EDDI
Electronic Design
Development Institute

에디로봇아카데미

임베디드 마스터 Lv2 과정
[자료구조 프로그래밍]

제 1기

2021. 12. 10

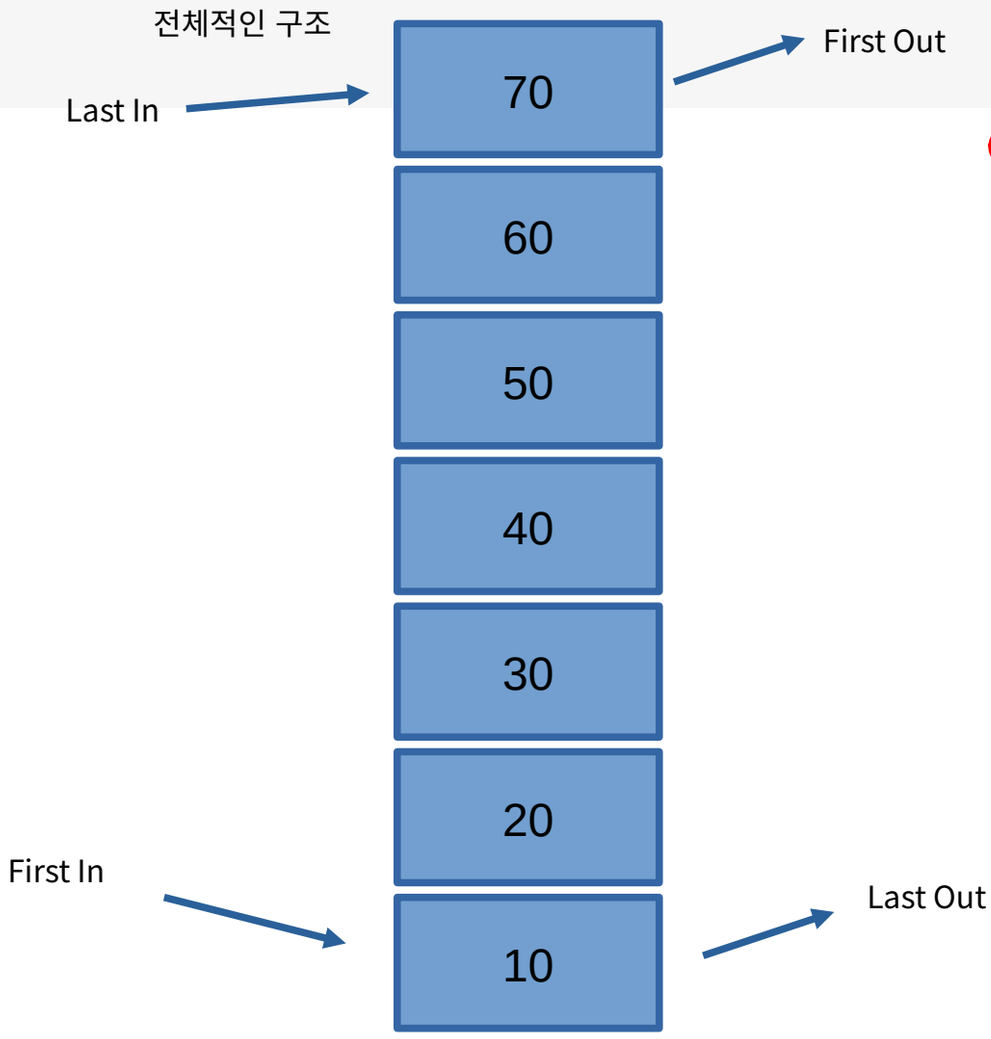
박태인

EDDI
Electronic Design
Development Institute

목차

1) stack 구조 그림을 코드화
2) Queue 구조 그림을 코드화 (다음 시간에 계속..)

1) stack 구조 그림을 코드화



1 각각의 노드를
값의 data와
노드를 가르칠 link 두 분으로 나눈 구조체 정의를 하자.

```
Struct _stack
{
        int data;
        struct _stack *link;
}
```

이것을 typedef 을 이용해서 명칭을 정의 한다.

```
Typedef struct _stack stack;
Struct _stack
{
        int data;
        struct _stack *link;
}
```

```
typedef struct _stack stack;
struct _stack
{
        int data;
        struct _stack *link;
};
```

1) stack 구조 그림을 코드화

② 우선 main을 구성 해본다.

Stack 쌓을 값을 나열하고
for문을 통해
stack을 입력 및 출력 한다.

```
Int main(void)
{
        stack *top = NULL;
        int data[] ={10, 20, 30, 40, 50, 60, 70}
        int I;

        for(i=0; i<7; i++)
        {
                stack 값 in  → push_data(&top, data[i])
        }

        for(t=0; i<7; i++)
        {
                stack 값 out
        }

        return 0;
}
```

```
Void push_data(stack **top, int data)
{
        stack *tmp = *top;

        *top = create_stack_node();
        (*top)→data = data;
        (*top)→link = tmp;
}
```

Push 함수로 오면서 top의 주소 값과 data 값을 가져온다.

그리고 main의 top의 값이 새로 생성 될 node를 가르키게 하기 위해 *top = 생성 node

이제 부터 top 은 heap 영역에 새로 생성될 node를 가르키고 있으므로
(*top)→data 에 data 값을 넣어 준다.

link에는 어떤 값을 넣을 까? link는 앞의 node를 가르켜야 할 것이다. Why?
여기서 tmp가 필요한 이유가 나오는 것.

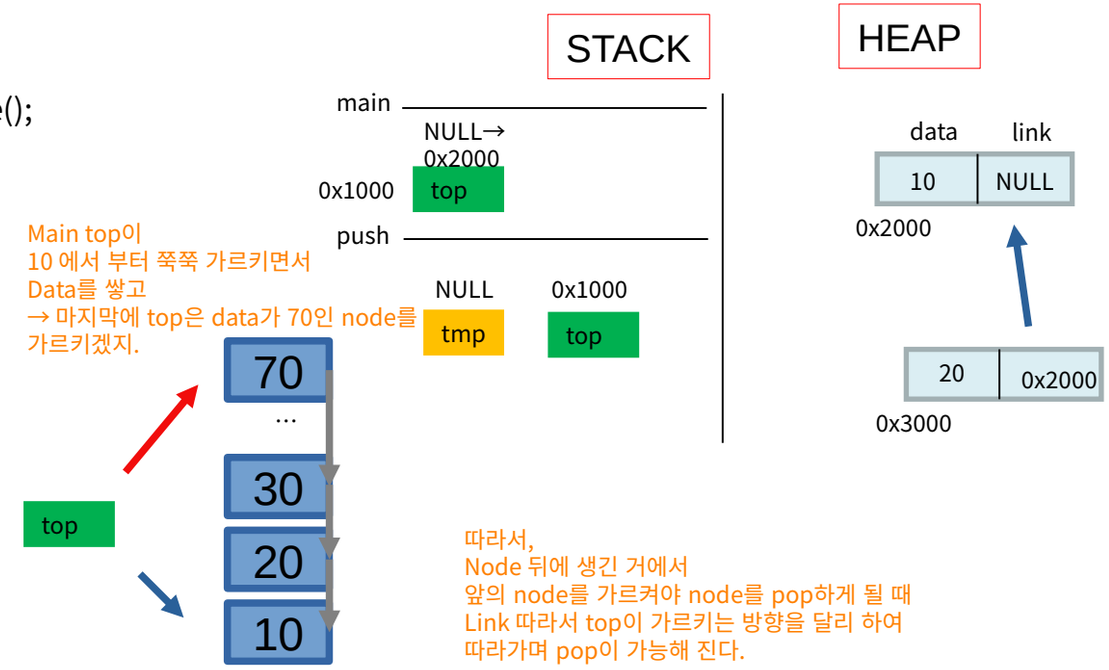
top은 create_node가 됨에 따라 값이 달라지게 될 것이고,
link에 create_node 전 top의 주소를 넣을 수 있도록 tmp에 주소 값을 저장 시켜 놓았다.

③ push에 대한 개념을 잡아보자

조건 - push를 할려면 push를 할 node가 필요하다.
- 첫 시작 node를 top 이라 지칭 하고 시작 값을 NULL 로 한다.
- 가르키는 node에 data를 순서대로 집어 넣는다.

push_data(&top, data[i])

Push 함수를 만들어 보면,
main에서 가르키는 첫 node top의 주소를 가져가고 data를 가져가서
가르키는 node에 data를 넣도록 해보자.



1) stack 구조 그림을 코드화

```
Void push_data(stack **top, int data)
{
        stack *tmp = *top;

        *top = create_stack_node();
        (*top)→data = data;
        (*top)→link = tmp;
}
```

4 앞서서 push 함수를 만들었는데, data를 push 하려면 자연스럽게 node를 생성하는 함수가 필요 할 것이다.

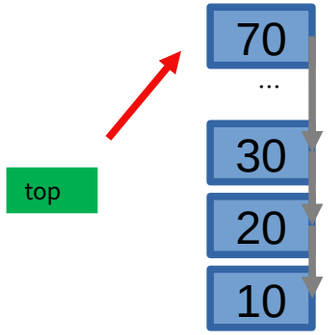
```
create_stack_node();

Stack *create_stack_node(void)
{
        stack *tmp;

        tmp = (stack *)malloc(sizeof(stack));
        tmp→link = NULL;

        return tmp;
}
```

Heap 영역에 node를 생성하기 위해 malloc을 사용 할 것 이고, 그 값을 받기 위한 tmp를 생성.

새로운 node이기 때문에 우선 tmp→link는 NULL 처리.

5 이번에는 pop을 해봐야 겠죠?

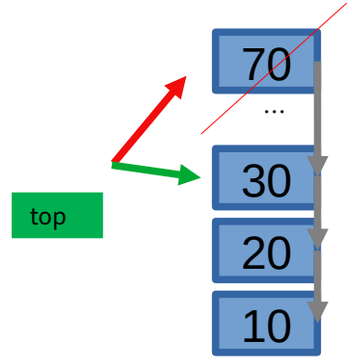
create_stack_node 함수가 stack * 형인 이유?
→ 반환되어 들어가는 top이 *top 이므로



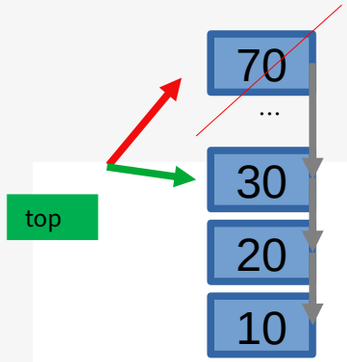
pop의 알고리즘을 생각해 보면
모든 data의 push가 완료 되면 왼쪽과 같은 그림이 되고.

오른쪽 그림과 같이 가르키고 있는 node를 free 시켜 준뒤

link가 가르키는 다음 node로 이동하고,
차례로 가르키는 node를 free 시키도록 한다.



1) stack 구조 그림을 코드화



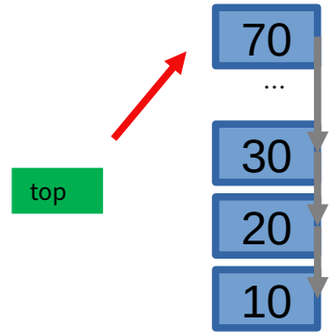

```
Void pop_data(stack **top)
{
        stack *tmp;

        if(!(*top))                     // 우선 pop이 할게 없는지 확인
        {
                printf("stack is empty\n");
                return -1;
        }

        tmp = *top;             // top의 값을 tmp에 백업

        *top = tmp→link;   // tmp의 link는 이전 node 값의 주소가 들어 있으므로 *top을 하면 top이 이전의 node를 가르키게 된다.

        free(tmp);              // free 시킨다.

        return 0;
}
```

1) stack 구조 그림을 코드화

- 이제 main이 좌측과 같은 형태를 가지게 되고, 이번에는 node의 data를 print 하는 함수를 만듦으로써 함수가 제대로 동작하는지 살펴 볼 것이다.

EDDI Electronic Design Development Institute

```
Int main(void)
{
        stack *top = NULL;
        int data[] ={10, 20, 30, 40, 50, 60, 70};
        int I;

        for(i=0; i<7; i++)
        {
                push_data(&top, data[i]);
        }

        print_stack_data(top);

        for(t=0; i<8; i++)
        {
                pop_data(&top);
        }

        print_stack_data(top);

        return 0;
}
```




```
Void print_stack_data(stack *top)
{
        while(top)
        {
                printf("data = %d\n, top→data);  // top의 data 출력
                top = top→link;      //  top을 다음 link 가르키게
        }
        printf("\n");
}
```

```
data = 70
data = 60
data = 50
data = 40
data = 30
data = 20
data = 10

Stack is empty
```

Print 함수는 문제가 없는 듯 하나
Push 하듯이 print 하면 이미

모두 free가 된 상태기 때문에
Stack is empty가 인쇄 되어 버린다.

삭제 되는게 어떤건지 그리고 결과가 어떻게 됐는지 알고 싶으니 고쳐 보자.

```
for(t=0; i<8; i++)
        {
                printf("pop_data = %d\n", pop_data(&top);
                print_stack_data(top);
        }
```

이런식으로 하면 pop 되는게 무엇인지 알 수 있게 되고,
for문을 돌 때 마다 어떤식으로 삭제 됐는지 알 수 있게 된다.

아니 근데, 이렇게 될려면 pop_data가 int형 data를 반환 해야 한다.

1) stack 구조 그림을 코드화

```
for(t=0; i<8; i++)
        {
                printf("pop_data = %d\n", pop_data(&top);
                print_stack_data(top);
        }
```

이런식으로 하면 pop 되는게 무엇인지 알 수 있게 되고,
for문을 돌 때 마다 어떤식으로 삭제 됐는지 알 수 있게 된다.

아니 근데, 이렇게 될려면 pop_data가 int형 data를 반환 해야 한다.

```
int pop_data(stack **top)
{
        int data;
        stack *tmp;

        if(!(*top))                     // 우선 pop이 할게 없는지 확인
        {
                printf("stack is empty\n");
                return -1;
        }

        tmp = *top;             // top의 값을 tmp에 백업

        data = tmp→data;  // 반환 할 data 값은 tmp의 data 값 이다.
        *top = tmp→link;   // tmp의 link는 이전 node 값의 주소가 들어 있으므로 *top을 하면 top이 이전의 node를 가르키게 된다.

        free(tmp);              // free 시킨다.

        return data;
}
```

```
data = 70
data = 60
data = 50
data = 40
data = 30
data = 20
data = 10

pop = 70
data = 60
data = 50
data = 40
data = 30
data = 20
data = 10

pop = 60
data = 50
data = 40
data = 30
data = 20
data = 10

pop = 50
data = 40
data = 30
data = 20
data = 10

pop = 40
data = 30
data = 20
data = 10

pop = 30
data = 20
data = 10

pop = 20
data = 10

pop = 10

Stack is empty
pop = -1
```