스택

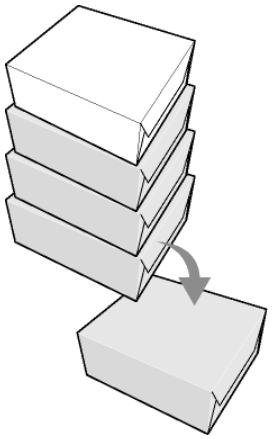
6주차-강의

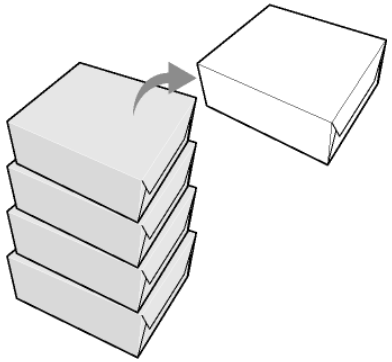
남춘성

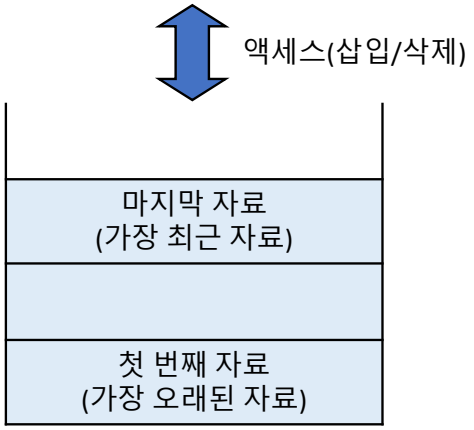
SUNG KYUN KWAN UNIVERSITY

스택(Stack)의 개념적 이해 - I

- 접시를 쌓듯이 자료를 차곡차곡 쌓아 올린 형태의 자료구조
- 스택에 저장된 원소는 top으로 정한 곳에서만 접근 가능
  - Top의 위치에서만 원소를 삽입하므로, 먼저 삽입한 원소는 밑에 쌓이고, 나중에 삽입한 원소는 위에 쌓이는 구조
  - 마지막에 삽입(**L**ast-**I**n)한 원소는 맨 위에 쌓여 있다가 가장 먼저 삭제(**F**irst-**O**ut)
  - **후입선출 구조** (**LIFO**, Last-In-First-Out)



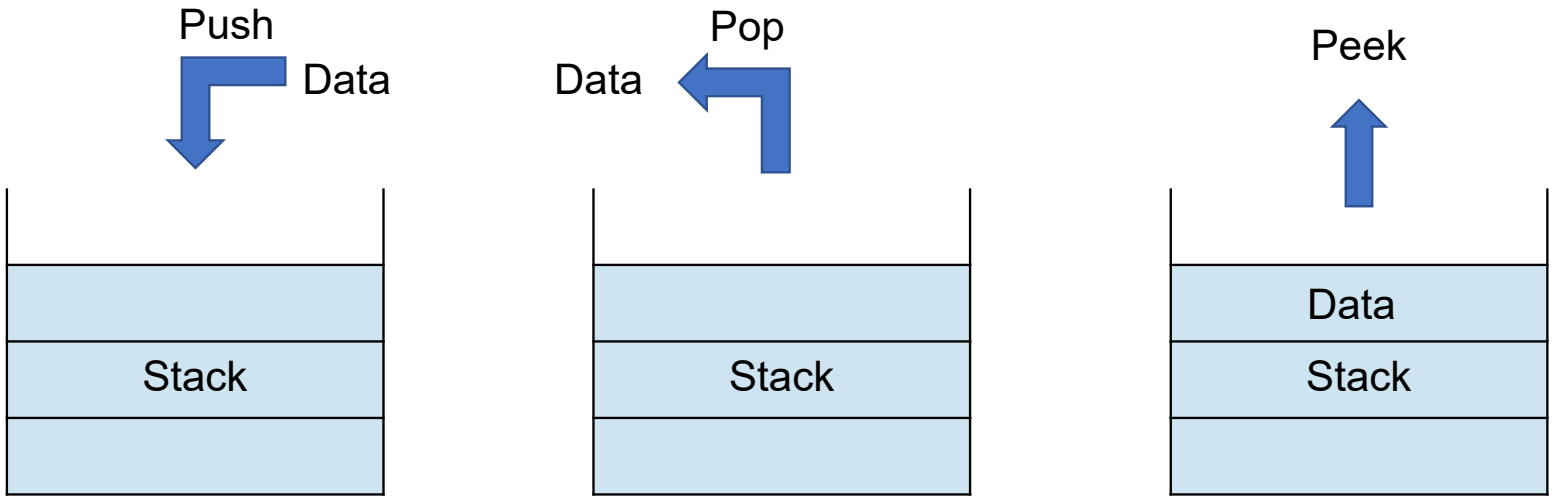




스택(Stack)의 개념적 이해 - II

SUNG KYUN KWAN UNIVERSITY

- **스택(Stack)의 연산**
  - 스택(Stack)에서의 삽입 연산 : push : 데이터를 넣음
  - 스택(Stack)에서의 삭제 연산 : pop : 데이터를 빼냄
  - 스택(Stack)에서의 피크 연산 : peek : 데이터를 봄
  - 스택(Stack)에서의 통이 비움 : ??? : 통이 차거나 비움



스택(stack)의 ADT 정의

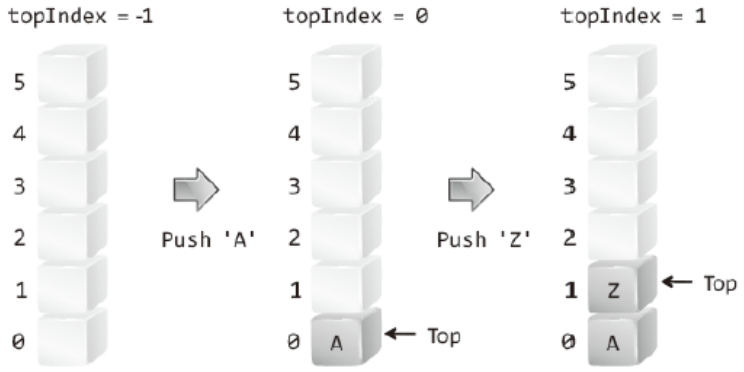
SUNG KYUN KWAN UNIVERSITY

- Void StackInit(Stack * pstack)
  - 스택의 초기화 진행
  - 스택 생성 후 제일 먼저 호출되어야 하는 함수
- Int SIsEmpty(Stack * pstack)
  - 스택이 빈 경우 True(1)을, 그렇지 않은 경우 False(0)을 반환
- Void Spush(Stack * pstack, Data data)
  - 스택에 데이터 저장, 매개변수 data로 전달된 값을 저장
- Data Spop(Stack * pstack)
  - 마지막에 저장된 요소를 삭제
  - 삭제된 데이터를 반환
  - 적어도 하나 이상의 데이터가 존재했을 경우 실행
- Data Speek(Stack * pstack)
  - 마지막에 저장된 요소를 반환하지만 삭제는 하지 않음
  - 적어도 하나 이상의 데이터가 존재했을 경우 실행

스택(stack)의 구현 방법 - 배열

SUNG KYUN KWAN UNIVERSITY

- 자료구조인 배열을 이용하여 구현
  - 스택의 크기 : 배열의 크기
  - 스택에 저장된 원소의 순서 : 배열 원소의 인덱스
    - ✓인덱스 0번 : 스택의 첫번째 원소
    - ✓인덱스 n-1번 : 스택의 n번째 원소
  - 변수 top : 스택에 저장된 마지막 원소에 대한 인덱스 저장
    - ✓공백 상태 : top = -1 (초기값)
    - ✓포화 상태 : top = n-1
  - Push
    - Top을 한 칸 올리고, top이 가리키는 위치에 데이터 저장
  - Pop
    - Top이 가리키는 데이터 반환, top을 아래라 한 칸 내림



스택(stack)의 헤더파일

SUNG KYUN KWAN UNIVERSITY

```
#define TRUE            1
#define FALSE           0
#define STACK_LEN       100

Typedef int Data;

Typedef struct _arrayStack {
        Data stackArr[STACK_LEN];     // 스택의 구조체
        int topIndex;
} ArrayStack;

Typedef ArrayStack Stack;

Void StackInit(Stack * pstack);
Int SIsEmpty(Stack * pstack);
Void Spush(Stack * pstack, Data data);
Dtat Spop(Stack * pstack);
Data Speek(Stack * pstack);
```

스택(stack)의 Init, Empty, Push

SUNG KYUN KWAN UNIVERSITY

```
Void StackInit(Stack * pstack) {
	pstack -> topIndex = -1;  //-1은 스택이 비었음을 의미
}
```

```
Int SIsEmpty(Stack * pstack) {
	if (pstack->topIndex == -1)
		return TRUE:
	else
		return FALSE;
}
```

```
Void Spush(Stack * pstack, Data data) {
	pstack->topIndex += 1;
	pstack ->stackArr[pstack->topIndex] = data;
}
```

스택(stack)의 Push, Pop, Peek

SUNG KYUN KWAN UNIVERSITY

```
Data Spop (Stack * pstack){
	int rIdx;
	if(SIsEmpty(pstack)) {
		printf("Stack memory Error !");
		exit(-1);
	}

	rIdx = pstack->topIndex;
	pstack->topIndex -=1;

	return pstack->stackArr[rIdx];
}
```

```
Data Speek(Stack * pstack) {
	if(SIsEmpty(pstack)) {
		printf("Stacm Memory Error!");
		exit(-1);
	}
	return pstack->stackArr[pstack->topIndex];
}
```

스택(stack)의 main

SUNG KYUN KWAN UNIVERSITY

```
int main(void) {

	//Stack의 생성 및 초기화
	Stack stack;

	StackInit(&stack);

	//데이터 넣기
	SPush(&stack, 1);
	SPush(&stack, 2);
	SPush(&stack, 3);
	SPush(&stack, 4);
	SPush(&stack, 5);

	//데이터 꺼내기
	while (!SIsEmpty(&stack))
		printf("%d ", SPop(&stack));

	return 0;
}
```

5 4 3 2 1 출력결과

스택(stack)의 구현 방법 - 연결리스트 I

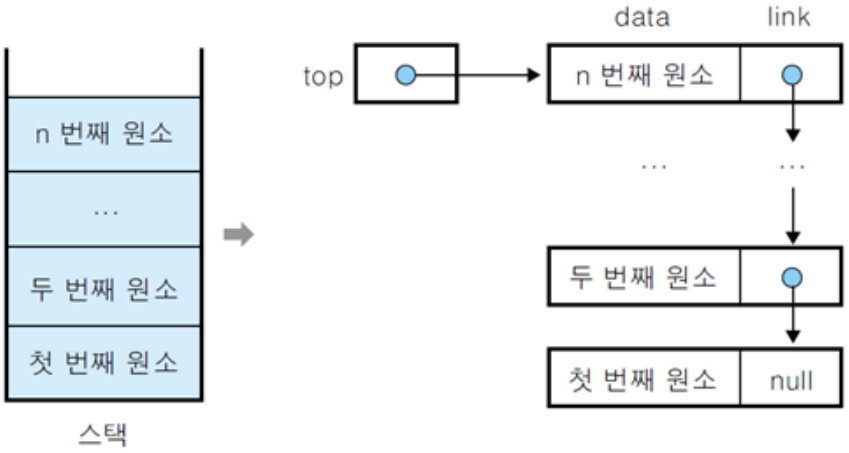
SUNG KYUN KWAN UNIVERSITY

- 순차 자료구조로 구현한 스택의 장점
  - 순차 자료구조인 1차원 배열을 사용하여 쉽게 구현

- 순차 자료구조로 구현한 스택의 단점
  - 물리적으로 크기가 고정된 배열을 사용하므로 스택의 크기 변경 어려움
  - 순차 자료구조의 단점을 그대로 가짐

스택(stack)의 구현 방법 - 연결리스트 II

- 연결 자료구조를 이용한 스택의 구현
  - 단순 연결 리스트를 이용하여 구현
    - 스택의 원소 : 단순 연결 리스트의 노드
      - 스택 원소의 순서 : 노드의 링크 포인터로 연결
      - push : 리스트의 마지막에 노드 삽입
      - pop : 리스트의 마지막 노드 삭제
    - 변수 top : 단순 연결 리스트의 마지막 노드를 가리키는 포인터 변수
      - 초기 상태 : top = null



스택(stack)의 헤더파일

SUNG KYUN KWAN UNIVERSITY

```
typedef int Data;

typedef struct _node {
	Data data;
	struct _node * next;
}Node;

typedef struct _listStack {
	Node * head;
}ListStack;

typedef ListStack Stack;

void StackInit(Stack * pstack);
int SIsEmpty(Stack* pstack);
void SPush(Stack * pstack, Data data);
Data SPop(Stack * pstack);
Data SPeek(Stack * pstack);
```

스택의 구현 방법 :
저장된 순서의 역순으로 데이터(노드)를 참조(삭제)하는
연결 리스트가 바로 연결리스트 기반의 스택이다

스택(stack)의 Init, Empty, Push

SUNG KYUN KWAN UNIVERSITY

```
void StackInit(Stack * pstack) {
	pstack->head = NULL;
}
```

```
int SIsEmpty(Stack* pstack) {
	if (pstack->head == NULL)
		return TRUE;
	else
		return FALSE;
}
```

```
void SPush(Stack * pstack, Data data) {
	Node * newNode = (Node*)malloc(sizeof(Node));
	newNode->data = data;

	newNode->next = pstack->head;
	pstack->head = newNode;
}
```

스택(stack)의 Pop, Peek

SUNG KYUN KWAN UNIVERSITY

```
Data SPop(Stack * pstack) {
	Data rdata;
	Node * rnode;

	if (SIsEmpty(pstack)) {
		printf("Stack memory is emplty");
		exit(-1);
	}

	rdata = pstack->head->data;
	rnode = pstack->head;

	pstack->head = pstack->head->next;
	free(rnode);

	return rdata;
}
```

```
Data SPeek(Stack * pstack) {
	if (SIsEmpty(pstack)) {
		printf("Stack memory is emplty");
		exit(-1);
	}

	return pstack->head->data;
}
```

스택(stack)의 main함수

SUNG KYUN KWAN UNIVERSITY

```
int main(void) {
	//Stack의 생성 및 초기화
	Stack stack;
	StackInit(&stack);

	//데이터 넣기
	SPush(&stack, 1);
	SPush(&stack, 2);
	SPush(&stack, 3);
	SPush(&stack, 4);
	SPush(&stack, 5);

	//데이터 꺼내기
	while (!SIsEmpty(&stack))
		printf("%d ", SPop(&stack));

	return 0;
}
```

5 4 3 2 1 출력결과

실습 - 1 : 수식의 괄호 검사

SUNG KYUN KWAN UNIVERSITY

- 수식의 괄호 검사
  - 수식에 포함되어있는 괄호는 가장 마지막에 열린 괄호를 가장 먼저 닫아 주어야 하는 후입선출 구조로 구성되어있으므로, 후입선출 구조의 스택을 이용하여 괄호를 검사한다.
  - 수식을 왼쪽에서 오른쪽으로 하나씩 읽으면서 괄호 검사

> ① 왼쪽 괄호를 만나면 스택에 **push**
>
> ② 오른쪽 괄호를 만나면 스택을 **pop**하여 마지막에 저장한 괄호와 같은 종류인지를 확인
>
> - 같은 종류의 괄호가 아닌 경우 괄호의 짝이 잘못 사용된 수식임.

  - 수식에 대한 검사가 모두 끝났을 때 스택은 공백 스택이 됨
    - 수식이 끝났어도 스택이 공백이 되지 않으면 괄호의 개수가 틀린 수식임.

예) (1+1) 이면 True, ((3+1)+(4*2) 는 False

스택(stack)을 이용한 계산기 프로그램 이해

SUNG KYUN KWAN UNIVERSITY

- 수식 계산은 다음 문장을 통해 계산
  - (3+4)*(5/2)+(7+(9-5))

- 수식 계산을 위한 고려사항
  - 소괄호를 파악하여 그 부분을 먼저 연산
  - 연산자 우선순위를 근거로 연산의 순위를 결정

세가지 수식 표기법 : 전위, 중위, 후위

SUNG KYUN KWAN UNIVERSITY

- 중위 표기법(infix notation) : **연산자를 피연산자 가운데 표기**
  - 수식 내에 연산의 순서에 대한 정보가 담겨있지 않음
  - 소괄호와 연산의 우선 순위라는 것을 정의하여 이를 기반으로 연산의 순서를 정함
    - 예) 5 + 2 / 7
- 전위 표기법(prefix notation) : **연산자를 피연산자의 앞에 표기**
  - 수식 내에 연산의 순서가 있음
  - 소괄호가 필요 없고 연산의 우선 순위를 결정할 필요도 없음
    - 예) +5 / 2 7
- 후위 표기법(postfix notation) : **연산자를 피연산자 두에 표기**
  - 전위 표기법과 마찬가지로 수식 내 연산의 정보가 있음
  - 소괄호가 필요 없고 연산의 순위를 결정할 필요 없음
    - 예) 5 2 7 / +

중위에서 후위 : 소괄호 고려하지 않음 I

수식 왼쪽부터 시작해서 처리

피 연산자를 만나면 무조건 반환된 수식이 위치할 자리로 이동

연산자를 만나면 무조건 쟁반으로 이동

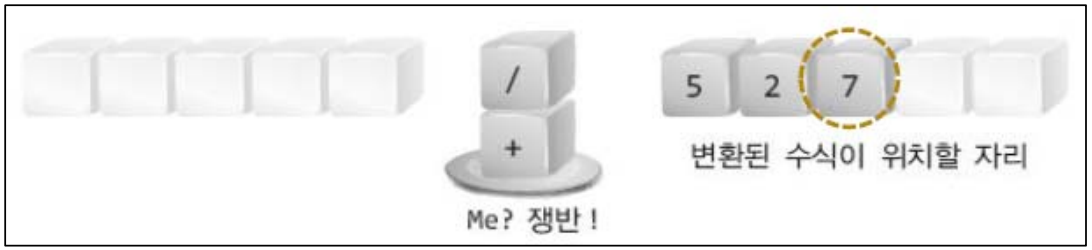
숫자를 만나면 변환된 수식이 위치할 자리로 이동

중위에서 후위 : 소괄호 고려하지 않음 II

/ 연산자의 우선순위가 높으므로 + 연산자 위에 올림

- 쟁반에 위치한 연산자의 우선순위가 높으면
  - 쟁반에 위치한 연산자를 꺼내 변환된 수식이 위치할 자리로 옮김
  - 새 연산자는 쟁반으로 옮김
- 쟁반에 위치한 연산자의 우선순위가 낮으면
  - 쟁반에 위치한 연산자 위에 새 연산자를 쌓음

우선 순위가 높은 연산자는 우선순위가 낮은 연산자 위에 올라서서, 우선순위가 낮은 연산자가 먼저 자리를 잡지 못하게(계산되지 못하게) 하려는 목적



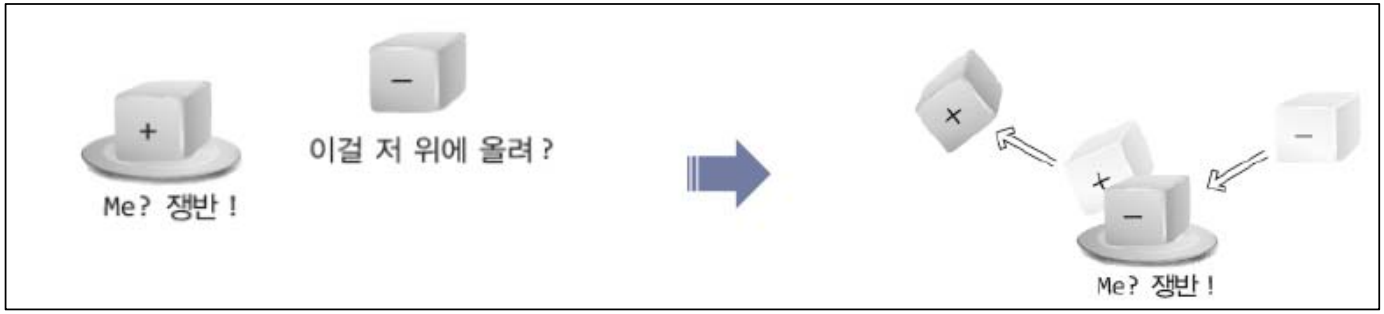
피 연산자는 무조건 변환된 수식의 자리로 이동

중위에서 후위 : 소괄호 고려하지 않음 III

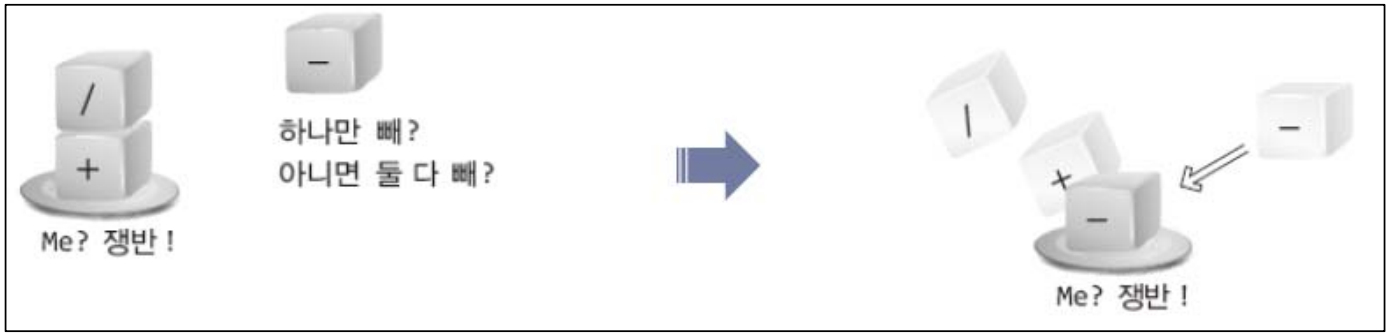
나머지 연산자들은 쟁반에서 차례로 옮김

- 변환 규칙 정리
  - 피연산자는 그냥 옮김
  - 연산자는 쟁반으로 옮김
  - 연산자가 쟁반에 있다면 우선순위를 비교하여 처리방법을 결정
  - 마지막까지 쟁반에 남아있는 연산자들은 하나씩 꺼내서 옮김

중위에서 후위 : 소괄호 고려하지 않음 IV



우선순위가 같은 경우 : 먼저 등장한 연산자를 먼저 진행하는 것이 나음
즉, + 연산자를 먼저 처리하고(옮기고) –를 처리(쟁반에 넣음)

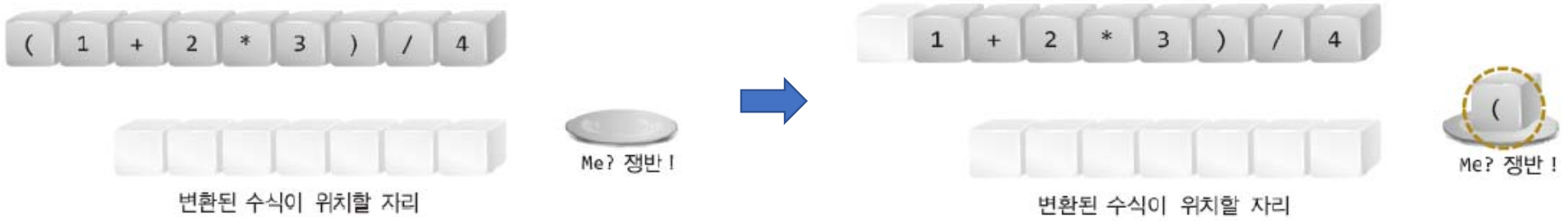


/ 혹은 /+ 를 빼야 하나? 역시나 같은 원리로 /와 +가 우선순위가 높다고 생각하여 진행함

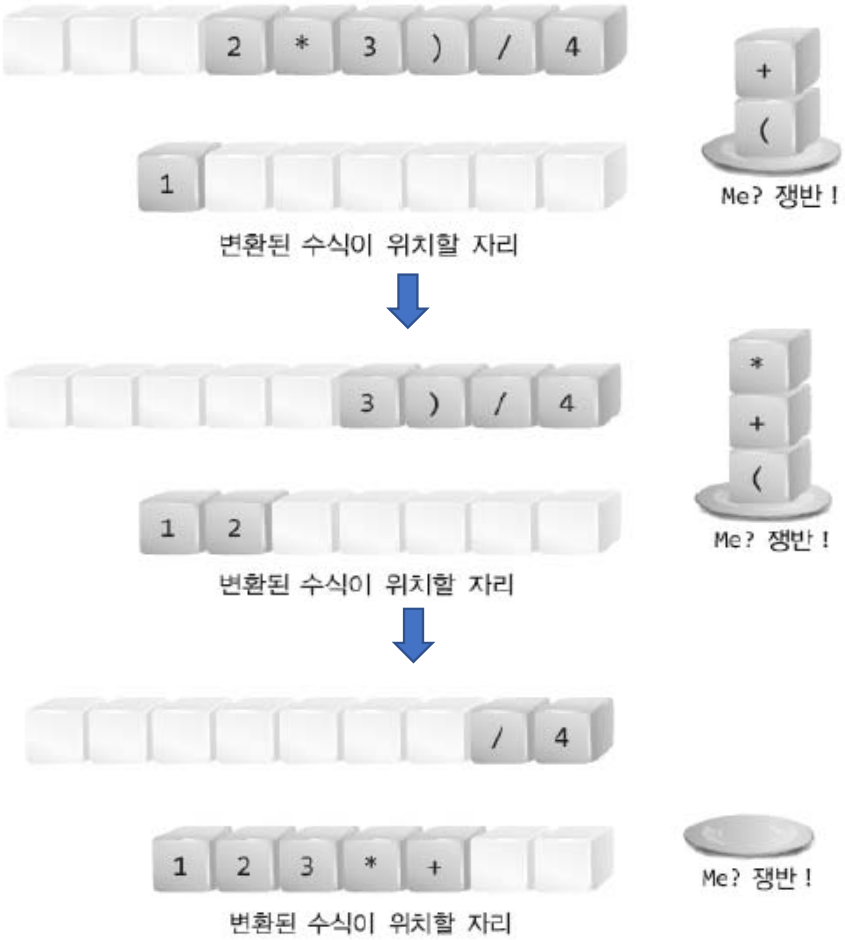
중위에서 후위 : 소괄호 고려함 I

소괄호 안에 있는 연산자들이 후위 표기법의 수식에서 앞부분에 위치해야 함
→ 후위 표기법의 수식에서 먼저 연산이 이뤄져야 하는 연산자가 뒤에 있는 연산이 이뤄지는 연산자보다 앞에 위치해야 함.

‘(‘ 연산자의 우선순위는 사칙연산 어느 연산자들보다 낮다고 간주해야 함
→ ‘)’ 연산자가 나올때까지 소괄호의 경계를 구분하는 도구로 사용해야 하기 때문



중위에서 후위 : 소괄호 고려함 II



즉, ')' 의 의미는 '('이후에 쌓인 연산자들을 변환된 수식의 자리로 옮기라는 것임. 따라서 ')' 연산자는 변환된 수식의 자리로 옮기지 않아도 됨

')' 연산자를 만나면 쟁반에서 '('연산자 만날 때까지 연산자를 반환된 수식의 자리로 옮김

중위에서 후위 변환 프로그램 구현 - I

SUNG KYUN KWAN UNIVERSITY

stack header 파일 동일

```
typedef int Data;

typedef struct _node {
	Data data;
	struct _node * next;
}Node;

typedef struct _listStack {
	Node * head;
}ListStack;

typedef ListStack Stack;

void StackInit(Stack * pstack);
int SIsEmpty(Stack* pstack);
void SPush(Stack * pstack, Data data);
Data SPop(Stack * pstack);
Data SPeek(Stack * pstack);
```

ConvToRPNExp 만 작성(RPN : Reverse Polish Notation)

```
Void ConvToRPNExp(char exp[]) {
	…
}

int main(void) {
	char exp[] =“3-2+4”;  //중위표기법 인자
	ConvToRPNExp(exp); //호출후 출력
	…
}
```

중위에서 후위 변환 프로그램 구현 - II

연산자 우선순위 알아내는 기능 설정 : int GetOpPrec(char op)

```
int GetOpPrec(char op){//연산자 우선 순위 정보를 반환
	switch (op){
		case '*':
		case '/':
			return 5;
		case '+':
		case '-':
			return 3;
		case '(':		// ')'연산자가 등장할 때까지 가장 밑에 남아 있어야 하기 때문에 우선순위가 낮음
			return 1;
	}

	return -1;//등록되지 않은 연산자임

}
```

‘)’ 연산자는 쟁반으로 가지 않음(정의될 필요가 없음)

중위에서 후위 변환 프로그램 구현 - III

SUNG KYUN KWAN UNIVERSITY

연산자 비교결과에 대한 값을 반환 :int WhoPrecOp(char op1, char op2)

```
int WhoPrecOp(char op1, char op2) {          // 두 연산자의 우선순위 비교 결과를 반환

    int op1Prec = GetOpPrec(op1);
    int op2Prec = GetOpPrec(op2);

    if (op1Prec > op2Prec)                   // op1의 우선순위가 더 높다면 1
        return 1;
    else if (op1Prec < op2Prec)              // op2의 우선순위가 더 높다면-1
        return -1;
    else                                     // op1과 op2의 우선순위가 같다면 0
        return 0;
}
```

중위에서 후위 변환 프로그램 구현 - IV

SUNG KYUN KWAN UNIVERSITY

중위를 후위로 변환:void ConvToRPNExp(char exp[]) {

```
void ConvToRPNExp(char exp[]){
	Stack stack;
	int expLen = strlen(exp);
	char * convExp = (char*)malloc(expLen + 1);	//수식을 담을 공간 마련(EOF를 위해 1개 더 생성)
	int i, idx = 0;					//i : for를 위한, idx : 변환된 식에 넣을 위치
	char tok;					//문자열에 있는  char
	char popOp;					//stack에서 빼낼 char(수식)
	memset(convExp, 0, sizeof(char)*expLen + 1);	//마련한 공간 0으로 초기화
	StackInit(&stack);

	//과정 : 숫자면 convExp에 저장 아니면(수식:괄호포함) Push할지 PoP할지를 결정
	for (i = 0; i < expLen; i++) {...}

	while (!SIsEmpty(&stack))			// tok이 다 빠진후에 나머지 stack에 있는 수식을 빼냄
		convExp[idx++] = SPop(&stack);

	strcpy(exp, convExp);				//exp에 convExp를 복사
	free(convExp);					//convExp를 제거
}
```

중위에서 후위 변환 프로그램 구현 - V

중위를 후위로 변환:void ConvToRPNExp(char exp[]) {

```
void ConvToRPNExp(char exp[]){
	…
	//과정 : 숫자면 convExp에 저장 아니면(수식:괄호포함) Push할지 PoP할지를 결정
	for (i = 0; i < expLen; i++) {
		tok = exp[i];
		if (isdigit(tok))			//피연산자(숫자)이기 때문에 convExp에 저장
		{
			convExp[idx++] = tok;
		}
		else{					// 수식이면 다음의 경우에 따라
			switch (tok){
			…
		}
	…
}
```

중위에서 후위 변환 프로그램 구현 - VI

SUNG KYUN KWAN UNIVERSITY

중위를 후위로 변환:void ConvToRPNExp(char exp[]) {

```
void ConvToRPNExp(char exp[]){
        …
                switch (tok){
                        case '(':                               // (가 나오면 무조건 push 시킴.
                                SPush(&stack, tok);
                                break;
                        case ')':                               // ')'가 나오면 '('가 나올때 까지 모두 Pop해서 convExp에 저장
                                while (1){
                                        popOp = SPop(&stack);
                                        if (popOp == '(')
                                                break;
                                        convExp[idx++] = popOp;
                                }
                                break;
                        case '+':       case '-':       case '*':       case '/':
        //사칙연산이면 stack, 비워있으면 Push
        //                      비워있지 않으면 연산자우선순위를 봐서(현재 맨위-Peek과 tok을 비교해서 0보다 크면 우선순위가 크면
        //                      먼저처리되어야 함으로 Pop해서 Peek을 제거하고이를 convExp에 넣고, tok를 다시 Push)
                        while (!SIsEmpty(&stack) && WhoPrecOp(SPeek(&stack), tok) >= 0)
                                convExp[idx++] = SPop(&stack);
                        SPush(&stack, tok);
                        break;
                }
…. }
```

중위에서 후위 변환 프로그램 구현 - VII

Main함수에서 이를 실행

```
int main(void){
	char exp1[] = "1+2*3";
	char exp2[] = "(1+2)*3";
	char exp3[] = "((1-2)+3)*(5-2)";

	ConvToRPNExp(exp1);
	ConvToRPNExp(exp2);
	ConvToRPNExp(exp3);

	printf("%s \n", exp1);
	printf("%s \n", exp2);
	printf("%s \n", exp3);

return 0;
}
```

InfixToPostfix.h
InfixToPostfix.c

ListBaseStack.h
ListBaseStack.c

InfixToPostfixMain.c

```
123*+
12+3*
12-3+52-*
```

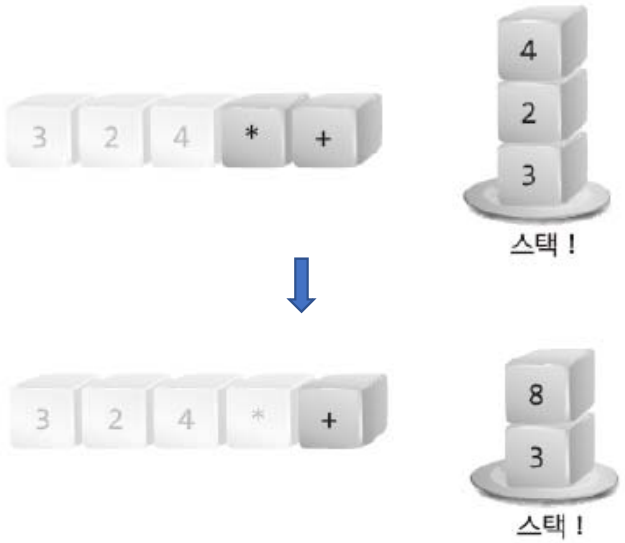
후위 표기법 수식 계산 방법 - I

SUNG KYUN KWAN UNIVERSITY

피연산자 두 개가 연산자 앞에 항상 위치하는 구조 확인

3 + 2 * 4	(1*2+3) /4
↓	↓
3 2 4 * +	1 2 * 3 + 4/
↓	↓
3 2 4 * +	2 3 + 4/
↓	↓
3 8 +	5 4 /
↓	↓
11	1.25

- 피연산자는 무조건 스택으로 옮김
- 연산자를 만나면 스택에서 두 개의 피연산자를 꺼내서 계산함
- 계산 결과는 다시 스택에 넣음



후위 표기법 수식 계산 방법 - II

SUNG KYUN KWAN UNIVERSITY

후위 계산 :int EvalRPNExp(char exp[]) {

```
int EvalRPNExp(char exp[]){

        Stack stack;                    //stack 생성
        int explen = strlen(exp);       // 후위식 exp의 길이
        int i;                          // for
        char tok, op1, op2;             // exp의 내용을 담을 tok, 두 수를 담을 op1, op2

        StackInit(&stack);              // stack 초기화

        for (i = 0; i < explen; i++) { ...
        }

        return SPop(&stack);      // 3. 최종적으로 계산된 값이 Stack에 남아있으므로 이를 꺼내서(Pop)하여 확인

}
```

후위 표기법 수식 계산 방법 - III

SUNG KYUN KWAN UNIVERSITY

후위 계산 :int EvalRPNExp(char exp[]) {

```
int EvalRPNExp(char exp[]){

for (i = 0; i < explen; i++) {
        tok = exp[i];
        if (isdigit(tok)) {                     // 1. 만약 stack에 저장된 값이 정수이면 다른 stack에 Push
                SPush(&stack, tok - '0');       // char형인 tok 값에 -'0'을 통해 정수화 함.
        } else {                                // 2. 만약 연산자라면 op를 2개 꺼내서 계산하고 Push
                op2 = SPop(&stack);
                op1 = SPop(&stack);

                switch (tok) {
                        case '+':
                                SPush(&stack, op1 + op2);
                                break;
                        case '-':       …
                        case '*':       …
                        case '/':       …
                }
        }
        return SPop(&stack);      // 3. 최종적으로 계산된 값이 Stack에 남아있으므로 이를 꺼내서(Pop)하여 확인
}
```

후위 표기법 수식 계산 방법 - IV

Main 함수

```
int main(void) {
	char exp1[] = "1+2*3";
	char exp2[] = "(1+2)*3";
	char exp3[] = "((1-2)+3)*(5-2)";

	printf("%s = %d \n", exp1, EvalInfixExp(exp1));
	printf("%s = %d \n", exp2, EvalInfixExp(exp2));
	printf("%s = %d \n", exp3, EvalInfixExp(exp3));

	return 0;
}
```

PostCalculator.h
PostCalculator.c

ListBaseStack.h
ListBaseStack.c

PostCalculatorMain.c

```
42*8+ = 16
123+*4/ = 1
```

후위 변환과 후위 수식계산 통합

SUNG KYUN KWAN UNIVERSITY

중위 표기법 수식 → ConvToRPNExp → EvalRPNExp → 연산결과

ListBaseStack.h, ListBaseStack.c : 스택
InfixToPostfix.h, InfixToPostfix.c : 후위표기법 변환
PostCalculator.h, PostCalculator.c : 후위 표기법 수식 계산
InfixCalculator.h, InfixCalculator.c : 중위표기법 수식 계산
InfixCalculatorMain.c

위와 같이 구성함