두가지 방법으로 stack 구현

5671144-강범창

1.스택을 다음과 같은 방법으로 구현한다.

최대 stack 의 크기

```
#define MAX_STACK_SIZE 10
```

[1] 구조체 배열 사용

```
typedef int element;

typedef struct
{
    element data[MAX_STACK_SIZE];
    int top;
} StackType;
```

[2] 동적 배열 사용

```
typedef int element;

typedef struct
{
    element* data;
    int capacity; // 동적 배열의 경우 capacity 을 넣어 배열의 길이를 조절합니다.
    int top;
} StackType;
```

2. 난수(1~100) 를 발생시켜서 난수가 짝이면 push , 홀이면 pop 를 실행 내용을 출력한다.

[1] 구조체 배열 사용

```
StackType* create(int size)
{
    StackType* sp; // stack pointer
    sp = malloc(sizeof(StackType));
    sp->top = -1;
    return sp;
}

int isFull(StackType* sp)
{
    return (sp->top == (MAX_STACK_SIZE - 1));
}
```

```
void push(StackType* sp, element item)
{
    if (isFull(sp))
    {
        fprintf(stderr, "stack 포화 에러 ₩n");
        return;
    }
    sp->top++;
    sp->data[sp->top] = item;
}

int isEmpty(StackType* sp)
{
    return (sp->top == -1);
}

element pop(StackType* sp)
{
    if (isEmpty(sp))
    {
        fprintf(stderr, "stack 공백 에러 ₩n");
        return -1;
    }
    element result = sp->data[sp->top];
    sp->top--;
    return result;
}
```

[2] 동적 배열 사용

```
StackType* create(int size)
{
    StackType* sp; // stack pointer
    sp = malloc(sizeof(StackType));
    sp->top = -1;
    return sp;
}

int isFull(StackType* sp)
{
    return (sp->top == (sp->capacity - 1));
}

void push(StackType* sp, element item)
{
    if (isFull(sp))
    {
        sp->capacity *= 2; // 배열의 길이 // 배열의 길이를 동적으로 늘립니다.
        sp->data = (element*)realloc(sp->data, sp->capacity * sizeof(element));
    }
    sp->top++;
    sp->data[sp->top] = item;
}
```

```
int isEmpty(StackType* sp)
{
    return (sp->top == -1);
}

element pop(StackType* sp)
{
    if (isEmpty(sp))
    {
        fprintf(stderr, "stack 공백 에러 ₩n");
        return -1;
    }
    element result = sp->data[sp->top];
    sp->top--;
    return result;
}
```

main 부분은 동일합니다. 짝수면 push 홀수면 pop 을 실행합니다.

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

    StackType* stack = create(MAX_STACK_SIZE);

    for (int i = 0; i < 30; i++)
    {
        int ranNum = rand() % 100 + 1; // 1~100 의 랜덤된 수를 적성합니다.
        if (ranNum % 2 == 0) // 값이 짝수면 push 아니면 pop 을합니다.
        {
            push(stack, ranNum);
        }
        else
        {
            element item = pop(stack); // stack 에서 item 을 pop 해와 print합니다.
            if (item != -1)
            {
                printf("pop : %d₩n", item);
            }
        }
    }
    system("pause"); // console 창이 꺼지지 않도록  하기 위함 입니다.
}
```

3. 앞의 2를 30회 실행하고 실행 결과를 2가지 구현 방법에 대하여 비교 평가한다.

[1] 구조체 배열 사용

```
pop : 32
pop : 42
pop : 86
pop : 26
pop : 78
pop : 26
stack 포화  에러
pop : 10
pop : 82
pop : 22
pop : 74
pop : 22
```

[2] 동적 배열 사용

```
pop : 26
pop : 26
pop : 52
pop : 2
stack 공백  에러
pop : 58
pop : 46
pop : 42
pop : 88
pop : 46
pop : 78
pop : 8
pop : 54
pop : 30
```

구조체 배열과 , 동적 배열의 차이는 구조체 배열은 동적으로 길이를 늘리지 못하여 stack 포화 에러 즉 stack의 최대크기만큼 가득 차버리는 문제가 있었는데, 동적 배열의 경우 공백 에러만 나올 뿐 포화 에러는 나오지 않습니다. 그 이유는 배열의 크기가 최대크기 만큼 가득 차버렸을 때 동적의 경우에는 최대 길이를 다시 설정한후, 다시 메모리의 크기 즉 배열의 크기를 늘려버리기 때문입니다,

결론적으로 구조체 배열과 동적 배열의 차이는 배열의 길이를 늘려 포화 상태가 되나 안되는지 에 대한 것 입니다.