay = 4817
평균 = 4809.200195, 표준편차 = 201.666458
8 년차
name = 이경수, pay = 4780
name = 박지완, pay = 5035
name = 강지훈, pay = 5293
name = 김지환, pay = 5671
name = 조민현, pay = 4961
평균 = 5148.000000, 표준편차 = 309.100647
9 년차
name = 이경수, pay = 5162
name = 박지완, pay = 5387
name = 강지훈, pay = 5451
name = 김지환, pay = 5784
name = 조민현, pay = 5159
평균 = 5388.600098, 표준편차 = 229.913559
10 년차
name = 이경수, pay = 5420
name = 박지완, pay = 5494
name = 강지훈, pay = 5505
name = 김지환, pay = 6015
name = 조민현, pay = 5623
평균 = 5611.399902, 표준편차 = 212.021317
10년후 가장 연봉이 높은 사원은 김지환, 6015
```

질문) 1

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

void input_info(char *name, int *age, char *phone, char *city, char *street, char *detail)
{
        printf("회원 정보를 입력해주세요.\n이름: ");
        scanf("%s", name);

        printf("나이: ");
        scanf("%d", age);

        printf("전화 번호: ");
        scanf("%s", phone);

        printf("도시: ");
        scanf("%s", city);

        printf("도로명: ");
        scanf("%s", street);

        printf("상세 주소: ");
        scanf("%s", detail);
}
```

여기에서 retrun name 이라던가
Return age 라던가를 하지 않아도
각각의 변수의 값이 main으로 넘어 갈
때
반환 되는 이유가 있나요??

질문) 2

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

typedef struct _member_info member_info;

struct _member_info
{
        int age;
        char name[32];
        char phone[32];
        char city[32];
        char street[128];
        char detail[128];
};

int total_member;

member_info *init_member_info(void)
{
        member_info *tmp = (member_info *)malloc(sizeof(member_info) * 64);
}
```

여기에서 malloc을 사용 할 때
tmp를 따로 선언하지 않은 이유는
Member info 포인터형인 형태인 tmp가 생성과 동시에 malloc이
실행 되기 때문 인가요??

그리고 sizeof 뒤에 x 64를 한 이유가 있을 까요??

질문) 3

```c
void print_employee(emp *e)
{
        int i;

        for (i = 0; i < 5; i++)
        {
                printf("name = %s, pay = %d\n", e[i].name, e[i].pay);
        }
}

float calc_mean(emp *e, int num)
{
        int i;
        float sum = 0;

        for (i = 0; i < num; i++)
        {
                sum += e[i].pay;
        }

        return sum / (float)num;
}

float calc_std_dev(emp *e, int num, float mean)
{
        int i;
        float sum = 0;

        for (i = 0; i < num; i++)
        {
                sum += pow((e[i].pay - mean), 2);
        }

        return sum / num;
}
```

여기에서 num을 int형 인자로 받았는데

Return 할 때는 float 형으로 하는 이유가 있을까요???