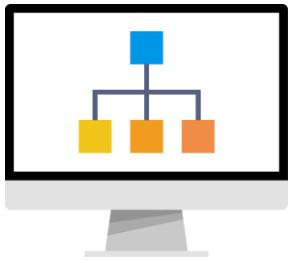

Data Structure & Algorithm
자료구조 및 알고리즘

11. 스택 (Stack Part 2, Chapter 6)

8강 보고서 돌아보기

연결 리스트는 배열과 달리 *i*번째 노드의 데이터를 가져오는 인터페이스가 없다. 오늘 배운 함수들을 활용하여 리스트 plist의 *i*번째 데이터를 반환하는 함수 LData LIndex(List *plist, int i)를 의사 코드로 나타내 보세요.

- 비효율적이더라도 괜찮음.

8강 보고서 돌아보기

어떤 연결 리스트 L1의 순서를 뒤집은 L2를 만들고 싶다.
어떻게 하면 될까?

- 비효율적이더라도 괜찮음.
- 정렬 규칙을 사용하지 않는다고 가정

8강 보고서 돌아보기

어떤 연결 리스트 L1의 순서를 뒤집은 L2를 만들고 싶다.
어떻게 하면 될까?

- 비효율적이더라도 괜찮음.
- 정렬 규칙을 사용하지 않는다고 가정

9강 보고서 돌아보기

이중 연결 리스트에서 삭제는 어떻게 구현하면 될 지 LRemove 함수를 의사 코드로 표현해 보세요. (가능하다면, C 코드도 괜찮음)

- head 포인터가 가리키고 있는 노드가 지워질 때 head 포인터를 바꿔야 함에 주의
- 삭제 후 앞, 뒤 노드를 서로 연결해야 함에 주의

계산기 프로그램 구현

계산기 프로그램

다음과 같은 문장의 수식을 계산할 수 있어야 한다.

(3 + 4) * (5 / 2) + (7 + (9 – 5))

고려할 것

- 소괄호를 파악하여 그 부분을 먼저 연산한다.
- 연산자의 우선순위를 근거로 연산의 순위를 결정한다.

제약 사항

- 숫자는 0-9 한자리 수라고 가정
- 연산자는 사칙연산으로 한정하되, - 연산의 경우 항상 이항연산이라고 가정 (binary)
- 항상 올바른 수식만 입력된다고 가정 (괄호 짝, 0으로 나누기)

세 가지 수식의 표기법: 전위, 중위, 후위

- 중위 표기법(infix notation) 예) 5 + 2 / 7

  수식 내에 연산의 순서에 대한 정보가 담겨 있지 않다. 그래서 소괄호와 연산자의 우선순위라는 것을 정의하여 이를 기반으로 연산의 순서를 명시한다.

- 전위 표기법(prefix notation) 예) + 5 / 2 7

  수식 내에 연산의 순서에 대한 정보가 담겨 있다. 그래서 소괄호가 필요 없고 연산의 우선순위를 결정할 필요도 없다.

- 후위 표기법(postfix notation) 예) 5 2 7 / +

  전위 표기법과 마찬가지로 수식 내에 연산의 순서에 대한 정보가 담겨 있다. 그래서 소괄호가 필요 없고 연산의 우선순위를 결정할 필요도 없다.

소괄호와 연산자의 우선순위를 인식하게 하여 중위 표기법의 수식을 직접 계산하게 프로그래밍 하는 것보다 후위 표기법의 수식을 계산하도록 프로그래밍 하는 것이 더 쉽다!

중위 → 후위 :
소괄호 고려하지 않고 1

▶ [그림 06-3: 수식 변환의 과정 1/7]

수식을 이루는 왼쪽 문자부터 시작해서 하나씩 처리해 나간다.

▶ [그림 06-4: 수식 변환의 과정 2/7]

피 연산자를 만나면 무조건 변환된 수식이 위치할 자리로 이동시킨다.

중위 → 후위 : 소괄호 고려하지 않고 2

▶ [그림 06-5: 수식 변환의 과정 3/7]

연산자를 만나면 무조건 쟁반으로 이동한다.

▶ [그림 06-6: 수식 변환의 과정 4/7]

숫자를 만났으니 변환된 수식이 위치할 자리로 이동!

중위 → 후위 :
소괄호 고려하지 않고 3

/ 연산자의 우선순위가 높으므로 + 연산자 위에 올린다.

▶ [그림 06-7: 수식 변환의 과정 5/7]

쟁반에 기존 연산자가 있는 상황에서의 행동 방식!

☞ 쟁반에 위치한 연산자의 우선순위가 높다면

- 쟁반에 위치한 연산자를 꺼내서 변환된 수식이 위치할 자리로 옮긴다.
- 그리고 새 연산자는 쟁반으로 옮긴다.

☞ 쟁반에 위치한 연산자의 우선순위가 낮다면

- 쟁반에 위치한 연산자의 위에 새 연산자를 쌓는다.

우선순위가 높은 연산자는 우선순위가 낮은 연산자 위에 올라서서, 우선순위가 낮은 연산자가 먼저 자리를 잡지 못하게 하려는 목적!

중위 → 후위 :
소괄호 고려하지 않고 4

▶ [그림 06-8: 수식 변환의 과정 6/7]

피 연산자는 무조건 변환된 수식의 자리로 이동!

▶ [그림 06-9: 수식 변환의 과정 7/7]

나머지 연산자들은 쟁반에서 차례로 옮긴다!

중위 → 후위 : 정리하면

변환 규칙의 정리 내용

- 피 연산자는 그냥 옮긴다.
- 연산자는 쟁반으로 옮긴다.
- 연산자가 쟁반에 있다면 우선순위를 비교하여 처리방법을 결정한다.
- 마지막까지 쟁반에 남아있는 연산자들은 하나씩 꺼내서 옮긴다.

중위 → 후위 : 고민 될 수 있는 상황

상황 1

이걸 저 위에 올려 ?

▶ [그림 06-10: 우선순위가 같은 경우]

+ 연산자가 우선순위가 높다고 가정하고(+ 연산자가 먼저 등장했으므로) 일을 진행한다.

즉 + 연산자를 옮기고 그 자리에 – 연산자를 가져다 놔야 한다."

상황 2

하나만 빼?
아니면 둘 다 빼?

▶ [그림 06-12: 둘 이상의 연산자가 쌓여 있는 경우]

중위 → 후위 : 소괄호 고려 1

후위 표기법의 수식에서는 먼저 연산이 이뤄져야 하는 연산자가 뒤에 연산이 이뤄지는 연산자보다 앞에 위치해야 한다. 따라서 소괄호 안에 있는 연산자들이 후위 표기법의 수식에서 앞부분에 위치해야 한다.

(연산자의 우선순위는 그 어떤 사칙 연산자들보다 낮다고 간주! 그래서) 연산자가 등장할 때까지 쟁반에 남아 소괄호의 경계 역할을 해야 함!

▶ [그림 06-14: 소괄호가 포함된 수식의 변환 1/6]

▶ [그림 06-15: 소괄호가 포함된 수식의 변환 2/6]

중위 → 후위 : 소괄호 고려 2

▶ [그림 06-16: 소괄호가 포함된 수식의 변환 3/6]

▶ [그림 06-17: 소괄호가 포함된 수식의 변환 4/6]

▶ [그림 06-18: 소괄호가 포함된 수식의 변환 5/6]

) 연산자를 만나면 쟁반에서 (연산자 만날 때까지 연산자를 변환된 수식의 자리로 옮긴다!

중위 → 후위 : 소괄호 고려 3

▶ [그림 06-19: 소괄호가 포함된 수식의 변환 6/6]

조금 달리 설명하면 (연산자는 쟁반의 또 다른 바닥이다! 그리고) 연산자는 변환이 되어야 하는 작은 수식의 끝을 의미한다. 그래서) 연산자를 만나면 (연산자를 만날 때까지 연산자를 이동시켜야 한다.

지금까지 설명한 내용에 해당하는 코드의 구현에 앞서 중위 표기법의 수식을 후위 표기법의 수식으로 바꾸는 연습을 할 필요가 있다.

중위 → 후위 : 프로그램 구현 1

```
void ConvToRPNExp(char exp[])
{
    . . . .
}
```

변환 함수의 타입

함수 이름의 일부인 RPN은 후위 표기법의 또 다른 이름인 Reverse Polish Notation의 약자이다.

```
int main(void)
{
    char exp[] = "3-2+4";
    ConvToRPNExp(exp);
    . . . .
}
```

중위 표기법 수식을 배열에 담아 함수의 인자로 전달한다.

호출 완료 후 exp에는 변환된 수식이 담긴다.

중위 → 후위 : 프로그램 구현 2

함수 ConvToRPNExp의 첫 번째 helper function!

```
int GetOpPrec(char op)        // 연산자의 연산 우선순위 정보를 반환한다.
{
    switch(op)
    {
    case '*':
    case '/':
        return 5;             // 가장 높은 연산의 우선순위
    case '+':
    case '-':
        return 3;
    case '(':
        return 1;
    }

    return -1;                // 등록되지 않은 연산자임을 알림!
}
```

연산자의 우선순위에 대응하는 값을 반환한다. 값이 클수록 우선순위가 높은 것으로 정의되어 있다.

(연산자는) 연산자가 등장할 때까지 쟁반에 남아 있어야 하기 때문에 가장 낮은 우선순위를 부여!

) 연산자는 소괄호의 끝을 알리는 메시지의 역할을 한다. 따라서 쟁반으로 가지 않는다.
때문에) 연산자에 대한 반환 값은 정의되어 있을 필요가 없다!

중위 → 후위 : 프로그램 구현 3

함수 ConvToRPNExp의 두 번째 helper function!

두 연산자의 우선순위 비교 결과를 반환한다.

```
int WhoPrecOp(char op1, char op2)
{
    int op1Prec = GetOpPrec(op1);
    int op2Prec = GetOpPrec(op2);

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

ConvToRPNExp 함수의 실질적인 Helper Function은 위의 함수 하나이다!

중위 → 후위 : 프로그램 구현 4

```
void ConvToRPNExp(char exp[])
{
    Stack stack;
    int expLen = strlen(exp);
    char * convExp = (char*)malloc(expLen+1);

    int i, idx=0;
    char tok, popOp;

    memset(convExp, 0, sizeof(char)*expLen+1);
    StackInit(&stack);

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

    while(!SIsEmpty(&stack))
        convExp[idx++] = SPop(&stack);

    strcpy(exp, convExp);
    free(convExp);
}
```

변환된 수식을 담을 공간 마련

마련한 공간 0으로 초기화

일련의 변환 과정을 이 반복문 안에서 수행

스택에 남아 있는 모든 연산자를 이동시키는 반복문

변환된 수식을 반환!

중위 → 후위 : 프로그램 구현 5

```
void ConvToRPNExp(char exp[])
{
    . . . .

    for(i=0; i<expLen; i++)
    {
        tok = exp[i];

        if(isdigit(tok))
        {
            convExp[idx++] = tok;
        }
        else
        {
            switch(tok)
            {
                . . . .
            }
        }
    }
}
```

tok에 저장된 문자가 피연산자라면

tok에 저장된 문자가 연산자라면

연산자일 때의 처리 루틴을 switch문에 담는다!

중위 → 후위 : 프로그램 구현 6

함수 ConvToRPNExp의 일부인 switch문

```
switch(tok)
{
case '(' :                  // 여는 소괄호라면,
    SPush(&stack, tok);     // 스택에 쌓는다.
    break;
case ')' :                  // 닫는 소괄호라면,
    while(1)                // 반복해서,
    {
        popOp = SPop(&stack);    // 스택에서 연산자를 꺼내어,
        if(popOp == '(')         // 연산자 ( 을 만날 때까지,
            break;
        convExp[idx++] = popOp;  // 배열 convExp에 저장한다.
    }
    break;
case '+': case '-':
case '*': case '/':
    while(!SIsEmpty(&stack) && WhoPrecOp(SPeek(&stack), tok) >= 0)
        convExp[idx++] = SPop(&stack);
    SPush(&stack, tok);
    break;
}
```

tok에 저장된 연산자를 스택에 저장하기 위한 과정

중위 → 후위 : 프로그램 구현 6

함수 ConvToRPNExp의 일부인 switch문

```
switch(tok)
{
case '(' :                  // 여는 소괄호라면,
    SPush(&stack, tok);     // 스택에 쌓는다.
    break;
case ')' :                  // 닫는 소괄호라면,
    while(1)                // 반복해서,
    {
        popOp = SPop(&stack);    // 스택에서 연산자를 꺼내어,
        if(popOp == '(')         // 연산자 ( 을 만날 때까지,
            break;
        convExp[idx++] = popOp;  // 배열 convExp에 저장한다.
    }
    break;
case '+': case '-':
case '*': case '/':
    while(!SIsEmpty(&stack) && WhoPrecOp(SPeek(&stack), tok) >= 0)
        convExp[idx++] = SPop(&stack);
    SPush(&stack, tok);
    break;
}
```

tok에 저장된 연산자를 스택에 저장하기 위한 과정

중위 → 후위 : 프로그램의 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
: ConvToRPNExp 함수의 선언과 정의

ListBaseStack.h
ListBaseStack.c
: 스택관련 함수의 선언과 정의

InfixToPostfixMain.c : main 함수의 정의

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

실행결과

후기 표기법 수식의 계산

피연산자 두 개가 연산자 앞에 항상 위치하는 구조

3 + 2 * 4

후위 표기법 수식으로

3 2 4 * +

3 2 4 * +

2와 4의 곱 진행

3 8 +

(1 * 2 + 3) / 4

후위 표기법 수식으로

1 2 * 3 + 4 /

1과 2의 곱 진행

2 3 + 4 /

2와 3의 합 진행

5 4 /

후기 표기법 수식 계산 프로그램의 구현

계산의 규칙

- 피연산자는 무조건 스택으로 옮긴다.
- 연산자를 만나면 스택에서 두 개의 피연산자를 꺼내서 계산을 한다.
- 계산결과는 다시 스택에 넣는다.

스택 !

▶ [그림 06-20: 후위 표기법의 수식 계산 1]

스택 !

▶ [그림 06-21 : 후위 표기법의 수식 계산 2]

후기 표기법 수식 계산 프로그램의 구현

```
int EvalRPNExp(char exp[])
{
    Stack stack;
    int expLen = strlen(exp);
    int i;
    char tok, op1, op2;

    StackInit(&stack);

    for(i=0; i<expLen; i++)
    {
        tok = exp[i];
        if(isdigit(tok)
        {
            SPush(&stack, tok - '0');
        }
        else
        {
            op2 = SPop(&stack);
            op1 = SPop(&stack);
            switch(tok)
            {
            case '+':
                SPush(&stack, op1+op2);
                break;
            case '-':
                SPush(&stack, op1-op2);
                break;
            case '*':
                SPush(&stack, op1*op2);
                break;
            case '/':
                SPush(&stack, op1/op2);
                break;
            }
        }
    }
    return SPop(&stack);
}
```

이어서

후위 표기법 수식 계산 프로그램의 실행

PostCalculator.h
PostCalculator.c
: EvalRPNExp 함수의 선언과 정의

ListBaseStack.h
ListBaseStack.c
: 스택관련 함수의 선언과 정의

PostCalculatorMain.c : main 함수의 정의

```
int main(void)
{
    char postExp1[] = "42*8+";
    char postExp2[] = "123+*4/";

    printf("%s = %d \n", postExp1, EvalRPNExp(postExp1));
    printf("%s = %d \n", postExp2, EvalRPNExp(postExp2));

    return 0;
}
```

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

실행결과

계산기 프로그램의 완성1

계산의 과정

중위 표기법 수식 → ConvToRPNExp → EvalRPNExp → 연산결과

계산기 프로그램의 파일 구성

• 스택의 활용	ListBaseStack.h, ListBaseStack.c
• 후위 표기법의 수식으로 변환	InfixToPostfix.h, InfixToPostfix.c
• 후위 표기법의 수식을 계산	PostCalculator.h, PostCalculator.c
• 중위 표기법의 수식을 계산	InfixCalculator.h, InfixCalculator.c
• main 함수	InfixCalculatorMain.c

InfixCalculator.h　　InfixCalculator.c

계산기 프로그램의 완성2

InfixCalculator.h

```
#ifndef __INFIX_CALCULATOR__
#define __INFIX_CALCULATOR__

int EvalInfixExp(char exp[]);

#endif
```

InfixCalculatorMain.c

```
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;
}
```

InfixCalculator.c

```
int EvalInfixExp(char exp[])
{
    int len = strlen(exp);
    int ret;
    char * expcpy = (char*)malloc(len+1);  // 문자열 저장공간 마련
    strcpy(expcpy, exp);                   // exp를 expcpy에 복사

    ConvToRPNExp(expcpy);      // 후위 표기법의 수식으로 변환
    ret = EvalRPNExp(expcpy);  // 변환된 수식의 계산

    free(expcpy);    // 문자열 저장공간 해제
    return ret;      // 계산결과 반환
}
```

실행결과

```
1+2*3 = 7
(1+2)*3 = 9
((1-2)+3)*(5-2) = 6
```

#ifndef와 #define을 왜?

InfixCalculator.h

```
#ifndef __INFIX_CALCULATOR__
#define __INFIX_CALCULATOR__

int EvalInfixExp(char exp[]);

#endif
```

aa.h

#include “InfixCalculator.h”

bb.h

#include “InfixCalculator.h”
#include “aa.h”

cc.c

#include “bb.h”

요약

- 전위표기법과 후위표기법은 괄호를 사용하지 않고 연산자 우선순위를 고려할 수 있다.

- 중위표기법을 후위표기법으로 바꾸기
  - 피연산자는 바로 출력
  - 연산자는 스택에 쌓되, 우선순위가 높은 연산자는 모두 pop하기
  - 여는 괄호는 스택에 쌓고, 닫는 괄호가 나올 시 사이의 연산자를 모두 pop

- 후위 표기법 계산하기
  - 피연산자를 스택에 쌓으면서 연산자를 만나면 상위 두 피연산자를 계산하고 다시 스택에 push

출석 인정을 위한 보고서 작성

- A4 반 장 이상으로 아래 질문에 답한 후 포털에 있는 과제 제출 란에 PDF로 제출

- 입력으로 주어진 수식에 세 종류의 괄호만 있다고 하자. 괄호의 짝이 맞는지 어떻게 알 수 있을까?
  - (): 맞음
  - {[]()}: 맞음
  - {[](}: 틀림
  - [(]})]: 틀림