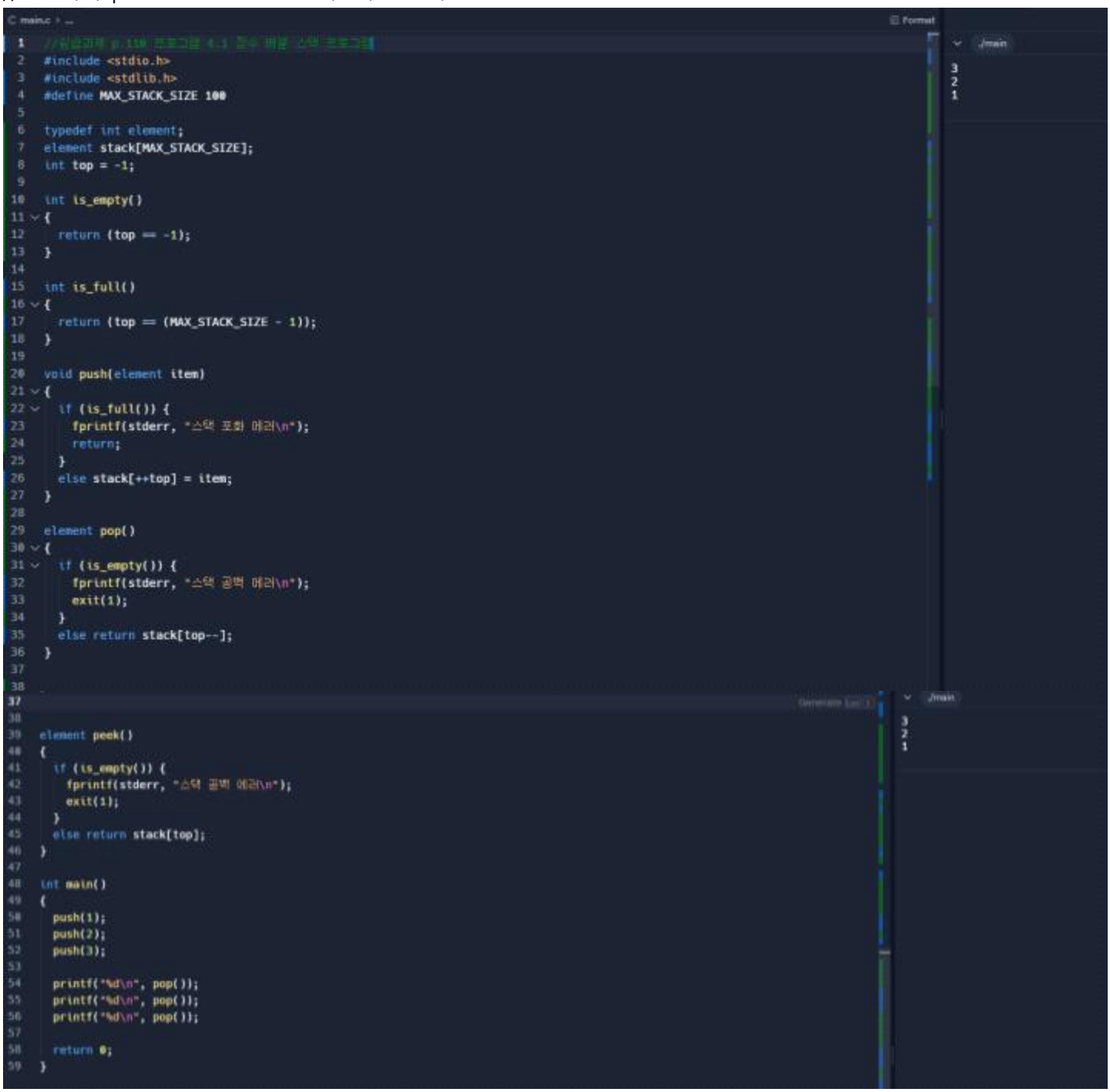
데이터 구조 4장 실습과제

20223100 박신조

//실습과제 p.110 프로그램 4.1 정수 배열 스택 프로그램



```c
//실습과제 p.110 프로그램 4.1 정수 배열 스택 프로그램
#include <stdio.h>
#include <stdlib.h>
#define MAX_STACK_SIZE 100

typedef int element;
element stack[MAX_STACK_SIZE];
int top = -1;

int is_empty()
{
  return (top == -1);
}

int is_full()
{
  return (top == (MAX_STACK_SIZE - 1));
}

void push(element item)
{
  if (is_full()) {
    fprintf(stderr, "스택 포화 에러\n");
    return;
  }
  else stack[++top] = item;
}

element pop()
{
  if (is_empty()) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return stack[top--];
}

element peek()
{
  if (is_empty()) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return stack[top];
}

int main()
{
  push(1);
  push(2);
  push(3);

  printf("%d\n", pop());
  printf("%d\n", pop());
  printf("%d\n", pop());

  return 0;
}
```

```
3
2
1
```

위 코드는 배열로 스택 구현한 코드입니다.
스택은 후입선출(LIFO, Last In First Out) 방식으로 동작합니다.
ex) 책상에 쌓여있는 책, 창고에 쌓여있는 상자.

#define MAX_STACK_SIZE 100

MAX_STACK_SIZE를 100으로 설정합니다.
ex) int arr[MAX_STACK_SIZE] -> arr배열의 크기를 100으로 초기화합니다.

typedef int element;

element를 int자료형을 정의합니다.

ex) element stack[MAX_STACK_SIZE]는 stack배열을 element타입(int형)으로 정의합니다.

int top = -1;

top변수의 초기값을 –1으로 설정합니다. 이 변수는 스택의 요소가 추가될때 +1씩 되며 –1일땐 스택이 비어있음을 의미합니다.

int is_empty()

스택이 비어 있는지를 확인하는 함수입니다.

top == -1일 경우 스택이 비어있다면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full()

스택이 가득 찼는지를 확인하는 함수입니다.

top == (MAX_STACK_SIZE - 1)일 경우 스택이 가득차 있다면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

이때 MAX_STACK_SIZE – 1인 이유는

1. top == MAX_STACK_SIZE 라면 stact배열의 크기보다 요소 수가 많기 때문에 에러가 납니다.(저장공간 초과)
2. top < MAX_STACK_SIZE-1 이라면 stack배열의 잔여 공간이 남아있습니다.

void push(element item)

스택에 데이터를 삽입하는 함수입니다.

is_full 함수를 호출하여 스택이 가득 찼으면 에러 메시지를 출력하고 함수를 종료합니다.

그렇지 않으면 top을 1 증가시키고 이전에 저장된 데이터 바로 뒤에 저장됩니다.

element pop()

스택에서 데이터를 꺼내는 함수입니다.

is_empty함수를 호출하여 스택이 비어 있으면 에러 메시지를 출력하고 프로그램을 종료합니다.

그렇지 않으면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.(후입선출)

element peek()

스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.

pop 함수와 동일하게 스택이 비어 있으면 에러 메시지를 출력하고 프로그램을 종료합니다.

pop()함수와 동일하게 가장 마지막에 저장된 값을 반환하지만 삭제시키는 것은 아닙니다.

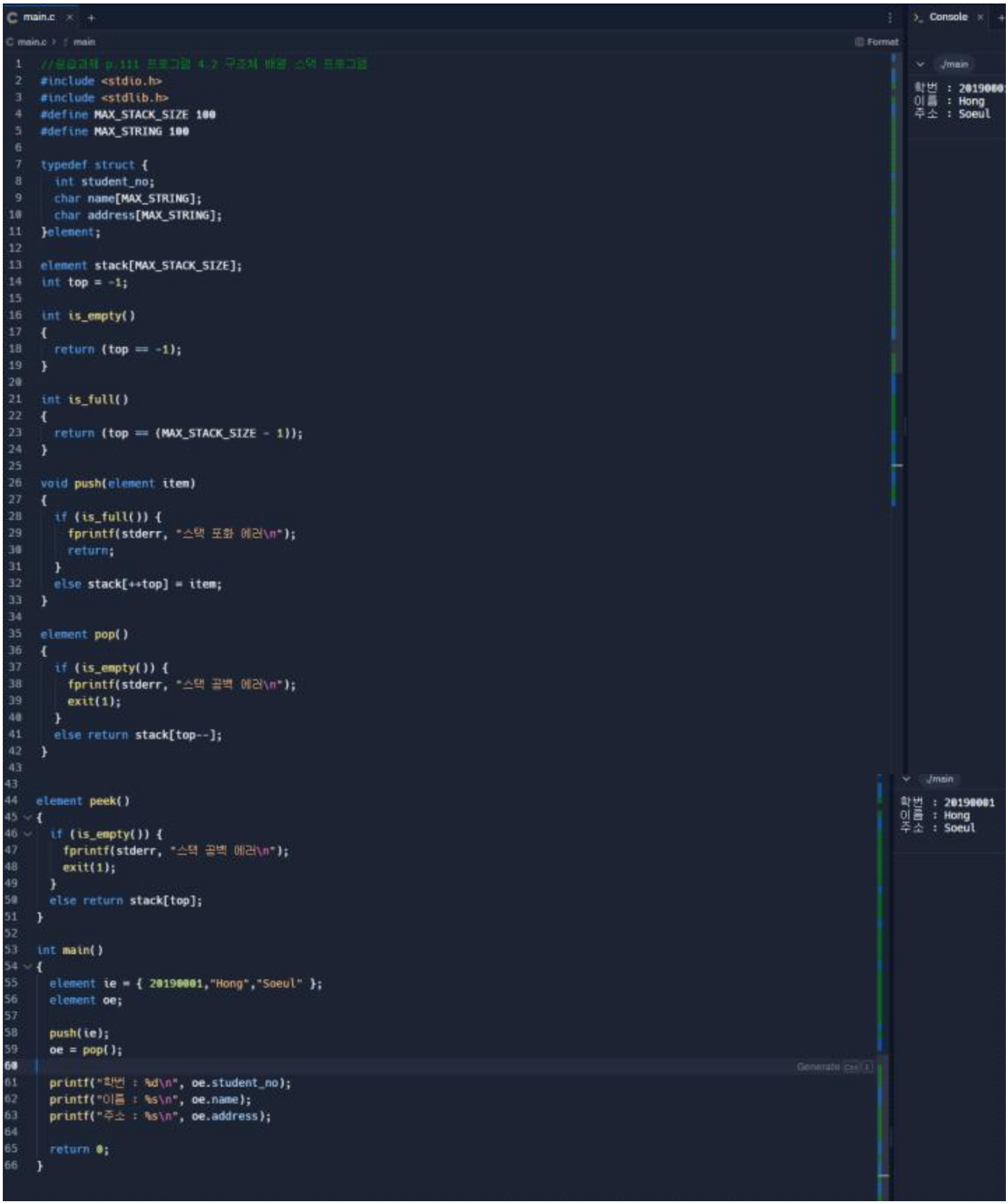
top 의 값과 stack의 요소 수는 변하지 않습니다.

main()

push를 통해 1 -> 2 -> 3 순으로 stack에 저장되고

pop을 통해 가장 마지막에 저장된 3 -> 2 -> 1순으로 반환되어 출력됩니다.

//실습과제 p.111 프로그램 4.2 구조체 배열 스택 프로그램



```c
//실습과제 p.111 프로그램 4.2 구조체 배열 스택 프로그램
#include <stdio.h>
#include <stdlib.h>
#define MAX_STACK_SIZE 100
#define MAX_STRING 100

typedef struct {
  int student_no;
  char name[MAX_STRING];
  char address[MAX_STRING];
}element;

element stack[MAX_STACK_SIZE];
int top = -1;

int is_empty()
{
  return (top == -1);
}

int is_full()
{
  return (top == (MAX_STACK_SIZE - 1));
}

void push(element item)
{
  if (is_full()) {
    fprintf(stderr, "스택 포화 에러\n");
    return;
  }
  else stack[++top] = item;
}

element pop()
{
  if (is_empty()) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return stack[top--];
}


element peek()
{
  if (is_empty()) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return stack[top];
}

int main()
{
  element ie = { 20190001,"Hong","Soeul" };
  element oe;

  push(ie);
  oe = pop();

  printf("학번 : %d\n", oe.student_no);
  printf("이름 : %s\n", oe.name);
  printf("주소 : %s\n", oe.address);

  return 0;
}
```

```
Console
./main
학번 : 2019000
이름 : Hong
주소 : Soeul
```

```
./msin
학번 : 20190001
이름 : Hong
주소 : Soeul
```

위 코드는 구조체로 스택 구현한 코드입니다.
위 실습 4.1에선 자료형이 1가지로 제한적이였지만 구조체로 스택을 구현하여 여러 자료형들을 스택 1칸에 저장가능합니다.

#define MAX_STACK_SIZE 100 #define MAX_STRING 100

MAX_STACK_SIZE를 100으로 설정합니다.
MAX_STRING를 100으로 설정합니다.

typedef struct { ... } element;
이 부분은 element라는 이름으로 구조체를 생성합니다.
구조체 멤버 : int student; char name[MAX_STRING]; char address[MAX_STRING];
element는 3가지 데이터 값을 저장할 수 있습니다.

element stack[MAX_STACK_SIZE];
stack는 구조체 element 타입의 배열입니다.
ex) stack[index].student -> stack의 index번째 배열의 student 자료형의 값 가르킵니다.

int top = -1;
실습 4.1에서 설명했듯이 스택의 마지막 데이터 값을 가르키는 변수입니다. -1일때 스택은 비어있습니다.

int is_empty()
스택이 비어 있는지 확인합니다.
top == -1이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full()
스택이 가득 찼는지를 확인합니다.
top == (MAX_STACK_SIZE - 1)이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

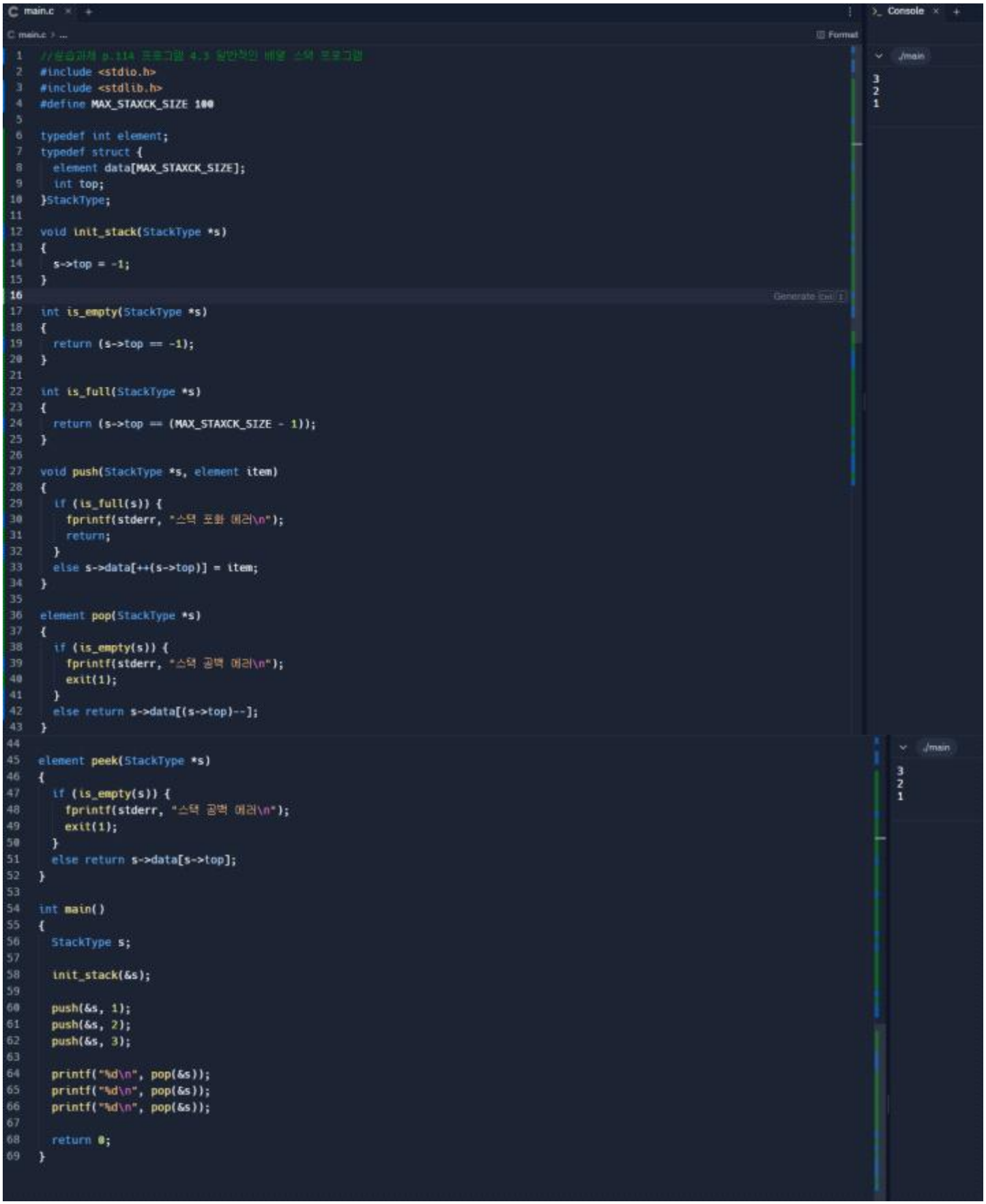
void push(element item)
스택에 데이터를 삽입합니다.
is_full()함수를 호출하여 리턴 값이 1이면 top을 1 증가시켜 새로운 데이터를 삽입합니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element pop()
스택에서 데이터를 꺼내는 함수입니다.
is_empty함수를 호출하여 리턴 값이 1이면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element peek()
스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.
가장 마지막에 저장된 값을 반환합니다. -> 데이터 삭제X 그냥 보기만하는것
스택이 비어있다면 에러 메시지를 출력합니다.

main()
element ie = { 20194034, "Yun", "YongIn" }; // element(구조체)타입의 ie를 선언하고 {"~"}값으로 초기화합니다.
push를 통해 ie의 값을 스택에 삽입합니다.
pop을 통해 가장 마지막에 삽입한 데이터를 oe에 값을 저장합니다.
eo의 구조체 멤버들을 호출하여 값을 출력합니다.

//실습과제 p.114 프로그램 4.3 일반적인 배열 스택 프로그램



```c
//실습과제 p.114 프로그램 4.3 일반적인 배열 스택 프로그램
#include <stdio.h>
#include <stdlib.h>
#define MAX_STAXCK_SIZE 100

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

void init_stack(StackType *s)
{
  s->top = -1;
}

int is_empty(StackType *s)
{
  return (s->top == -1);
}

int is_full(StackType *s)
{
  return (s->top == (MAX_STAXCK_SIZE - 1));
}

void push(StackType *s, element item)
{
  if (is_full(s)) {
    fprintf(stderr, "스택 포화 에러\n");
    return;
  }
  else s->data[++(s->top)] = item;
}

element pop(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[(s->top)--];
}

element peek(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[s->top];
}

int main()
{
  StackType s;

  init_stack(&s);

  push(&s, 1);
  push(&s, 2);
  push(&s, 3);

  printf("%d\n", pop(&s));
  printf("%d\n", pop(&s));
  printf("%d\n", pop(&s));

  return 0;
}
```

```
3
2
1
```

구조체를 사용하여 스택을 구현하고, 포인터를 활용하여 스택을 관리하는 코드입니다.

#define MAX_STACK_SIZE 100

MAX_STACK_SIZE를 100으로 정의합니다.

typedef int element;
element를 int자료형을 정의합니다.

typedef struct { ... } StackType;
이 부분은 stackType 라는 이름으로 구조체를 생성합니다.
구조체 멤버 : element data[MAX_STACK_SIZE]; int top;
stackType는 크기가 100인 element타입의 배열과 int형 top에 데이터를 저장할 수 있습니다.
이때 data 배열에 데이터 값이 저장되고 top 변수를 통해 데이터 위치가 저장됩니다.

void init_stack(StackType *s)
실습 4.1, 4.2의 int top = -1; 와 같은 역활을 합니다.
스택을 초기화하는 함수입니다. top을 -1로 설정하여 스택이 비어있는 상태로 초기화합니다.
이 함수는 StackType 구조체의 포인터 s를 매개변수로 받습니다. 이는 수정한 값을 리턴해주는 역활을 합니다.
정확히는 main문의 구조체 s의 주소값을 받아와 구조체의 데이터를 수정할 수 있습니다.(함수 내에서 수정 가능)

int is_empty(StackType *s)
스택이 비어 있는지 확인합니다.
top == -1이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full(StackType *s)
스택이 가득 찼는지를 확인합니다.
top == (MAX_STACK_SIZE - 1)이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

void push(StackType *s, element item)
스택에 데이터를 삽입합니다.
is_full()함수를 호출하여
리턴 값이 1이면 top을 1 증가시켜 새로운 데이터를 삽입합니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element pop(StackType *s)
스택에서 데이터를 꺼내는 함수입니다.
is_empty함수를 호출하여
리턴 값이 1이면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element peek(StackType *s)
스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.
가장 마지막에 저장된 값을 반환합니다. -> 데이터 삭제X 그냥 보기만하는것
스택이 비어있다면 에러 메시지를 출력합니다.

main()
StackType s; 구조체 s를 선언합니다.
init_stack(&s); 구조체 s의 값을 초기화합니다. 이때 포인터를 통해 함수에서 바로 구조체의 값을 수정합니다.
init_stack 함수를 호출하여 스택을 초기화합니다. &s는 s의 주소를 전달하여 구조체를 수정할 수 있도록 합니다.
push를 통해 1 -> 2 -> 3 순서로 값을 삽입합니다.
pop을 통해 삽입의 역순으로 출력합니다. 3 -> 2 -> 1

//실습과제 p.116 프로그램 4.4 동적 스택 프로그램

```c
//실습과제 p.116 프로그램 4.4 동적 스택 프로그램
#include <stdio.h>
#include <stdlib.h>
#define MAX_STAXCK_SIZE 100

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

void init_stack(StackType *s)
{
  s->top = -1;
}

int is_empty(StackType *s)
{
  return (s->top == -1);
}

int is_full(StackType *s)
{
  return (s->top == (MAX_STAXCK_SIZE - 1));
}

void push(StackType *s, element item)
{
  if (is_full(s)) {
    fprintf(stderr, "스택 포화 에러\n");
    return;
  }
  else s->data[++(s->top)] = item;
}

element pop(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[(s->top)--];
}

element peek(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[s->top];
}

int main()
{
  StackType *s;
  s = (StackType *)malloc(sizeof(StackType));

  init_stack(s);

  push(s, 1);
  push(s, 2);
  push(s, 3);

  printf("%d\n", pop(s));
  printf("%d\n", pop(s));
  printf("%d\n", pop(s));

  free(s);
  return 0;
}
```

```
./main
3
2
1
```

동적 메모리를 할당하여 스택을 사용하는 코드입니다.
코드 1~53번 줄까지 실습 4.3과 동일하지만 main 함수에서 동적 메모리를 할당한다는 차이점이 있습니다.

#define MAX_STACK_SIZE 100

MAX_STACK_SIZE를 100으로 정의합니다.

typedef int element;

element를 int자료형을 정의합니다.

typedef struct { ... } StackType;

이 부분은 stackType 라는 이름으로 구조체를 생성합니다.
구조체 멤버 : element data[MAX_STACK_SIZE]; int top;
stackType는 크기가 100인 element타입의 배열과 int형 top에 데이터를 저장할 수 있습니다.
이때 data 배열에 데이터 값이 저장되고 top 변수를 통해 데이터 위치가 저장됩니다.

void init_stack(StackType *s)

실습 4-1, 4-2의 int top = -1; 와 같은 역활을 합니다.
스택을 초기화하는 함수입니다. top을 -1로 설정하여 스택이 비어있는 상태로 초기화합니다.
이 함수는 StackType 구조체의 포인터 s를 매개변수로 받습니다. 이는 수정한 값을 리턴해주는 역활을 합니다.
정확히는 main문의 구조체 s의 주소값을 받아와 구조체의 데이터를 수정할 수 있습니다.(함수 내에서 수정 가능)

int is_empty(StackType *s)

스택이 비어 있는지 확인합니다.
top == -1이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full(StackType *s)

스택이 가득 찼는지를 확인합니다.
top == (MAX_STACK_SIZE - 1)이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

void push(StackType *s, element item)

스택에 데이터를 삽입합니다.
is_full()함수를 호출하여
리턴 값이 1이면 top을 1 증가시켜 새로운 데이터를 삽입합니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element pop(StackType *s)

스택에서 데이터를 꺼내는 함수입니다.
is_empty함수를 호출하여
리턴 값이 1이면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

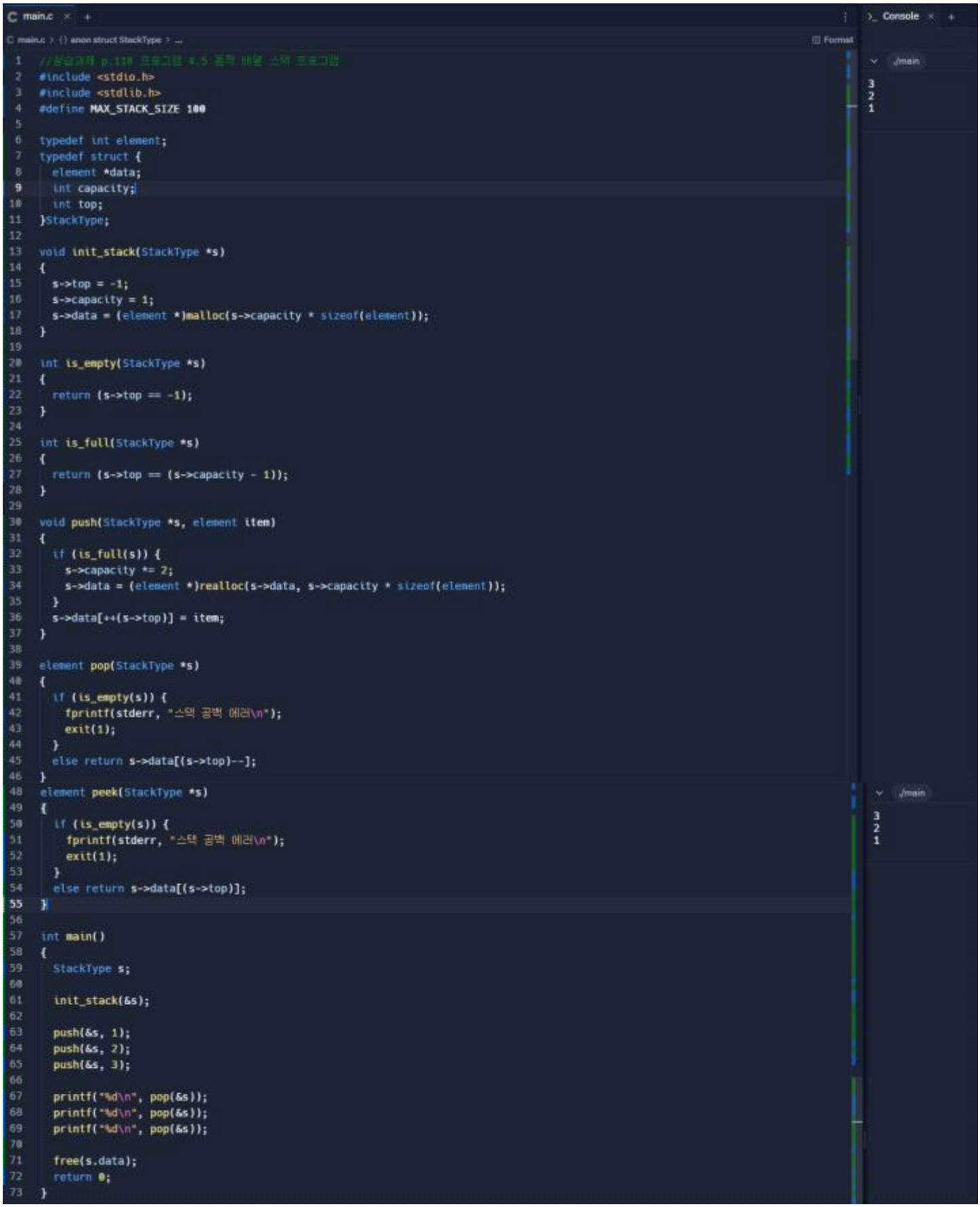
element peek(StackType *s)

스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.
가장 마지막에 저장된 값을 반환합니다. -> 데이터 삭제X 그냥 보기만하는것
스택이 비어있다면 에러 메시지를 출력합니다.

main()

StackType *s;
s = (StackType *)malloc(sizeof(StackType));
StackType의 포인터 s를 선언하고 stackType크기 만큼의 메모리를 동적으로 할당하여 s에 시작 위치를 저장합니다.
init_stack(s); 구조체 s의 값을 초기화합니다. 이때 포인터를 통해 함수에서 바로 구조체의 값을 수정합니다.
인자가 &s가 아닌 이유는 실습4.3에서의 구조체 s는 포인터가 아니기 때문에 포인터처럼 사용하기 위해 &를 사용했지만
현재 코드의 s는 타입이 포인터이기 때문에 함수 인자에 &가 없다.
push를 통해 1 -> 2 -> 3 순서로 값을 삽입합니다.
pop을 통해 삽입의 역순으로 출력합니다. 3 -> 2 -> 1
free(s); 할당한 메모리를 해제하는 부분입니다.
main함수가 끝나면 더 이상 동적 메모리를 사용하지 않기 때문에 반납을 해야합니다.
그렇지 않으면 메모리 누수가 발생할 수 있습니다.

//실습과제 p.118 프로그램 4.5 동적 배열 스택 프로그램



```c
//실습과제 p.118 프로그램 4.5 동적 배열 스택 프로그램
#include <stdio.h>
#include <stdlib.h>
#define MAX_STACK_SIZE 100

typedef int element;
typedef struct {
  element *data;
  int capacity;
  int top;
}StackType;

void init_stack(StackType *s)
{
  s->top = -1;
  s->capacity = 1;
  s->data = (element *)malloc(s->capacity * sizeof(element));
}

int is_empty(StackType *s)
{
  return (s->top == -1);
}

int is_full(StackType *s)
{
  return (s->top == (s->capacity - 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;
}

element pop(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[(s->top)--];
}
element peek(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[(s->top)];
}

int main()
{
  StackType s;

  init_stack(&s);

  push(&s, 1);
  push(&s, 2);
  push(&s, 3);

  printf("%d\n", pop(&s));
  printf("%d\n", pop(&s));
  printf("%d\n", pop(&s));

  free(s.data);
  return 0;
}
```

```
3
2
1
```

```
3
2
1
```

동적 배열을 사용하여 스택 구현한 코드입니다.
스택의 크기를 동적으로 조정할 수 있으며, 스택이 가득 차면 자동으로 크기를 늘려주는 방식입니다.

typedef int element;

element를 int자료형을 정의합니다.

typedef struct { ... } StackType;

이 부분은 stackType라는 이름으로 구조체를 생성합니다.

구조체 멤버 : element *data; int capacity; int top;

*data에 값이 저장되고 스택이 가득 차면 capacity에 의해 크기가 늘어납니다. top은 이전에 삽입 데이터의 주소를 나타냅니다.

void init_stack(StackType *s)

스택을 초기화하는 함수입니다.

s->top = -1;

int top = -1 과 같은 역활을 합니다. 스택을 비어있는 상태로 초기화 합니다.

s->capacity = 1;

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

메모리를 (element*capacity)크기 만큼 동적으로 할당하여 data에 주소를 저장합니다.

이때 capacity의 값을 1로 설정하여 초기 메모리를 할당합니다.

int is_empty(StackType *s)

스택이 비어 있는지 확인합니다.

top == -1이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full(StackType *s)

스택이 가득 찼는지를 확인합니다.

top == (MAX_STACK_SIZE - 1)이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

void push(StackType *s, element item)

스택에 데이터를 삽입합니다.

스택이 가득 차있다면 스택의 용량을 두 배로 늘립니다.

ex)

s->capacity *= 2;

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

realloc함수를 사용하여 data 배열의 크기를 늘립니다. 이전에 할당된 메모리 크기를 새로운 크기로 재조정합니다.

element pop(StackType *s)

스택에서 데이터를 꺼내는 함수입니다.

is_empty함수를 호출하여

리턴 값이 1이면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.

리턴 값이 0이면 에러 메시지를 출력합니다.

element peek(StackType *s)

스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.

가장 마지막에 저장된 값을 반환합니다. -> 데이터 삭제X 그냥 보기만하는것

스택이 비어있다면 에러 메시지를 출력합니다.

main()

StackType s; 구조체 s를 선언합니다.

init_stack(&s); 스택을 초기화합니다. [top은 -1, capacity는 1, data는 element*capacity크기 만큼 동적으로 할당]

push를 통해 1 -> 2 -> 3 순서로 값을 삽입합니다. 이때 스택이 가득 차면 capacity를 증가시켜 스택의 용량을 두 배로 늘립니다.

pop을 통해 삽입의 역순으로 출력합니다. 3 -> 2 -> 1

free(s.data); 할당한 메모리를 해제합니다.

main함수가 끝나면 더 이상 동적 메모리를 사용하지 않기 때문에 반납을 해야합니다.

그렇지 않으면 메모리 누수가 발생할 수 있습니다.

p119 추가 과제 : 괄호 검사문제

괄호의 검사 조건
1. 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야한다.
2. 같은 종류의 괄호에서 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야한다.
3. 서로 다른 종류의 왼쪽 괄호와 오른쪽 괄호 쌍은 서로 교차하면 안된다.

아래 예시가 틀린이유
1. (a (b) -> 괄호 순서 (( )
왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같이 않다 ') ' 오른쪽 괄호가 한개 더 추가되어야 한다 -> 조건 1 위배

2. a (b) c) -> 괄호 순서 ( ))
오른쪽 괄호가 왼쪽 괄호보다 1개더 많으므로 ') ' 괄호가 나오기 전에 먼저 ' (' 괄호가 1개 더 나와야 옳바른 괄호가 된다. -> 조건 1, 2 위배

3. a { b (c [d] e) f } -> 괄호 순서 { ([ ]) }
왼쪽 괄호에 첫번째로 ' { ' 가 나오면 오른쪽 괄호의 시작도 ' } '로 시작되어야 한다.
왼쪽 괄호의 종류와 오른쪽 괄호의 종류가 같아야 한다. -> 조건 3 위배

//실습과제 p.123 프로그램 4.6 괄호 검사 프로그램

main.c

```c
//실습과제 p.123 프로그램 4.6 괄호 검사 프로그램
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#define MAX_STAXCK_SIZE 100

typedef char element;
typedef struct {
  element data[MAX_STAXCK_SIZE];
  int top;
}StackType;

void init_stack(StackType *s) {s->top = -1;}

int is_empty(StackType *s) {return (s->top == -1);}

int is_full(StackType *s){return (s->top == (MAX_STAXCK_SIZE - 1));}

void push(StackType *s, element item)
{
  if (is_full(s)) {
    fprintf(stderr, "스택 포화 에러\n");
    return;
  }
  else s->data[++(s->top)] = item;
}

element pop(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[(s->top)--];
}

element peek(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\\n");
    exit(1);
  }
  else return s->data[s->top];
}
int check_matching(const char *in)
{
  StackType s;
  char ch, open_ch;
  int i, n = strlen(in);
  init_stack(&s);

  for (i = 0; i < n; i++) {
    ch = in[i];

    switch (ch) {
    case '(':case '[': case '{':
      push(&s, ch);
      break;
    case ')':case ']':case '}':
      if (is_empty(&s))
        return 0;
      else {
        open_ch = pop(&s);
        if ((open_ch == '('&&ch != ')') ||
          (open_ch == '['&&ch != ']') ||
          (open_ch == '{'&&ch != '}')) {
          return 0;
        }
        break;
      }
    }
  }
  if (!is_empty(&s))
    return 0;
  return 1;
}

int main()
{
  char *p = "{A[(i+1)]=0;}";
  if (check_matching(p) == 1)
    printf("%s 괄호검사성공\n", p);
  else
    printf("%s 괄호검사실패\n", p);
  return 0;
}
```

Console

```
./main
{A[(i+1)]=0;} 괄호검사성공
```

```
./main
{A[(i+1)]=0;} 괄호검사성공
```

이 코드는 괄호 검사 프로그램입니다. 주어진 문자열에서 괄호가 올바르게 짝을 이루는지 확인하는 프로그램으로, 스택을 사용하여 후입선출(LIFO) 방식으로 괄호의 짝을 맞추는 방식을 구현합니다.

typedef char element;
element를 char 자료형을 정의합니다.

typedef struct { ... } StackType;
이 부분은 stackType 라는 이름으로 구조체를 생성합니다.
구조체 멤버 : element data[MAX_STACK_SIZE]; int top;
stackType는 크기가 100인 element타입의 배열과 int형 top에 데이터를 저장할 수 있습니다.
이때 data 배열에 데이터 값이 저장되고 top 변수를 통해 데이터 위치가 저장됩니다.

void init_stack(StackType *s)
실습 4.1, 4.2의 int top = -1; 와 같은 역활을 합니다.
스택을 초기화하는 함수입니다. top을 -1로 설정하여 스택이 비어있는 상태로 초기화합니다.
이 함수는 StackType 구조체의 포인터 s를 매개변수로 받습니다. 이는 수정한 값을 리턴해주는 역활을 합니다.
정확히는 main문의 구조체 s의 주소값을 받아와 구조체의 데이터를 수정할 수 있습니다.(함수 내에서 수정 가능)

int is_empty(StackType *s)
스택이 비어 있는지 확인합니다.
top == -1이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full(StackType *s)
스택이 가득 찼는지를 확인합니다.
top == (MAX_STACK_SIZE - 1)이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

void push(StackType *s, element item)
스택에 데이터를 삽입합니다.
is_full()함수를 호출하여
리턴 값이 1이면 top을 1 증가시켜 새로운 데이터를 삽입합니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element pop(StackType *s)
스택에서 데이터를 꺼내는 함수입니다.
is_empty함수를 호출하여
리턴 값이 1이면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element peek(StackType *s)
스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.
가장 마지막에 저장된 값을 반환합니다. -> 데이터 삭제X 그냥 보기만하는것
스택이 비어있다면 에러 메시지를 출력합니다.

int check_matching(const char *in)
괄호가 짝을 이루는지 확인하는 함수입니다.
이 함수는 주어진 문자열을 한 문자씩 순차적으로 읽으면서 괄호를 처리합니다:
열린 괄호((, [, {)가 나오면 스택에 해당 괄호를 푸시합니다.
닫힌 괄호(),], })가 나오면 스택에서 하나를 꺼내서 짝이 맞는지 확인합니다. 만약 짝이 맞지 않거나 스택이 비어 있으면 잘못된 괄호로 판단하여 0을 반환합니다.
문자열 끝까지 순차적으로 검사한 후 스택이 비어 있지 않다면 짝을 맞추지 못한 괄호가 있다는 의미로 0을 반환합니다. 모두 맞으면 1을 반환하여 괄호 검사가 성공했음을 알립니다.

main()
main 함수에서는 check_matching 함수를 호출하여 문자열에 대해 괄호 검사를 실행합니다. 만약 검사 결과가 1이면 괄호 검사 성공 메시지를 출력하고, 0이면 괄호 검사 실패 메시지를 출력합니다.

//실습과제 p.127 프로그램 4.7 후위표기식 계산

main.c

```c
//실습과제 p.127 프로그램 4.7 후위표기식 계산
#include <stdio.h>
#include <stdlib.h>
#define MAX_STAXCK_SIZE 100

typedef char element;
typedef struct {
  element data[MAX_STAXCK_SIZE];
  int top;
}StackType;

void init_stack(StackType *s)
{
  s->top = -1;
}

int is_empty(StackType *s)
{
  return (s->top == -1);
}

int is_full(StackType *s)
{
  return (s->top == (MAX_STAXCK_SIZE - 1));
}

void push(StackType *s, element item)
{
  if (is_full(s)) {
    fprintf(stderr, "스택 포화 에러\n");
    return;
  }
  else s->data[++(s->top)] = item;
}

element pop(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[(s->top)--];
}
element peek(StackType *s)
{
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[s->top];
}

int eval(char exp[])
{
  int op1, op2, value, i = 0;
  int len = strlen(exp);
  char ch;
  StackType s;

  init_stack(&s);
  for (i = 0; i < len; i++) {
    ch = exp[i];
    if (ch != '+'&&ch != '-'&&ch != '*'&&ch != '/') {
      value = ch - '0';
      push(&s, value);
    }
    else {
      op2 = pop(&s);
      op1 = pop(&s);
      switch (ch) {
      case '+':push(&s, op1 + op2); break;
      case '-':push(&s, op1 - op2); break;
      case '*':push(&s, op1 * op2); break;
      case '/':push(&s, op1 / op2); break;
      }
    }
  }
  return pop(&s);
}

int main()
{
  int result;
  printf("후위표기식은 82/3-32*+\n");
  result = eval("82/3-32*+");
  printf("결과값은 %d\n", result);
  return 0;
}
```

Console

```
./main
후위표기식은 82/3-32*+
결과값은 7
```


```
./main
후위표기식은 82/3-32*+
결과값은 7
```

수식을 표기할땐 중의, 전위, 후위 3가지 방법이 있습니다.
중위 표기법이란 대부분의 사람들이 수식을 작성할때 쓰는 방법으로 2*3+7=13 로 표기됩니다.
전위 표기법이란 연산자가 피연산자 앞에 위치하는 수식입니다. +*237=13 로 표기합니다.
후위 표기법이란 연산자가 피연산자 뒤에 위치하는 수식입니다. 23*7+=13 로 표기합니다.
보통 컴파일러가 수식을 표시할땐 후위 표기법을 사용합니다.

위 코드는 후위표기식 계산을 구현한 코드입니다.

typedef char element;
element를 char 자료형을 정의합니다.

typedef struct { ... } StackType;
이 부분은 stackType 라는 이름으로 구조체를 생성합니다.
구조체 멤버 : element data[MAX_STACK_SIZE]; int top;
stackType는 크기가 100인 element타입의 배열과 int형 top에 데이터를 저장할 수 있습니다.
이때 data 배열에 데이터 값이 저장되고 top 변수를 통해 데이터 위치가 저장됩니다.

void init_stack(StackType *s)
실습 4.1, 4.2의 int top = -1; 와 같은 역활을 합니다.
스택을 초기화하는 함수입니다. top을 -1로 설정하여 스택이 비어있는 상태로 초기화합니다.
이 함수는 StackType 구조체의 포인터 s를 매개변수로 받습니다. 이는 수정한 값을 리턴해주는 역활을 합니다.
정확히는 main문의 구조체 s의 주소값을 받아와 구조체의 데이터를 수정할 수 있습니다.(함수 내에서 수정 가능)

int is_empty(StackType *s)
스택이 비어 있는지 확인합니다.
top == -1이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full(StackType *s)
스택이 가득 찼는지를 확인합니다.
top == (MAX_STACK_SIZE - 1)이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

void push(StackType *s, element item)
스택에 데이터를 삽입합니다.
is_full()함수를 호출하여
리턴 값이 1이면 top을 1 증가시켜 새로운 데이터를 삽입합니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element pop(StackType *s)
스택에서 데이터를 꺼내는 함수입니다.
is_empty함수를 호출하여
리턴 값이 1이면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element peek(StackType *s)
스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.
가장 마지막에 저장된 값을 반환합니다. -> 데이터 삭제X 그냥 보기만하는것
스택이 비어있다면 에러 메시지를 출력합니다.

int eval(char exp[])
후위표기식 수식을 계산하는 함수입니다.
int형 변수 op1, op2, value, i=0을 선언합니다.
이때 op1 op2는 각각 스택에 저장된 숫자들을 pop하여 저장할 변수입니다.
value는 char형인 exp[i]를 정수형으로 변경하여 저장할 int타입 변수입니다.
char ch는 exp의 요소를 저장할 변수입니다.
StackType s는 후위 계산식에 사용될 숫자를 저장할 스택입니다.(숫자라면 push, 연산자라면 pop)
init_stack(&s)는 스택을 비어있는 상태로 초기화시켜줍니다.
만약 exp[index]가 숫자(피연산자)라면, 그 값을 스택에 삽입합니다.

이때 exp[]배열은 char형으로 문자이지만 스택에 저장할땐 아스키코드를 사용하여 ch - '0'을 통해 int형으로 저장합니다.
만약 문자가 연산자(+, -, *, /)라면, 스택에서 두 개의 피연산자를 꺼내어 연산을 수행한 후 결과를 다시 스택에 삽입합니다.
계산이 모두 끝나면 스택에 남아있는 결과 값을 반환합니다.

main()

eval 함수를 호출하여 후위표기식을 "82/3-32*+" 계산하고 결과를 출력합니다.
후위표기식 계산 과정
주어진 후위표기식 "82/3-32*+"을 계산하는 과정은 다음과 같습니다:

exp[i]		스택							
		[0]	[1]	[2]	[3]	[4]	[5]	. . .	
0	8	8							exp[i] == 숫자 push
1	2	8	2						
2	/	4							exp[i] == 연산자 pop->op1, pop->op2
3	3	4	3						계산 결과값 push
4	-	1							
5	3	1	3						
6	2	1	3	2					
7	*	1	6						
8	+	7							최종 결과값 return
return		7							

//실습과제 p.133 프로그램 4.8 중위표기식을 후위표기식으로 변환하는 프로그램

```
//실습과제 p.133 프로그램 4.8 중위표기식을 후위표기식으로 변환하는 프로그램
#include<stdio.h>
#include<stdlib.h>
#define MAX_STACK_SIZE 100

typedef char element;

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

void init_stack(StackType *s){s->top = -1;}
int is_empty(StackType *s){  return (s->top == -1);}
int is_full(StackType *s){  return (s->top == (MAX_STACK_SIZE - 1));}

void push(StackType *s, element item){
  if (is_full(s)) {
    fprintf(stderr, "스택 포화 에러\n");
    return;
  }
  else s->data[++(s->top)] = item;
}
element pop(StackType *s){
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[(s->top)--];
}

element peek(StackType* s){
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[s->top];
}
int prec(char op)
{
  switch (op) {
  case '(':case ')':return 0;
  case '+':case '-':return 1;
  case '*':case '/': return 2;
  }
  return -1;
}
void infix_to_postfix(char exp[])
{
  int i = 0;
  char ch, top_op;
  int len = strlen(exp);
  StackType s;

  init_stack(&s);
  for (i = 0; i < len; i++) {
    ch = exp[i];
    switch (ch) {
    case '+': case '-': case '*': case '/':
      while (!is_empty(&s) && (prec(ch) <= prec(peek(&s))))
        printf("%c", pop(&s));
      push(&s, ch);
      break;
    case '(':
      push(&s, ch);
      break;
    case ')':
      top_op = pop(&s);
      while (top_op != '(') {
        printf("%c", top_op);
        top_op = pop(&s);
      }
      break;
    default:
      printf("%c", ch);
      break;
    }
  }
  while (!is_empty(&s))
    printf("%c", pop(&s));
}

int main(void)
{
  char *s = "(2+3)*4+9";
  printf("중위표기수식 %s \n", s);
  printf("후위표기수식 ");
  infix_to_postfix(s);
  printf("\n");
  return 0;
}
```

```
./main
중위표기수식 (2+3)*4+9
후위표기수식 23+4*9+
```


```
./main
중위표기수식 (2+3)*4+9
후위표기수식 23+4*9+
```

중위표기식을 후위표기식으로 변환하는 알고리즘을 구현한 코드입니다.

typedef char element;
element를 char 자료형을 정의합니다.

typedef struct { ... } StackType;
이 부분은 stackType 라는 이름으로 구조체를 생성합니다.
구조체 멤버 : element data[MAX_STACK_SIZE]; int top;
stackType는 크기가 100인 element타입의 배열과 int형 top에 데이터를 저장할 수 있습니다.
이때 data 배열에 데이터 값이 저장되고 top 변수를 통해 데이터 위치가 저장됩니다.

void init_stack(StackType *s)
실습 4.1, 4.2의 int top = -1; 와 같은 역할을 합니다.
스택을 초기화하는 함수입니다. top을 -1로 설정하여 스택이 비어있는 상태로 초기화합니다.
이 함수는 StackType 구조체의 포인터 s를 매개변수로 받습니다. 이는 수정한 값을 리턴해주는 역할을 합니다.
정확히는 main문의 구조체 s의 주소값을 받아와 구조체의 데이터를 수정할 수 있습니다.(함수 내에서 수정 가능)

int is_empty(StackType *s)
스택이 비어 있는지 확인합니다.
top == -1이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full(StackType *s)
스택이 가득 찼는지를 확인합니다.
top == (MAX_STACK_SIZE - 1)이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

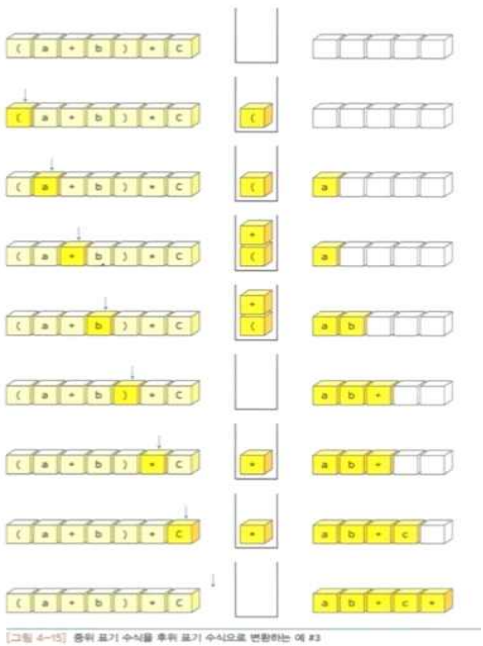
void push(StackType *s, element item)
스택에 데이터를 삽입합니다.
is_full()함수를 호출하여
리턴 값이 1이면 top을 1 증가시켜 새로운 데이터를 삽입합니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element pop(StackType *s)
스택에서 데이터를 꺼내는 함수입니다.
is_empty함수를 호출하여
리턴 값이 1이면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element peek(StackType *s)
스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.
가장 마지막에 저장된 값을 반환합니다. -> 데이터 삭제X 그냥 보기만하는것
스택이 비어있다면 에러 메시지를 출력합니다.

int prec(char op)
prec 함수는 주어진 연산자의 우선순위를 반환하는 함수입니다.
ex) '(', ')' > *, % > +, -
괄호=0 / 뎃셈, 뺄셈=1 / 곱셈, 나눗셈=2

void infix_to_postfix(char exp[])
중위표기식 exp를 후위표기식으로 변환합니다.
exp[]의 요소들을 1문자씩 ch에 저장하여 피연산자면 바로 출력합니다.
연산자라면 스택에 저장 또는 스택의 연산자와 비교하여 우선순위가 높은 것을 출력합니다.
만약 (가 나오면 스택에 넣고,)가 나오면 스택에서 (가 나올 때까지 모든 연산자를 출력합니다.
모든 문자를 처리한 후, 스택에 남아있는 연산자들을 차례대로 출력합니다.



int main()
중위표기수식을 저장하여 infix_to_postfix()함수를 호출하여 후위표기수식으로 출력합니다.

//실습과제 p.139 프로그램 4.9 미로탐색 프로그램

```c
//실습과제 p.139 프로그램 4.9 미로탐색 프로그램
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#define MAX_STACK_SIZE 100
#define MAZE_SIZE 6

typedef struct {
  short r;
  short c;
}element;

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

void init_stack(StackType *s){s->top = -1;}
int is_empty(StackType *s){  return (s->top == -1);}
int is_full(StackType *s){  return (s->top == (MAX_STACK_SIZE - 1));}

void push(StackType *s, element item){
  if (is_full(s)) {
    fprintf(stderr, "스택 포화 에러\n");
    return;
  }
  else s->data[++(s->top)] = item;
}
element pop(StackType *s){
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[(s->top)--];
}

element peek(StackType* s){
  if (is_empty(s)) {
    fprintf(stderr, "스택 공백 에러\n");
    exit(1);
  }
  else return s->data[s->top];
}
element here = { 1, 0 }, entry = { 1, 0 };
char maze[MAZE_SIZE][MAZE_SIZE] = {
  {'1', '1', '1', '1', '1', '1'},
  {'0', '0', '1', '0', '0', '1'},
  {'1', '0', '0', '0', '1', '1'},
  {'1', '0', '1', '0', '1', '1'},
  {'1', '0', '1', '0', '0', 'x'},
  {'1', '1', '1', '1', '1', '1'},
};
void push_loc(StackType* s, int r, int c) {
  if (r < 0 || c < 0) return;
  if (maze[r][c] != '1' && maze[r][c] != '.') {
    element tmp;
    tmp.r = r;
    tmp.c = c;
    push(s, tmp);
  }
}
void maze_print(char maze[MAZE_SIZE][MAZE_SIZE]) {
  printf("\n");
  for (int r = 0;r < MAZE_SIZE;r++) {
    for (int c = 0;c < MAZE_SIZE;c++) {
      printf("%c", maze[r][c]);
    }
    printf("\n");
  }
}
int main(void) {
  int r, c;
  StackType s;
  init_stack(&s);
  here = entry;
  while (maze[here.r][here.c] != 'x') {
    r = here.r;
    c = here.c;
    maze[r][c] = '.';
    maze_print(maze);
    push_loc(&s, r - 1, c);
    push_loc(&s, r + 1, c);
    push_loc(&s, r, c - 1);
    push_loc(&s, r, c + 1);
    if(is_empty(&s)){
      printf("실패\n");
      return 0;
    }
    else
      here = pop(&s);
  }
  printf("성공\n");
  return 0;
}
```

```
/main

111111
.01001
100011
101011
10100x
111111

111111
..1001
100011
101011
10100x
111111

111111
..1001
1.0011
101011
10100x
111111

111111
..1001
1..011
101011
10100x
111111

111111
..1001
1...11
101011
10100x
111111

111111
..1001
1...11
101.11
10100x
111111

111111
..1001
1...11
101.11
101.0x
111111

111111
..1001
1...11
101.11
101..x
111111
성공
```

```
/main

111111
.01001
100011
101011
10100x
111111

111111
..1001
100011
101011
10100x
111111

111111
..1001
1.0011
101011
10100x
111111

111111
..1001
1..011
101011
10100x
111111

111111
..1001
1...11
101011
10100x
111111

111111
..1001
1...11
101.11
10100x
111111

111111
..1001
1...11
101.11
101.0x
111111

111111
..1001
1...11
101.11
101..x
111111
성공
```

스택을 이용하여 미로를 탐색하고 탈출을 구현한 코드입니다.
이 코드는 깊이 우선 탐색 (DFS, Depth-First Search) 방법을 사용하여 미로를 탐색합니다.
깊이 우선 탐색이란 (후입선출)
1-1길을 탐색하고 이후 1-2길, 1-3길을 탐색하고 이후 갈 곳이 없을 때 2-1길, 2-2길, 2-3길을 탐색하여 원하는 결과를 찾는 것을 칭합니다.
다른 방식으론 넓이 우선 탐색(BFS, Breadth-First Search)가 있습니다.
넓이 우선 탐색이란 (선입선출)
깊이 우선 탐색과는 정 반대로 1-1길, 2-1길, 3-1길. . . 모든 길을 탐색하고 이후 1-2길, 2-2길, 3-2길. . .이런식으로 탐색해 나가는 방식입니다.

typedef struct {. . .}element;
현재 위치를 저장하는 구조체입니다.

typedef struct {. . .}StackType;
이 부분은 stackType 라는 이름으로 구조체를 생성합니다.
미로탐색에서 탐색할 수 있는 길을 저장합니다.
구조체 멤버 : element data[MAX_STACK_SIZE]; int top;
stackType는 크기가 100인 element타입의 배열과 int형 top에 데이터를 저장할 수 있습니다.
이때 data 배열에 데이터 값이 저장되고 top 변수를 통해 데이터 위치가 저장됩니다.

void init_stack(StackType *s)
실습 4.1, 4.2의 int top = -1; 와 같은 역활을 합니다.
스택을 초기화하는 함수입니다. top을 -1로 설정하여 스택이 비어있는 상태로 초기화합니다.
이 함수는 StackType 구조체의 포인터 s를 매개변수로 받습니다. 이는 수정한 값을 리턴해주는 역활을 합니다.
정확히는 main문의 구조체 s의 주소값을 받아와 구조체의 데이터를 수정할 수 있습니다.(함수 내에서 수정 가능)

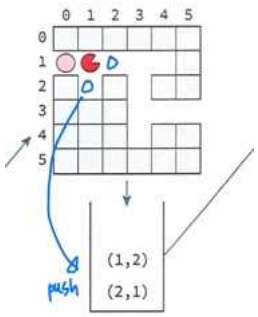
int is_empty(StackType *s)
스택이 비어 있는지 확인합니다.
top == -1이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

int is_full(StackType *s)
스택이 가득 찼는지를 확인합니다.
top == (MAX_STACK_SIZE - 1)이면 1을 반환하고, 그렇지 않으면 0을 반환합니다.

void push(StackType *s, element item)
스택에 데이터를 삽입합니다.
is_full()함수를 호출하여
리턴 값이 1이면 top을 1 증가시켜 새로운 데이터를 삽입합니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element pop(StackType *s)
스택에서 데이터를 꺼내는 함수입니다.
is_empty함수를 호출하여
리턴 값이 1이면 top을 1 감소시키고 stack에서 가장 마지막에 저장된 데이터를 반환하고 데이터를 삭제시킵니다.
리턴 값이 0이면 에러 메시지를 출력합니다.

element peek(StackType *s)
스택에서 가장 최근에 삽입된 데이터를 확인하는 함수입니다.
가장 마지막에 저장된 값을 반환합니다. -> 데이터 삭제X 그냥 보기만하는것
스택이 비어있다면 에러 메시지를 출력합니다.



void push_loc(StackType* s, int r, int c)
미로에서 탐색할 수 있는 위치를 스택에 삽입하는 함수입니다.
상하좌우로 이동하며, 벽('1')이나 이미 방문한 곳('.')을 제외한 위치를 스택에 추가합니다.
ex) 현재 위치를 (1,1)이라 하면 (1,2), (2,1)을 스택에 저장합니다.

void maze_print(char maze[MAZE_SIZE][MAZE_SIZE])
현재 미로 상태를 출력하는 함수입니다.
미로에서 각 셀의 상태를 출력하여, 미로 탐색의 진행 상태를 보여줍니다.

'1'은 벽, '0'은 비어있는 공간, 'e'는 시작점, 'x'는 출구, '.'은 방문한 지점을 나타냅니다.

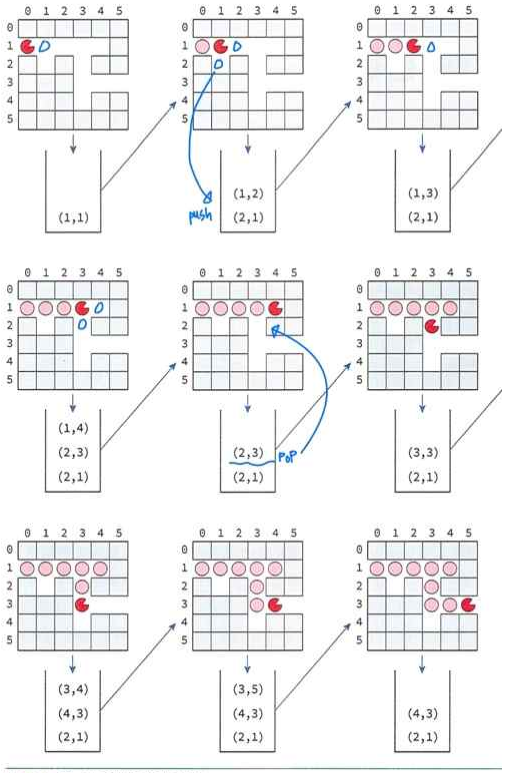
main()

스택 구조체를 선언하고 시작위치를 (1.0)으로 초기화합니다.
이후 while 반복문을 통해 탐색을 시작합니다.
현재 위치를 받아와 push_loc() 함수를 호출하여 상하좌우에 갈 수 있는 위치(0이라면)를 스택에 삽입합니다.
이후 스택에 있는 위치를 다시 불러와 또 상하좌우를 탐색하여 스택에 삽입합니다.
이 반복문은 스택이 비어있거나 x에 도착할때까지 반복됩니다.
이 과정을 maze_print() 함수를 활용하여 출력합니다.

미로 탐색 과정

시작 위치 초기화: here를 (1, 0)으로 설정하고 탐색을 시작합니다.

탐색: 상하좌우로 갈 수 있는 위치를 스택에 넣고, 그 위치로 이동하면서 미로를 탐색합니다.
이동한 위치는 '.'으로 표시하여 이미 방문한 곳을 표시합니다.
실패: 더 이상 갈 수 있는 곳이 없으면(스택이 비어있는 상태면) "실패"를 출력합니다.
성공: 출구('x')에 도달하면 "성공"을 출력합니다.



[그림 4-17] 미로탐색문제 알고리즘