1.소스코드

#include <stdio.h>

#include <stdlib.h>

#define MAX\_STACK\_SIZE 10

typedef int element;

typedef struct {

element\* data; // data은 포인터로 정의된다.

int capacity; // 현재 크기

int top;

} StackType;

// 스택 생성 함수

void init\_stack(StackType\* s)

{

s->top = -1;

s->capacity = 1;

s->data = (element\*)malloc(s->capacity \* sizeof(element));

}

// 공백 상태 검출 함수

int is\_empty(StackType\* s)

{

return (s->top == -1);

}

// 포화 상태 검출 함수

int is\_full(StackType\* s)

{

return (s->top == (MAX\_STACK\_SIZE - 1));

}

void push(StackType\* s, element item)

{

if (is\_full(s)) {

s->capacity \*= 2;

s->data =

(element\*)realloc(s->data, s->capacity \* sizeof(element));

}

s->data[++(s->top)] = item;

}

// 삭제함수

element pop(StackType\* s)

{

if (is\_empty(s)) {

printf("pop -1\n");

fprintf(stderr, "Stack Empty!\n");

return;

}

else

{

printf("pop : %d\n", s->data[(s->top)]);

return s->data[(s->top)--];

}

}

int main(void)

{

StackType s;

init\_stack(&s);

srand(time(NULL));

printf("-------using Malloc-------\n");

int rand\_num;

for (int i = 0; i < 30; i++)

{

printf("[%d]\t", i);

rand\_num = rand() % 100 + 1;

if (rand\_num % 2 == 0)

{

printf("push : %d\n", rand\_num);

push(&s, rand\_num);

}

else {

pop(&s);

}

}

}

2. 실행화면

차트이(가) 표시된 사진

자동 생성된 설명

3. 느낀점

Malloc 즉 동적배열을 사용하니 기존의 스택들과 다르게 is\_full이 일어나지 않으니 내가 원하는 만큼 담을 수 있다는 것이 좋은 것 같다. 오류도 덜 나니 신경쓸것도 적어졌다. 하지만 메모리를 찰때마다 2배씩 잡는다면 나중에 커졌을 때 괜찮을지 의문이 들었다.