

REPORT

전자공학도의 윤리 강령 (IEEE Code of Ethics)

(출처: <http://www.ieee.org>)

나는 전자공학도로서, 전자공학이 전 세계 인류의 삶에 끼치는 심대한 영향을 인식하여 우리의 직업, 동료와 사회에 대한 나의 의무를 짐에 있어 최고의 윤리적, 전문적 행위를 수행할 것을 다짐하면서, 다음에 동의한다.

1. **공중의 안전, 건강 복리에 대한 책임:** 공중의 안전, 건강, 복리에 부합하는 결정을 할 책임을 질 것이며, 공중 또는 환경을 위협할 수 있는 요인을 신속히 공개한다.
2. **지위 남용 배제:** 실존하거나 예기되는 이해 상충을 가능한 한 피하며, 실제로 이해가 상충할 때에는 이를 이해 관련 당사자에게 알린다. (이해 상충: conflicts of interest, 공적인 지위를 사적 이익에 남용할 가능성)
3. **정직성:** 청구 또는 견적을 함에 있어 입수 가능한 자료에 근거하여 정직하고 현실적으로 한다.
4. **뇌물 수수 금지:** 어떠한 형태의 뇌물도 거절한다.
5. **기술의 영향력 이해:** 기술과 기술의 적절한 응용 및 잠재적 영향에 대한 이해를 높인다.
6. **자기계발 및 책무성:** 기술적 능력을 유지, 증진하며, 훈련 또는 경험을 통하여 자격이 있는 경우이거나 관련 한계를 전부 밝힌 뒤에만 타인을 위한 기술 업무를 수행한다.
7. **엔지니어로서의 자세:** 기술상의 업무에 대한 솔직한 비평을 구하고, 수용하고, 제공하며, 오류를 인정하고 수정하며, 타인의 기여를 적절히 인정한다.
8. **차별 안하기:** 인종, 종교, 성별, 장애, 연령, 출신국 등의 요인에 관계없이 모든 사람을 공평하게 대한다.
9. **도덕성:** 허위 또는 악의적인 행위로 타인, 타인의 재산, 명예, 또는 취업에 해를 끼치지 않는다.
10. **동료애:** 동료와 협력자가 전문분야에서 발전하도록 도우며, 이 윤리 헌장을 준수하도록 지원한다.

위 IEEE 윤리헌장 정신에 입각하여 report를 작성하였음을 서약합니다.

학 부: 전자공학부

마감일: 2021. 04. 07.

과목명: 전자공학프로그래밍

교수명: 이정원 교수님

학 번: 201620837

성 명: 안민규

1. 문제 분석

1) Memory Allocation

memory allocation이란 메모리를 할당하는 것을 뜻하는데, 이는 프로그래밍 도중 혹은 실행 시간 동안 필요한 메모리 공간을 할당하는 것이다. 프로그래밍 도중에 할당하는 것은 정적 메모리할당이라 하고, 실행 시간에 할당하는 것은 동적 메모리할당이라고 한다. 선언되는 변수마다 할당하는 메모리의 양이 달라지는 것을 알 수 있다. 예를 들어, char형 변수는 1, int형 변수는 4의 메모리를 할당받는 것이다. 또한, int형의 3칸의 배열이라면 $4 \times 3 = 12$ 총 12의 메모리를 할당받는 것을 알 수 있다. 이번 과제는 실제로 할당되는 메모리를 보는 것이 아닌 memory_address라는 스택을 만들어 이를 통해 메모리가 할당되는 것을 눈으로 확인하기 위한 과제이다. 또한 스택에 저장한 값을 variable_table이라는 배열(실제로는 표)에 저장하여 프린트하는 것을 보는 것이다.

2) Stack

스택이란 제한적으로 접근할 수 있는 나열구조를 말하며, 이는 탑처럼 쌓인 모양을 띄며 이 모양을 따 스택이라 부른다.



	type	name	value
0	char	a1	
1~4	int	b1	
5~9	float	c1	

실습 variable_table

	type	name	array_size
0~3	char	a1	[3]
4~19	int	b1	[4]

과제 variable_table, array_size

2. 함수의 기능 및 코드 분석

2-1) updateMemory 함수

```
void updateMemory(int menu) {
    void (*function_pointer)(); //포인터함수 선언

    if (menu == 2) {
        function_pointer = viewMemory;
        function_pointer(); //선택된 메뉴가 2라면 할당된 메모리를 보기 위해 viewmemory 함수
동작
    }
    else if (menu == 3) {
        function_pointer = allocateVariable;
        function_pointer(); //선택된 메뉴가 3라면 변수를 할당하기 위해 allocatevariable 함수동작
    }
    else if (menu == 4) {
        function_pointer = deleteVariable;
        function_pointer(); //선택된 메뉴가 4라면 할당된 변수를 지우기 위해 deletevariable 함수
동작
    }
}
```

위 함수는 updateMemory 함수로써 사용자가 어떤 동작을 할지 입력한 값에 따라 동작을 진행하는 함수이다. 2는 할당된 메모리 보기, 3은 변수 할당하기, 4는 할당된 변수를 지우는 동작을 한다. 메뉴에 맞는 함수를 동작하고 난 다음 그의 리턴값을 function_pointer의 주소에 값을 저장

2-2) viewMemory 함수

```
void viewMemory() {
    printf("memory address\n-----\n");
    for (int n = 0; n < MEMORY_SIZE; n++) { //0부터 memory_size까지 탐색
        printf("%3d |", n);
        if (memory_address[n] != 0) { //memory_address[n]이 0이 아닐때까지 진행 -> 즉, 메모
리가 할당되어 있음
            printf(" } ");
            if (memory_address[n] == 2) //memory_address[n]이 2라면 변수를 처음 할당한
위치이므로 변수의 이름과 종류 배열의 사이즈를 보여줌
                printf("%s %s[%d]", variable_table[n][1], variable_table[n][2],
```

```

array_size[n][1]); //메모리가 들어있는 n을 찾아냈다면, 그 때 variable_table의 n번째 1,2. 즉, 할당된 변수의
종류와 이름을 출력/또한 array_size의 n번째에 저장된 숫자를 붙여줌
    }
    printf("\n");
}
printf("-----\n\n\n");
}

```

위 함수는 viewMemory함수로서 현재 할당되어 있는 변수의 상황을 보여주는 함수이다. 변수가 할당되어 있지 않는 스택은 0, 되어 있는 스택은 1, 처음 변수를 받은 스택은 2로 표현함으로서 위치에 맞게 문구가 프린트된다. 다른 함수들과는 다르게 function_pointer를 호출하지 않는다.

2-3) allocateVariable 함수

```

void allocateVariable() {
    void (*function_pointer)();
    int variable_size = 0, size = 0; //변수의 크기와 배열의 크기를 받기 위한 int형 변수선언
    char variable_type[VARIABLE_LENGTH], variable_name[VARIABLE_LENGTH]; //변수의 이름과
타입을 입력받기 위해 char형 변수 선언

    printf("variable_type : ");
    scanf("%s", variable_type);
    printf("variable_name : ");
    scanf("%s", variable_name);
    printf("array_size : ");
    scanf("%d", &array_size[array_pointer]); //변수의 이름, 종류, 배열 수를 입력받음. 배열 수는 먼
저 전역변수로 선언된 array_size[] stack에 저장

    size = array_size[array_pointer]; //배열에 저장된 값을 따로 size에 저장

    strcpy(variable_table[stack_pointer][1], variable_type);
    strcpy(variable_table[stack_pointer][2], variable_name); //입력받은 변수의 이름과 타입을
variable_table의 stack_pointer층 1,2칸에 저장

    if (strcmp(variable_type, "char") == 0)
        variable_size = 1;
    else if (strcmp(variable_type, "int") == 0)
        variable_size = 4;
    else if (strcmp(variable_type, "float") == 0)
        variable_size = 4;
    else if (strcmp(variable_type, "double") == 0)
        variable_size = 8; //입력받은 variable_type과 문자열을 비교하여 변수의 종류에 맞추어
variable_size에 값을 저장

    if (variable_size * size > MEMORY_SIZE)
    {
        printf("Variable size is too large.");
    }
}

```

```

        exit(0);
    } //예외처리 (메모리할당 불가), 입력받은 변수의 size와 배열의 크기의 곱이 스택의 사이즈
memory_size보다 크다면 할당불가 프린트하고 프로그램 종료

    if (MEMORY_SIZE - stack_pointer < (variable_size) * (array_size[array_pointer]))
    {
        for (int i = 0; i < MEMORY_SIZE; i++)
        {
            *variable_table[i][1] = 0;
            *variable_table[i][2] = 0;
            array_size[i] = 0;
            memory_address[i] = 0;
        } //이미 들어가있는 variable_table, array_size, memory_address를 0으로 초기화
        stack_pointer = 0;
        array_pointer = 0; //새로 시작해야하므로 stack_pointer와 array_pointer를 0으로 초기
화

        strcpy(variable_table[stack_pointer][1], variable_type);
        strcpy(variable_table[stack_pointer][2], variable_name); //초기화 전에 입력된 문자열
을 다시 복사해서 variable_table에 삽입. 이 때 stack_pointer가 0이므로 0층부터 다시 저장
        array_size[array_pointer] = size; //size에 저장해놓은 값을 초기화된 array_pointer층
의 array_size에 다시 저장. 이를 위해 size에 미리 값을 저장.
    } //예외처리 (자리 안 남았을 때), 입력한 변수의 할당량보다 메모리가 적게 남았을 경우, 초기화 후
0층부터 다시 진행.

    memory_address[stack_pointer] = 2; //stack_pointer층의 memory stack에는 2를 저장. 입력받
은 층이기 때문에
    for (int n = 1; n < variable_size; n++)
        memory_address[stack_pointer + n] = 1; //입력받은 변수의 크기만큼 1을 저장.

    stack_pointer += variable_size; //stack_pointer를 변수의 크기만큼 증가.

    for (int i = 1; i < size; i++)
    {
        for (int n = 0; n < variable_size; n++)
            memory_address[stack_pointer + n] = 1;
        stack_pointer += variable_size;
    } //입력된 배열의 크기만큼 1을 채워주는 반복문.

    array_pointer = stack_pointer; //array_pinter를 stack_pointer의 값과 같게 만들. 메모리를 저
장하는 memory_address와 배열의 값을 저장하는 array_size가 같은 층에 있게하기 위함

    function_pointer = viewMemory;
    function_pointer(); //viewmemory 함수 호출
}

```

위 함수는 allocatevariable 함수로 변수를 할당하는 역할을 하는 함수이다. 할당할 변수의 종류, 이름, 배열의 사이즈를 입력받은 후, 종류와 이름을 variable_table에 저장한다. 배열의 사이즈의 경우, 전역변

수로 선언된 array_size 배열에 직접 입력을 받는다. array_pointer 위치에 저장한다. array_pointer는 stack_pointer와 같은 위치에 있게끔 설정하였다. 그 후, 입력받은 변수의 종류에 따라 variable_size 값을 정한다.

입력받은 array_size를 미리 저장하기 위해 size 변수를 따로 선언하였고, variable_table을 초기화하기 위해 따로 clear_type, clear_name을 선언하였다. 이는 입력을 받는 variable_type, variable_name과 같은 길이를 받는 char 변수로 선언하였다.

할당하려는 메모리의 크기(변수의 크기*배열의 크기)가 stack 사이즈 100보다 큰 경우 메모리를 할당할 수 없으므로 변수의 사이즈가 너무 크다고 프린트하고 프로그램을 종료하게 설정하였다. (ex)double의 경우 변수의 크기가 8이고 배열의 크기가 13이라면 104이므로 100보다 크기에 할당 불가)

다음은, 메모리가 100까지 차지 않은 상태에서 변수를 할당할 때 스택과 variable_table을 모두 지우고 0부터 다시 탐색하여 메모리를 할당하는 부분이다. if문의 조건으로는 memory_size-stack_pointer < variable_size*array_size이다. 즉, (전체 메모리의 크기 - 현재 쌓인 스택의 높이)가 할당하려는 (변수의 크기*배열의 사이즈)일 때 실행된다. 스택에 할당할 수 있는 남은 메모리의 수가 새로 할당하려는 메모리의 양보다 적을 때 작동하는 것이다. 가장 먼저, variable_table과 메모리 스택, 배열 스택을 비우는 작업이 선행되어야 하므로, 미리 0으로 선언한 clear_type, name의 문자열을 variable_table[i][1], [2]에 복사하고 또한 array_size[i]와 memory_address[i]를 0으로 초기화한다. 이를 주어진 memory_size만큼 반복하여 모든 메모리 스택과 배열 스택, variable_table을 비운다. 이미 할당되었던 값들을 다 지우고 나면 다시 0부터 할당해야하므로 메모리 스택과 배열 스택의 위치를 나타내는 stack,array_pointer를 0으로 초기화한 후 마지막으로 입력받은 변수의 종류와 이름을 다시 variable_table[stack_pointer][1],[2]에 복사한다. 그 후 입력받은 array_size의 값을 저장해놓은 size의 값을 다시 array_size[array_pointer]에 저장한다.

새로이 변수를 입력받은 stack_pointer의 memory_address는 2로 하고 입력된 변수의 크기만큼 1로 만든다. 그 후 새로 입력받은 변수를 위해 stack_pointer를 변수의 크기만큼 증가시킨다. 이때 입력된 배열만큼 1이 되어야 하므로 입력받은 배열의 사이즈만큼 반복문을 돌며 같은 작업을 반복한다. 작업이 종료된 후 stack_pointer에 입력되어있는 값을 array_pointer에 입력한다. 이는 변수의 입력을 받는 메모리 스택과 배열 스택의 위치를 같게 하기 위함이다.

2-4) deleteVariable 함수

```
void deleteVariable() {
    void (*function_pointer)();
    int stack = 0; //배열의 수만큼 지우기 위해 얼마만큼의 스택이 쌓여있는지 알기위해 선언한 변수
    int variable_size = 0; //변수의 크기만큼 지워야하므로 이를 알기위한 변수
    char variable_name[VARIABLE_LENGTH]; //삭제할 변수의 이름을 받기 위해 char 변수 선언

    printf("variable_name : ");
    scanf("%s", variable_name); //삭제할 변수 이름 입력 받음

    for (int i = 0; i < MEMORY_SIZE; i++) { //0부터 원하는 변수의 종류까지 탐색을 함. 이 때 탐색된 위치값을 i라 함
        if (strcmp(variable_name, variable_table[i][2]) == 0) {
            if (strcmp(variable_table[i][1], "char") == 0)
                variable_size = 1;
            else if (strcmp(variable_table[i][1], "int") == 0)
```

```

        variable_size = 4;
    else if (strcmp(variable_table[i][1], "float") == 0)
        variable_size = 4;
    else if (strcmp(variable_table[i][1], "double") == 0)
        variable_size = 8; //입력된 변수의 이름과 문자열을 비교해 그에 맞는
변수의 크기를 입력함

    for (int n = 0; n < variable_size; n++)
        memory_address[i + n] = 0; //0부터 변수의 크기까지 스택에 저장된
값을 0으로 초기화, 하지만 딱 변수의 크기만큼만 반복

    stack = i; //탐색된 위치의 값을 stack이라는 변수에 저장함. 후에 배열만큼 지우
기 위해 선언함

    for (int m = 1; m < array_size[i]; m++) //1부터 배열의 크기만큼 반복
    {
        for (int n = 0; n < variable_size; n++) //앞서 i에서 변수의 크기만큼
진행했다면 미리 i의 값을 stack값에 넣어 stack에 저장된 값에 변수의 크기 및 n만큼 더함
        memory_address[stack+variable_size + n] = 0; //이는 이미 앞
에서 변수의 크기만큼 진행을 하였으므로 그 이후의 값을 지우기위함
        stack += variable_size; //변수의 크기만큼 진행을 했으므로 stack에 변
수의 크기를 저장해 stack값 증가. 따라서 배열만큼 지울 수 있음
    }

    *variable_table[i][1] = 0;
    *variable_table[i][2] = 0; //variable_table 초기화
    array_size[i] = 0; //탐색된 위치의 variable_table 및 array_size 0으로 초기화
    break;
}

}

function_pointer = viewMemory;
function_pointer(); //viewmemory 함수 호출
}

```

위 함수는 deleteVariable 함수로 할당된 변수를 제거하는 함수이다. 얼마만큼의 스택을 지워야 할지
를 저장하기 위해 stack이라는 변수를 추가로 선언하였다.

지울 변수의 이름을 입력받고 이를 0부터 memory_size만큼 탐색하여 그 이름의 변수가 할당되어있는
위치 l를 찾는다. 그 후 l에 입력된 변수의 종류가 무엇인지를 문자열 비교를 통해 알아내고 그에 맞는
변수의 크기 값을 variable_size에 저장한다.

그 후 l부터 변수의 크기만큼의 스택을 0으로 만든다. 즉, 변수의 크기만큼 한번의 값을 지우는 것으
로 입력된 배열이 만약 2라면 1만큼의 값을 0으로 지우는 것이다. 그렇다면 남은 1만큼의 메모리 또
한 스택과 array_size[]에서 지워야한다.

이를 위해 stack 변수에 지우려는 변수를 찾은 위치 l의 값을 저장한다. 그 후 array_size[i]에 저장되
어있는 입력받은 배열의 수만큼 반복하는데 stack+variable_size부터 stack+variable_size+(variable_size-
1)까지 메모리 스택을 0으로 만든다. 즉, b1이라는 이름을 가지는 변수가 int형이고 입력된 배열이 2인
데, 이는 총 메모리 스택에 8칸을 차지하고 있다. 이때 이 b1의 위치 l가 4라고한다면 앞서 4부터 7까

지 4개를 지웠다. 그렇다면 이제 8부터 11까지의 메모리를 지워야 한다. 따라서 $stack(=i)+variable_size=4+4=8$ 부터 $4+4+3=11$ 까지 탐색하며 이를 0으로 만드는 것이다. 그 후 배열의 크기가 2가 아닐 수 있으므로 stack을 변수의 크기만큼 증가시킨다.

배열의 크기만큼 메모리스택을 0으로 초기화를 시켰다면 `variable_table[i][1],[2]`과 `array_size[i]`에 저장된 값도 비워야하므로 이를 진행한 후에 반복문을 나온다. 이때 1번째만 지우는 이유는 1번째를 제외하고 1이 할당된 위치에는 `variable_table`과 `array_size`에는 값이 저장되어있지 않기 때문이다.

2-5) main 함수

```
int main() {
    float check_num; //메뉴 선택을 위해 check_num이라는 float형 변수 선언, 정수가 아닌 값이 입력
    됐을 때를 확인하기 위해 float형 변수로 선언
    int menu; //후에 updatememory 호출을 위한 int형 변수 선언

    do {
        printf("1: exit\n");
        printf("2: memory view\n");
        printf("3: variable allocation\n");
        printf("4: variable delete\n");
        printf(">> select menu : "); //선택할 메뉴 프린트

        if (scanf("%f", &check_num) != 1 || (int)check_num != check_num) {
            while (getchar() != '\n');
            system("cls");
            printf("Please enter an integer\n\n");
            continue;
        } //입력된 값이 정수가 아니라면 정수를 입력하게끔 다시 선언
        else if ((int)check_num < 1 || (int)check_num > 4) {
            system("cls");
            printf("Please enter the correct menu.\n\n");
            continue;
        } //입력된 숫자가 1보다 작거나 4보다 크면 올바른 메뉴를 고르게함
        else if ((int)check_num == 1) {
            printf("\n\nExit the program.\n\n");
            break;
        } //입력된 숫자가 1이라면 프로그램 종료

        system("cls"); //프로세스의 생성&대기&대체의 기능을 하는 system 함수 호출
        menu = check_num; //menu에 입력된 check_num의 값을 저장/ 이미 위에서 int형으로
        변환되어 저장 가능
        printf(" selected menu : %d\n\n", menu);
        updateMemory(menu); //menu로 updateMemory 호출

    } while (1); //do while 문

    return 0;
}
```



```
}
```

위 함수는 main 함수이다. main함수에서는 메뉴선택을 위해 check_num 변수와 후에 updatememory 함수를 호출할 menu변수를 선언하였다. do-while반복문을 사용하여 조건 없이 반복문을 시작한 후에 반복문을 나가는 문구가 나오기 전까지 무한정으로 반복한다.

우선 어떤 메뉴를 선택할지 4개의 문구가 프린트된 후 사용자가 입력한 숫자를 check_num으로 받는다. 이때, 입력된 값이 정수가 아니라면 정수를 입력하게끔 하고 다시 반복문의 시작으로 돌아가 메뉴 입력을 다시 받는다. 또한, 입력받은 값이 메뉴의 범위 1~4에서 벗어난다면 올바른 범위 안에 숫자를 입력하게끔 하고 앞과 같이 반복문의 시작으로 돌아간다. 그리고 입력된 숫자가 1이라면 1은 프로그램을 종료하는 것이므로 break문을 통해 반복문 밖으로 나가 프로그램을 종료시킨다.

메뉴가 1이 아닌 2~4의 범위 내에서 올바르게 입력이 됐다면 menu변수에 check_num 값을 저장하여 그 값에 맞추어 updatememory함수를 호출하여 작동한다.

3. 프로그램 코드

```
/*
학과 : 전자공학과
학번 : 201620837
이름 : 안민규
과목 : 전자공학 프로그래밍
교수님 : 이정원 교수님
*/
#define _CRT_SECURE_NO_WARNINGS
#define MEMORY_SIZE 100
#define VARIABLE_LENGTH 10
#include <stdio.h>
#include <stdbool.h>
#include <string.h>
#include <stdlib.h> //exit()함수를 사용하기 위해 불러옴

void updateMemory(int menu);
void viewMemory();
void allocateVariable();
void deleteVariable(); //사용할 함수들을 미리 선언

int stack_pointer = 0, memory_address[MEMORY_SIZE] = { 0 }, array_pointer = 0,
array_size[MEMORY_SIZE] = { 0 }; //stack 속 위치를 나타내는 stack_pointer, 0층부터 memory_size 만
크의 층을 갖는 memory_address stack, 추후에 입력되는 array_size를 저장하기 위해 새로운 스택 생성 및
위치를 알려주는 array_pointer 선언
char variable_table[MEMORY_SIZE][VARIABLE_LENGTH] = { 0 }; //0층부터 Memory_size까지의 층을
가지며, 한층에 2칸, 한 칸 당 variable_length의 입력을 받을 수 있는 table 선언

int main() {
    float check_num; //메뉴 선택을 위해 check_num이라는 float형 변수 선언, 정수가 아닌 값이 입
력됐을 때를 확인하기 위해 float형 변수로 선언
    int menu; //후에 updatememory 호출을 위한 int형 변수 선언
```

```

do {
    printf("1: exit\n");
    printf("2: memory view\n");
    printf("3: variable allocation\n");
    printf("4: variable delete\n");
    printf(">> select menu : "); //선택할 메뉴 프린트

    if (scanf("%f", &check_num) != 1 || (int)check_num != check_num) {
        while (getchar() != '\n');
        system("cls");
        printf("Please enter an integer\n\n");
        continue;
    } //입력된 값이 정수가 아니라면 정수를 입력하게끔 다시 선언
    else if ((int)check_num < 1 || (int)check_num > 4) {
        system("cls");
        printf("Please enter the correct menu.\n\n");
        continue;
    } //입력된 숫자가 1보다 작거나 4보다 크면 올바른 메뉴를 고르게함
    else if ((int)check_num == 1) {
        printf("\n\nExit the program.\n\n");
        break;
    } //입력된 숫자가 1이라면 프로그램 종료

    system("cls"); //프로세스의 생성&대기&대체의 기능을 하는 system 함수 호출
    menu = check_num; //menu에 입력된 check_num의 값을 저장/ 이미 위에서 int형으
로 변환되어 저장 가능
    printf(" selected menu : %d\n\n", menu);
    updateMemory(menu); //menu로 updateMemory 호출

} while (1); //do while 문

return 0;
}

void updateMemory(int menu) {
    void (*function_pointer)(); //포인터함수 선언

    if (menu == 2) {
        function_pointer = viewMemory;
        function_pointer(); //선택된 메뉴가 2라면 할당된 메모리를 보기 위해 viewmemory 함
수동작
    }
    else if (menu == 3) {
        function_pointer = allocateVariable;
        function_pointer(); //선택된 메뉴가 3라면 변수를 할당하기 위해 allocatevariable 함수

```

동작

```
    }  
    else if (menu == 4) {  
        function_pointer = deleteVariable;  
        function_pointer(); //선택된 메뉴가 4라면 할당된 변수를 지우기 위해 deletevariable 함
```

수동작

```
    }
```

```
}
```

```
void viewMemory() {
```

```
    printf("\nmemory address\n-----\n");
```

```
    for (int n = 0; n < MEMORY_SIZE; n++) { //0부터 memory_size까지 탐색
```

```
        printf("%3d |", n);
```

if (memory_address[n] != 0) { //memory_address[n]이 0이 아닐때까지 진행 -> 즉, 메모리가 할당되어 있음

```
        printf(" } ");
```

if (memory_address[n] == 2) //memory_address[n]이 2라면 변수를 처음 할당한 위치이므로 변수의 이름과 종류 배열의 사이즈를 보여줌

```
        printf("%s %s[%d]", variable_table[n][1], variable_table[n][2],
```

array_size[n]); //메모리가 들어있는 n을 찾아냈다면, 그 때 variable_table의 n번째 1,2. 즉, 할당한 변수의 종류와 이름을 출력/또한 array_size에 저장된 n번째 숫자를 붙여옴

```
    }
```

```
    printf("\n");
```

```
}
```

```
    printf("-----\n\n\n");
```

```
}
```

```
void allocateVariable() {
```

```
    void (*function_pointer)();
```

```
    int variable_size = 0, size = 0; //변수의 크기와 배열의 크기를 받기 위한 int형 변수선언
```

char variable_type[VARIABLE_LENGTH], variable_name[VARIABLE_LENGTH]; //변수의 이름과 타입을 입력받기 위해 char형 변수 선언

```
    printf("variable_type : ");
```

```
    scanf("%s", variable_type);
```

```
    printf("variable_name : ");
```

```
    scanf("%s", variable_name);
```

```
    printf("array_size : ");
```

scanf("%d", &array_size[array_pointer]); //변수의 이름, 종류, 배열 수를 입력받음. 배열 수는 먼저 전역변수로 선언된 array_size[] stack에 저장

```
    size = array_size[array_pointer]; //배열에 저장된 값을 따로 size에 저장
```

```
    strcpy(variable_table[stack_pointer][1], variable_type);
```

strcpy(variable_table[stack_pointer][2], variable_name); //입력받은 변수의 이름과 타입을 variable_table의 stack_pointer층 1,2칸에 저장

```

if (strcmp(variable_type, "char") == 0)
    variable_size = 1;
else if (strcmp(variable_type, "int") == 0)
    variable_size = 4;
else if (strcmp(variable_type, "float") == 0)
    variable_size = 4;
else if (strcmp(variable_type, "double") == 0)
    variable_size = 8; //입력받은 variable_type과 문자열을 비교하여 변수의 종류에 맞추어
variable_size에 값을 저장

if (variable_size * size > MEMORY_SIZE)
{
    printf("Variable size is too large.");
    exit(0);
} //예외처리 (메모리할당 불가), 입력받은 변수의 size와 배열의 크기의 곱이 스택의 사이즈
memory_size보다 크다면 할당불가 프린트하고 프로그램 종료

if (MEMORY_SIZE - stack_pointer < (variable_size) * (array_size[array_pointer]))
{
    for (int i = 0; i < MEMORY_SIZE; i++)
    {
        *variable_table[i][1] = 0;
        *variable_table[i][2] = 0;
        array_size[i] = 0;
        memory_address[i] = 0;
    } //이미 들어가있는 variable_table, array_size, memory_address를 0으로 초기화
    stack_pointer = 0;
    array_pointer = 0; //새로 시작해야하므로 stack_pointer와 array_pointer를 0으로 초기
화

    strcpy(variable_table[stack_pointer][1], variable_type);
    strcpy(variable_table[stack_pointer][2], variable_name); //초기화 전에 입력된 문자열
을 다시 복사해서 variable_table에 삽입. 이 때 stack_pointer가 0이므로 0층부터 다시 저장
    array_size[array_pointer] = size; //size에 저장해놓은 값을 초기화된 array_pointer층
의 array_size에 다시 저장. 이를 위해 size에 미리 값을 저장.
    } //예외처리 (자리 안 남았을 때), 입력한 변수의 할당량보다 메모리가 적게 남았을 경우, 초기화 후
0층부터 다시 진행.

    memory_address[stack_pointer] = 2; //stack_pointer층의 memory stack에는 2를 저장. 입력받
은 층이기 때문에
    for (int n = 1; n < variable_size; n++)
        memory_address[stack_pointer + n] = 1; //입력받은 변수의 크기만큼 1을 저장.

    stack_pointer += variable_size; //stack_pointer를 변수의 크기만큼 증가.

```

```

    for (int i = 1; i < size; i++)
    {
        for (int n = 0; n < variable_size; n++)
            memory_address[stack_pointer + n] = 1;
        stack_pointer += variable_size;
    } //입력된 배열의 크기만큼 1을 채워주는 반복문.

    array_pointer = stack_pointer; //array_pinter를 stack_pointer의 값과 같게 만들. 메모리를 저장하는 memory_address와 배열의 값을 저장하는 array_size가 같은 층에 있게하기 위함

    function_pointer = viewMemory;
    function_pointer(); //viewmemory 함수 호출
}

void deleteVariable() {
    void (*function_pointer)();
    int stack = 0; //배열의 수만큼 지우기 위해 얼마만큼의 스택이 쌓여있는지 알기위해 선언한 변수
    int variable_size = 0; //변수의 크기만큼 지워야하므로 이를 알기위한 변수
    char variable_name[VARIABLE_LENGTH]; //삭제할 변수의 이름을 받기 위해 char 변수 선언

    printf("variable_name : ");
    scanf("%s", variable_name); //삭제할 변수 이름 입력 받음

    for (int i = 0; i < MEMORY_SIZE; i++) { //0부터 원하는 변수의 종류까지 탐색을 함. 이 때 탐색된 위치값을 i라 함
        if (strcmp(variable_name, variable_table[i][2]) == 0) {
            if (strcmp(variable_table[i][1], "char") == 0)
                variable_size = 1;
            else if (strcmp(variable_table[i][1], "int") == 0)
                variable_size = 4;
            else if (strcmp(variable_table[i][1], "float") == 0)
                variable_size = 4;
            else if (strcmp(variable_table[i][1], "double") == 0)
                variable_size = 8; //입력된 변수의 이름과 문자열을 비교해 그에 맞는
변수의 크기를 입력함

            for (int n = 0; n < variable_size; n++)
                memory_address[i + n] = 0; //0부터 변수의 크기까지 스택에 저장된
값을 0으로 초기화. 하지만 딱 변수의 크기만큼만 반복

            stack = i; //탐색된 위치의 값을 stack이라는 변수에 저장함. 후에 배열만큼 지
우기 위해 선언함

            for (int m = 1; m < array_size[i]; m++) //1부터 배열의 크기만큼 반복
            {
                for (int n = 0; n < variable_size; n++) //앞서 i에서 변수의 크기만큼

```

```

진행했다면 미리 i의 값을 stack값에 넣어 stack에 저장된 값에 변수의 크기 및 n만큼 더함
memory_address[stack+variable_size + n] = 0; //이는 이미
앞에서 변수의 크기만큼 진행을 하였으므로 그 이후의 값을 지우기위함
stack += variable_size; //변수의 크기만큼 진행을 했으므로 stack에
변수의 크기를 저장해 stack값 증가. 따라서 배열만큼 지울 수 있음
    }

    *variable_table[i][1] = 0;
    *variable_table[i][2] = 0; //variable_table 초기화
    array_size[i] = 0; //탐색된 위치의 variable_table 및 array_size 0으로 초기화
    break;
}
}

function_pointer = viewMemory;
function_pointer(); //viewmemory 함수 호출
}

```

4. 결과화면 분석 (메모리할당 함수와 메모리 삭제 함수 결과)

메뉴 3 선택 -> 메모리 할당
 type:char, name:a1, array_size:3
 할당 결과
 입력된 변수의 크기1, 배열의 크기 3만큼 3칸의
 메모리 할당 (0~2)

```

selected menu : 3
variable_type : char
variable_name : a1
array_size : 3

memory address
-----
0 | } char a1[3]
1 | }
2 | }
3 | }
4 | }
5 | }

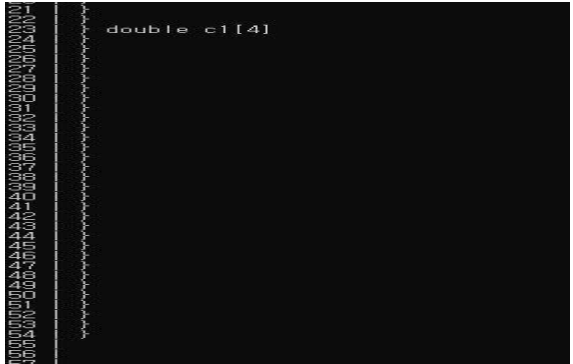
```

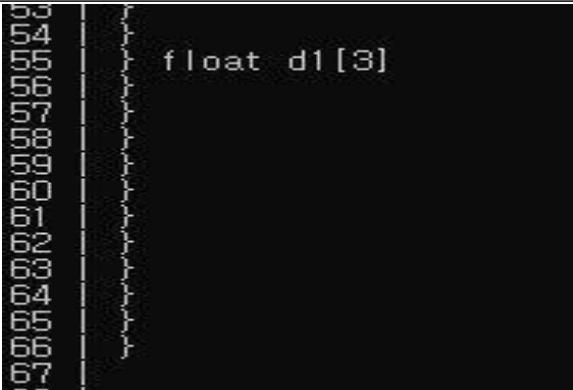
메뉴 3 선택 -> 메모리 할당
 type:char, name:a1, array_size:3
 type:int, name:b1, array_size:5
 할당 결과
 입력된 변수의 크기4, 배열의 크기5만큼 20칸의
 메모리 할당 (3~22)

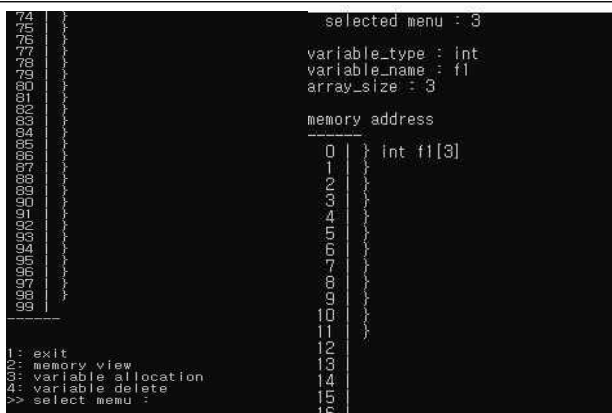
```

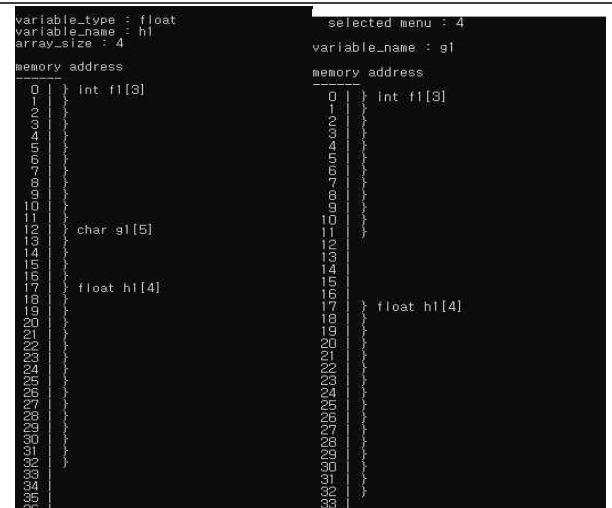
variable_name : b1
array_size : 5
memory address
-----
0 | } char a1[3]
1 | }
2 | }
3 | }
4 | }
5 | }
6 | }
7 | }
8 | }
9 | }
10 | }
11 | }
12 | }
13 | }
14 | }
15 | }
16 | }
17 | }
18 | }
19 | }
20 | }
21 | }
22 | }

```

<p>메뉴 3 선택 -> 메모리 할당</p> <p>type:char, name:a1, array_size:3</p> <p>type:int, name:b1, array_size:5</p> <p>type:double, name:c1, array_size:4</p> <p>할당 결과</p> <p>입력된 변수의 크기8, 배열의 크기4만큼 32칸의 메모리 할당 (23~54)</p>	
---	--

<p>메뉴 3 선택 -> 메모리 할당</p> <p>type:char, name:a1, array_size:3</p> <p>type:int, name:b1, array_size:5</p> <p>type:double, name:c1, array_size:4</p> <p>type:float, name:d1, array_size:3</p> <p>할당 결과</p> <p>입력된 변수의 크기4, 배열의 크기3만큼 12칸의 메모리 할당 (55~66)</p>	
--	--

<p>98까지 메모리가 할당된 이후에 int형 f1 배열의 크기를 3으로 가지는 메모리할당 결과, 메모리 스택의 0부터 다시 탐색하여 메모리를 할당</p>	
--	---

<p>앞의 f1에 이어 char g1[5], float h1[4]를 연달아 입력하여 메모리를 할당한 후, 삭제 메뉴를 통해 g1을 입력하여 g1의 메모리를 지운 결과</p> <p>g1의 배열의 크기만큼 5개가 지워짐을 확인할 수 있다.</p>	
---	--

앞의 f1에 이어 char g1[5], float h1[4]를 연달아
 입력하여 메모리를 할당한 후, 삭제 메뉴를 통해
 g1을 입력하여 g1의 메모리를 지운 결과

g1의 배열의 크기만큼 5개가 지워짐을 확인할 수
 있다.

입력된 배열만큼 입력된 변수의 크기에 맞추어 메
 모리를 할당할 때 메모리사이즈(100)보다 크면 할
 당이 불가능하게 설정

type:double, array_size:13 $8 \times 13 = 104 > 100$
 변수의 크기가 너무 크다는 문구를 프린트한 후
 프로그램 종료

5. 참고문헌
https://ko.wikipedia.org/wiki/%EB%8F%99%EC%A0%81_%EB%A9%94%EB%AA%A8%EB%A6%AC_%ED%95%A0%EB%8B%B9
<https://ko.wikipedia.org/wiki/%EC%8A%A4%ED%83%9D>
 전자공학 프로그래밍 과제, 실습, 포인터 강의노트