Report #1

HW1 – Stack Merge

제출일	2018.09.23	**전공**	전자공학부
과목	자료구조	**학번**	2015112260
담당교수	박대진	**이름**	배준현

우선 stack 을 구현하기 위해서 두 가지 구조체를 정의하였다.

```
typedef struct node {
    void* data_ptr;
    struct node* link;
}   STACK_NODE;
typedef struct stack {
    int count;
    STACK_NODE* top;
}   STACK;
```

node 라는 구조체는 stack의 node 에 들어가는 data 의 주소인 data_ptr 과, 다음 node 의 주소가 담겨 있는 link 를 구조체 멤버로 가진다. stack 이라는 구조체는 stack 의 node 의 수를 나타내는 count와, 현재 stack의 제일 상단 node 를 가리키는 top이라는 구조체 멤버를 가진다.

이 두 구조체를 이용하여, create_stack() 이라는 함수를 정의하였고, 이 함수는 실제로 stack 을 생성하는 역할을 하게 된다.

```
STACK* create_stack() {
    STACK* stack = (STACK*)malloc(sizeof(STACK));
    if ( !stack )
        return 0;
    stack->count = 0;
    stack->top = 0;
    return stack;
}
```

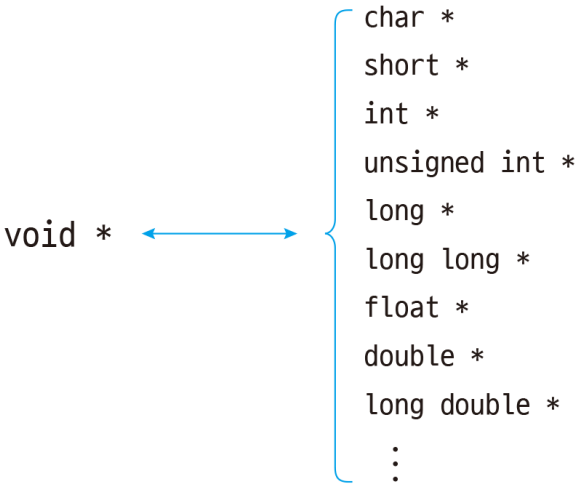
create_stack() 함수는 stack 을 return 하게 되는데, 함수를 정의할 때, typedef로 새롭게 정의한 자료형인 STACK 자료형을 구조체 포인터로 사용하여 malloc() 함수를 이용하여 동적으로 stack 에 메모리를 할당하였다. 그리고 stack 구조체의 멤버인 count 와 top 에 각각 0을 할당 해 주었다.

stack 에 data 를 쓰고 읽기 위한 push() 함수와 pop() 함수를 다음으로 정의하였다. 먼저 push() 함수이다.

```
int push(STACK* stack, void* in) {
    STACK_NODE* node = (STACK_NODE*)malloc(sizeof(STACK_NODE));
    if ( !node )
        return 0;

    node->data_ptr = in;
    node->link = stack->top;
    stack->top = node;
    stack->count++;
    return 1;
}
```

우선 push() 함수는 두 가지 parameters 를 가지는데, push 를 할 대상이 되는 stack 과, 그 stack 에 push 할 data 이다. data 는 자료형에 구애받지 않고 사용하기 위해 void 포인터(범용 포인터)인 void* 를 사용하였다.



우선 push() 함수에서, typedef 함수로 새롭게 정의한 STACK_NODE 라는 자료형의 node 변수를 하나 생성을 해 준 후, malloc() 함수를 이용하여 node 에 STACK_NODE 만큼의 메모리 공간을 동적으로 할당 해 주었다. 그 후, node 의 data_ptr 에는 새롭게 stack에 들어가게 될 data 의 주소인 in을 data_ptr 에 접근하여 할당 해 주었다. 마찬가지로 link 에는 현재 stack 의 가장 상단의 node 의 주소인 top 을 할당 해 주었다. 그리고 현재 stack 의 top 에는 새롭게 생성 된 node 를 할당 해 주고, stack 의 node 의 개수를 나타내는 count 를 1 증가 시켜주었다. push() 함수는 push 에 성공 할 경우 1을, 실패 할 경우 0을 반환하게 설정하였다.

다음으로는 pop() 함수이다.

```
void* pop(STACK* stack) {
    if ( stack->count == 0)
        return 0;
    else{
        STACK_NODE* temp = stack->top;
        void* data_out = stack->top->data_ptr;
        stack->top = stack->top->link;
        free(temp);
        (stack->count)--;
        return data_out;
    }
}
```

pop() 함수는 parameter 로 pop 을 수행 할 대상이 되는 stack 만 있으면 된다. 우선 조건문을 이용하여 pop 을 수행할 수 있는 조건인지를 확인한 후, count 값이 0이라서 pop 을 수행하지 못할 경우에는 0을 반환하게 설정하였다. count 값이 존재하는 경우에는 우선 temp 라는 임시의 stack node를 하나 생

성해준 후, stack 의 top 에 해당하는, 즉, pop 을 할 node 를 temp 에 할당을 해 주었다. 그리고 pop 을 할 대상이 되는 data 의 주소를 data_out 이라는 void 포인터로 선언한 후, stack 의 top 의 data_ptr 을 할당 해 주었다. 그 후, stack 의 새로운 top 이 될 node 의 주소를 원래 top 이 가지고 있던 link 로 새롭게 할당을 해 주었다. 그 후 free() 함수를 이용하여 temp 에 저장이 되어 있는 pop 이 되는 node 의 메모리를 반환 해 주었다. stack 의 count 을 1 줄인 후, data_out, 즉, pop 수행 결과로 나오는 data 의 주소를 pop 함수의 return 값으로 지정 해 주었다.

과제를 수행하기 위해서, stack 을 구현한 후, 과제 수행에 필요한 함수를 생성하였다. 함수 이름은 첫 번째 과제는 HW1_1() 로, 두 번째 과제는 HW1_2() 로 정하였다.

우선 main() 함수 내부에 과제의 기본이 될 두 개의 stack s1, s2를 생성한 후, 각각의 stack에 알맞은 data 를 push 해 두었다.

```
int main() {
    STACK* s1 = create_stack();
    printf("stack s1 is created...\n");
    STACK* s2 = create_stack();
    printf("stack s2 is created...\n");

    int a[5] = {0, 1, 2, 3, 4};
    int b[5] = {5, 6, 7, 8, 9};
    int i, j, s;
    for(i=0;i<(sizeof(a)/sizeof(int));i++){
        push(s1, &a[i]);
    }
    for(j=0;j<(sizeof(b)/sizeof(int));j++){
        push(s2, &b[j]);
    }
    ...
```

그 후, 과제 1을 해결하기 위해 HW1_1() 함수를 다음과 같이 정의하였다.

```
void HW1_1(STACK* s1, STACK* s2) {
    STACK* temp_stack = create_stack();
    int size_temp = s2->count;
    int* temp;
    for(int k=0;k<size_temp;k++){
        temp = (int*)pop(s2);
        push(temp_stack, &(*temp));
    }
    size_temp = temp_stack->count;
    for (int l=0;l<size_temp;l++){
        temp = (int*)pop(temp_stack);
        push(s1, &(*temp));
    }
}
```

HW1_1() 함수는 두 개의 parameters 를 가지는데, 각각 stack s1 과 s2 를 매개변수로 받아서, 이 두 개의 stack 을 이용하여 과제를 해결하게 된다. 우선 stack s2 에서 pop 할 data 들을 임시로 저장해 둘 temp_stack 이라는 새로운 stack 을 임시로 하나 생성 해 주었다. 그 후, stack s2의 count 수를 size_temp 라는 정수형 변수에 저장하고, 이 값과 for 반복문을 이용하여 stack s2 에 pop 한 후, temp_stack 에 push 를 하였다. pop 을 할 때, temp 라는 임시의 정수형 포인터를 사용하였는데, 우선 pop(s2) 를 실행하였을 때, pop() 함수는 return 값이 void 포인터이기 때문에 정수형 포인터로 일종의 형변환을 해준 후 temp 에 이를 할당하였다.

그 후, size_temp 에 이번에는 temp_stack 의 node 개수를 저장해둔 후, 마찬가지로 이 값과 for 반복문을 이용하여 stack s1 에 temp_stack 의 data 들을 차례로 pop 후 push 하였다.

HW1_1() 함수의 결과를 직접 확인해보기 위해서, 현재 stack 을 차례로 pop 하여 print 해 주는 함수인 stack_check() 라는 함수를 정의하였다.

```
void stack_check(STACK* stack) {
    int size_temp = stack->count;
    printf("Final result of the stack s1 is... \n");
    for (int i=0;i<size_temp;i++){
        int* temp;
        temp = (int*)pop(stack);
        printf("%d\n", *temp);
    }
}
```

stack_chec() 함수는 parameter 로 stack 을 받게 되고, 해당 stack 의 data 들을 차례로 pop 하여 보여주게 된다. HW1_1() 함수의 수행 후 stack_check() 함수의 수행 결과는 다음과 같다.

```
Final result of the stack s1 is...
9
8
7
6
5
4
3
2
1
0
FIN.
```

결과를 보았을 때, top 에 해당하는 node 의 data 가 먼저 print 되고, 순차적으로 pop 후 print 되므로 stack s1 이 출력 결과와 동일하게 생겼음을 알 수 있다. (FIN. 은 stack s1 의 data 가 아님.)

다음으로 과제 2를 해결하기 위해 HW1_2() 함수를 다음과 같이 정의하였다.

```
void HW1_2(STACK* s1, STACK* s2) {
    STACK* temp_stack = create_stack();
    int size_temp = s1->count;
    int* temp;
    for(int k=0;k<size_temp-1;k++){
        temp = (int*)pop(s2);
        push(temp_stack, &(*temp));
        temp = (int*)pop(s1);
        push(temp_stack, &(*temp));
    }
    temp = (int*)pop(s2);
    push(s1, &(*temp));
    size_temp = temp_stack->count;
    for (int l=0;l<size_temp;l++){
        temp = (int*)pop(temp_stack);
        push(s1, &(*temp));
    }
}
```

HW1_1() 함수와 마찬가지로 매개변수를 두 개의 stack 으로 받고, temp_stack 을 생성 해 주었다. size_temp 에는 stack s1 의 count 값을 저장하였는데, 현재 상황에서는 s1 과 s2 의 node 의 개수가 동일하므로 문제가 없다. 과제 2에서는, s2 와 s1 을 순서대로 pop 후 temp_stack 에 push 하는 방법을 통해 과제를 해결하였다. for 반복문을 size_temp 보다 1 작게 반복하도록 하였는데, 이는 마지막에는 stack s2에 있는 data '5' 가 temp_stack 에 저장될 필요 없이 바로 stack s1 으로 push 되면 되기 때문이다. 따라서 반복문 후에, 따로 pop & push 를 작성 해 주었다.

HW1_1() 과 마찬가지로 temp_stack 의 data 들을 stack s1 으로 각각 pop & push 하였다. HW1_2() 함수 구동 후에 stack_check() 로 확인해본 결과는 다음과 같았다.

```
Final result of the stack s1 is...
9
4
8
3
7
2
6
1
5
0
FIN.
```

결과를 보면, stack s1 이 과제에서 요구하는 조건을 만족함을 알 수 있다.

번외로, main() 함수에서 switch 문을 이용하여 과제 1과 2를 선택적으로 실행 해 볼 수 있게 하였다. 1 을 누르면 HW1_1() 함수가, 2 를 누르면 HW1_2() 함수가 구동되게 된다.

```
int main() {
    ...
    printf("\nType 1 or 2 for checking my homework...1 means HW1_1, 2 means HW1_2\n");
    scanf("%d", &s);
    switch(s){
        case 1 : HW1_1(s1, s2);
        break;
        case 2 : HW1_2(s1, s2);
        break;
    }
    stack_check(s1);
    printf("FIN.\n\n");
}
```