자료구조 002분반 프로그래밍 과제(1)-InfixToPostfix

2010044 박진희

<구현한 기능>

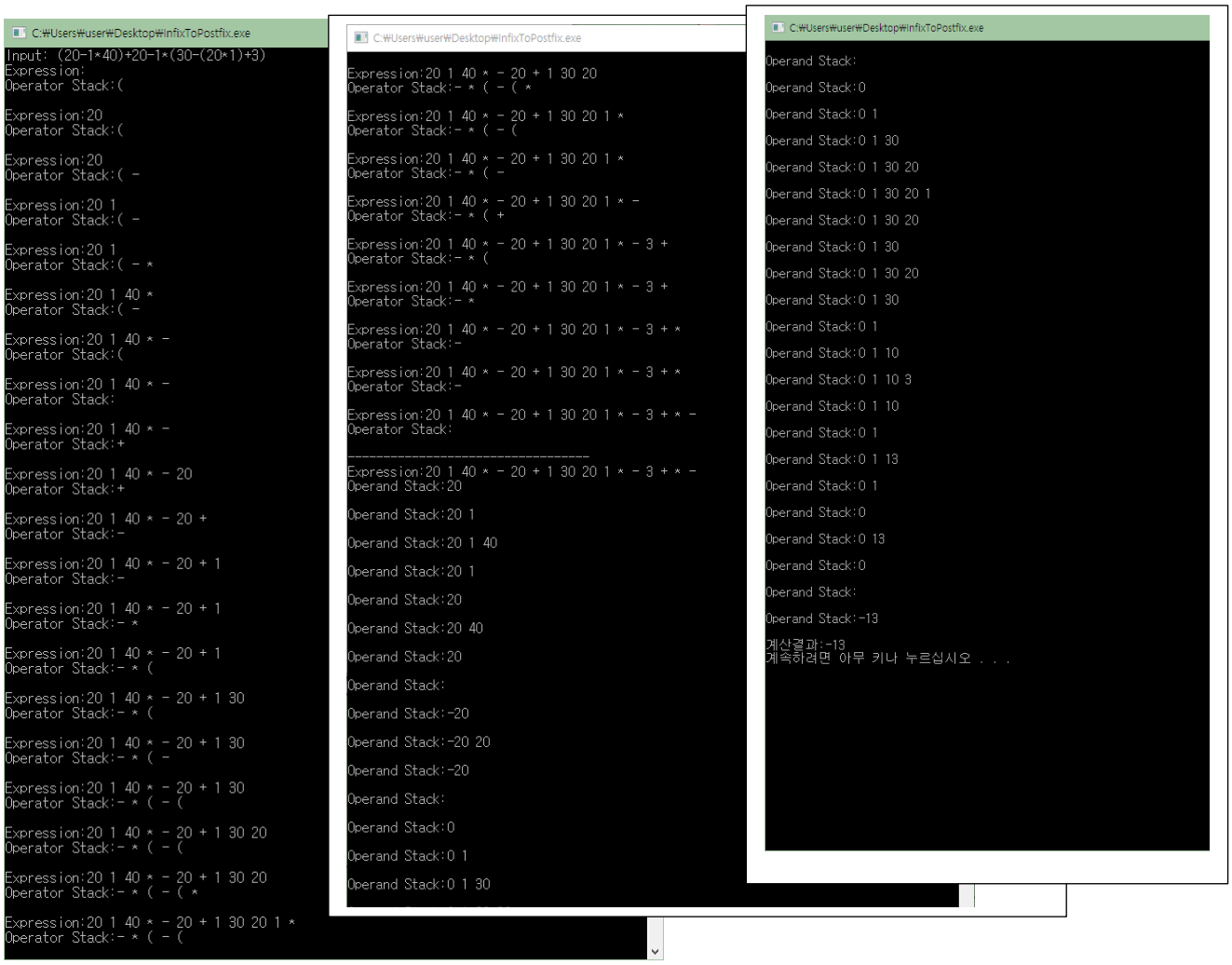
1. 다중 숫자 infix 수식을 postfix로 변환하는 기능(100%)

   조건:공백문자를 입력하지 않으나, 변환된 식에서는 피연산자간, 연산자간 공백이 있어야 함.

2. 각 변환 과정에서 스택의 상태 출력하는 기능(100%)

3. 식을 계산하여 결과값을 출력하는 기능(100%)

<실행 화면>



<전체 코드>

사용한 헤더: stdio.h, ctype.h windows.h

사용자 정의 헤더: Stack.h, Expression.h

Stack.h

```
#pragma once
#define MAX 1000
#include <stdio.h>

typedef struct stack_info {
        int top;
        char st[MAX];
}Stack;

typedef struct stack_info2 {
        int top;
        int st[MAX];
}Stack2;

void pop(Stack*);
void push(Stack*,char);
int pop2(Stack2*);
void push2(Stack2*, int);
int prior(char);
int isop(char);
int ispa(char);
void print_st(Stack*);
void print_st2(Stack2*);
```

연산자용 스택 구조체

피연산자용 스택 구조체

Stack.c

```
#include "Stack.h"
#include "Experession.h"

void pop(Stack* st )
{
    if (st->top <= -1)
    {
        printf("Empty Stack");
    }
    else
    {
        insert(' ');
        insert(st->st[st->top--]);
    }
}

void push(Stack* st,char c)
{
    if(c!='(')
        insert(' ');
    if (st->top >= MAX - 1)
        printf("Stack Full, Cannot Push");
    else
    {
        st->st[++st->top] = c;
    }
```

연산자 스택에서 top이 가리키는 토큰을 pop하여 expr에 insert하는 함수이다. 공백을 삽입하기 위해 토큰을 insert하기 전에 공백을 먼저 삽입한다.

연산자 스택에 입력된 연산자(char c)를 push한다, 피연산자간의 공백 구분을 위해 연산자가 push될때 먼저 insert(' ')을 실행한다.

```
}

int pop2(Stack2* st)
{
    if (st->top <= -1)
    {
        printf("Empty Stack");
    }
    else
    {
        return (st->st[st->top--]);
    }
}

void push2(Stack2* st, int c)
{
    if (st->top >= MAX - 1)
        printf("Stack Full, Cannot Push");
    else
    {
        st->st[++st->top] = c;
    }
}

int prior(char c)
{
    switch (c) {
    case '+':case '-': return 1;
    case '*':case '/': return 2;
    case '(':return 0;
    default: return -1;
    }
}

int isop(char c)
{
    switch (c) {
    case '+':case'-':case'/':case'*': return 1;
    default: return 0;
    }
}

int ispa(char c)
{
    switch (c) {
    case'(':return 1;
    case')':return 2;
    default:return 0;
    }
}


void print_st(Stack* st)
{
    printf("Operator Stack:");
    for (int i = 0; i <= st->top; i++)
        printf("%c ", st->st[i]);
    printf("\n\n");
}

void print_st2(Stack2* st)
```

식 결과값을 계산하기 위해 사용되는 함수이며, 피연산자를 위한 스택에 지원되는 pop 함수이다.

식 결과값을 계산하기 위해 사용되는 함수이며, 피연산자를 위한 스택에 지원되는 push 함수이다.

연산자의 우선순위를 반환하는 함수이다.

연산자라면 1을 리턴하는 함수이다. 연산자인지 확인을 위해 만든 함수이다.

괄호인지 확인을 위해 만든 함수이다. 동시에, 여는 괄호라면 push해야하고 닫는 괄호라면 pop을 실행해야 하기 때문에, 각 괄호를 구분하기 위해 값이 다르도록 설정했다.

현재 연산자 스택을 출력하는 함수이다. 각 변환과정에서의 스택을 출력하기 위해 정의된 함수이다.

현재 피연산자 스택을 출력하는 함수이다. 각 변환과정에서의 스택을 출력하기 위해 정의된 함수이다.

```
{
    printf("Operand Stack:");
        for (int i = 0; i <= st->top; i++)
            printf("%d ", st->st[i]);
    printf("\n\n");
}
```

Expression.h

```
#pragma once
#include <stdio.h>
#include <ctype.h>
#include "Stack.h"
#define MAX 1000
int p;
char expr[MAX];

int calculate(char*);
void insert(char);
void print_e(char*);
int ipow(int, int);
```

isdigit등을 현재 헤더파일을 포함하는 곳에서 사용하기 위해 ctype.h를 include 했다

변환된 식을 저장하는 char 배열 expr와 해당 배열을 관리하기 위한 index p를 선언한다.

Expression.c

```
#include "Experession.h"

void insert(char c) {
        expr[p++] = c;
}

void print_e(char* expr) {
    printf("Expression:");
    for (int i = 0; i <= p; i++) {
        printf("%c", expr[i]);
    }
    printf("\n");
}

int ipow(int a, int b)
{
    int sum = 1;
    for (int c = 0; c <b; c++)
        sum *= a;
    return sum;
}

int calculate(char* expr) {
    Stack2 oper;
    oper.top = -1;
    int temp[10] = {0,};
    int i = 0;
    int n = 0;
    int op1, op2;
```

배열 expr에 원소 c를 추가하는 함수이다.

함수를 호출하는 시점의 expr의 원소들을 모두 출력하는 함수이다.

int형 거듭제곱 계산 함수이다.

char 형태였던 피연산자를 int형으로 변환하기 위해 사용되는 함수이다.

식의 결과값을 계산한 후 반환하는 함수이다.

temp배열은 char로 자릿수의 개념없이 저장되어 있는 피연산자를 int로 변환하기 위해 일시적으로 사용하는 int형 배열이다.

i와 n은 배열의 인덱스로 사용되는 정수이며,

op1과 op2는 스택에 저장되어 pop된 피연산자들을 계산하기 위해 피연산자가 할당되는 변수이다. num은 temp배열로 자릿수 개념없이 int로 저장되어있는 피연산자가 자릿수가 반영되어 저장되

```
    int num = 0;
    char e;
    while ((e=expr[i++]) != '\0')
    {

        if (isdigit(e))
            temp[n++] = (e-'0');

        else if (isop(e))
        {
            op1 = pop2(&oper);
            op2 = pop2(&oper);

            switch (e)
            {
            case '+': {push2(&oper, op1 + op2); break; }
            case '-': {push2(&oper, op2 - op1); break; }
            case '*': {push2(&oper, op2 * op1); break; }
            case '/': {push2(&oper, op2 / op1); break; }
            default:break;
            }
        }
        else if(isspace(e)&&temp[0]!=0)
        {
            for (int k = 1; k <= n; k++)
            {
                num += temp[k-1] * ipow(10, n - k);
            }
            push2(&oper, num);
            num = 0;
            for (int t = 0; t < 10; t++)
                temp[t] = 0;
            n = 0;
        }

    }
    return pop2(&oper);

}
```

변환 및 계산방법은 추후 서술.

main.c

```
#include <stdio.h>
#include <windows.h> //실행파일의 창이 바로 종료되지 않게 하기 위해 포함함.
#include "Stack.h"
#include "Experession.h"



int main() {

    char c;
    p = 0;
    Stack op;
    op.top = -1;
```

char c는 입력받은 문자를 저장하는 변수이다.

p는 expr 배열의 인덱스이다.

연산자 스택의 구조체인 op를 선언한다.

동시에 top을 -1로 초기화하여 스택의 환경을 구성한다.

엔터키를 입력했을 때, 전체 프로세스가 실행된 후에 바로 종료되도록 개행문자를 받게되면 while문을 빠져나오도록 한다.

```
    printf("Input: ");
    while ((c = getchar()) != '\n')
    {
        if (isdigit(c))
        {
            insert(c);
        }

        else if (isop(c))
        {
            if (op.top==-1)
            {
                print_e(expr);
                print_st(&op);
                push(&op, c);
                print_e(expr);
                print_st(&op);
                continue;
             }
            else if (prior(c) > prior(op.st[op.top]))
            {
                print_e(expr);
                print_st(&op);
                push(&op, c);
                print_e(expr);
                print_st(&op);
            }
            else if (prior(c) <= prior(op.st[op.top]))
            {
                while (prior(c) <= prior(op.st[op.top]))
                {
                    print_e(expr);
                    print_st(&op);
                    pop(&op);
                }
                push(&op, c);
                print_e(expr);
                print_st(&op);
            }

        }

        else if (ispa(c))
        {
            switch (ispa(c))
            {
            case 1: {push(&op, c); print_e(expr);
                print_st(&op); break; }
            case 2:
            {
                while (op.top > -1)
                {
                    if (op.st[op.top] == '(')
                    {
                        op.top--;
                        break;
                    }
                        pop(&op);
                        print_e(expr);
                        print_st(&op);
                }
```

변환방법 추후 서술

```
                break;
            }
            }
        }
        else
            continue;
    }

    while (op.top > -1)
    {
        print_e(expr);
        print_st(&op);
        pop(&op);
        print_e(expr);
        print_st(&op);
    }
    printf("-----------------------------------\n");
    print_e(expr);
    int answer = 0;
    answer = calculate(expr);
    printf("계산결과:%d\n",answer);



    system("pause");}
```

1. infix 수식을 postfix로 변환하는 기능

다중 숫자 infix 수식을 postfix로 변환하기 위해서, 연산자를 저장하는 자료구조는 스택을 이용하였으며, 바로 식을 화면에 출력하는 식이 아닌, char 배열인 expr에 저장한 뒤에 출력하는 방식으로 설계하였다.

연산자를 저장하기 위한 스택 구현을 위한 함수, 구조체 선언은 Stack.h 헤더파일에 해두었다.

실제 코드는 Stack.c에 구현되어있는데, 변환된 식을 저장하기 위한 기능이 구현되어있는 Expression.h와 Expression.c에 접근하기 위해 해당 헤더파일을 include 하였다.

기본적인 infix to postfix 기능을 구현한 방법이다.

1. 피연산자는 바로 expr에 insert된다. 피연산자를 구분하기 위해 ctype헤더에 포함되어있는 함수인 isdigit을 사용한다.
2. 연산자라면 해당 규칙을 따른다.

    -연산자 스택이 비어있다면 바로 push한다.

    -직전에 push된 연산자의 우선순위보다 현재 입력된 연산자의 우선순위가 더 높다면, push한다.

-직전에 push된 연산자의 우선순위보다 작거나 같다면, 현재 연산자의 우선가 더 높을때 까지 스택의 연산자를 pop한 뒤에 push 한다.

3. 괄호라면, ispa함수를 통해 '('라면 1, ')'라면 2를 받아 1이라면(여는 괄호라면) push하고, 2라면(닫는 괄호라면) '('가 top이 가리키는 토큰일때까지 pop하고, top을 1만큼 감소시켜 (을 expr에 추가하지 않도록 한다.

4. getchar은 화살표키의 입력도 처리되기 때문에, else를 통해 입력이 무산되도록 continue하여 다음 피연산자나 연산자를 읽어들이도록 한다.

5. 마지막에는 모든 연산자를 pop하도록 한다.

공백문자를 입력하지 않고, 변환된 식에서 공백문자로 구별하는 기능을 구현하기 위해서

해당 규칙을 설정하였다. 여기서 insert()함수는 char형 배열인 expr에 원소를 추가하는 함수이다.

1. 피연산자 하나가 끝나면, 반드시 뒤에 연산자가 온다. 따라서 연산자 앞까지만 하나의 피연산자로 구분하도록, 연산자가 입력되어 push되면 insert(' ')한다. 다만, 조건없이 무조건 push될때 insert(' ')를 하게 된다면, 괄호가 push될 때 공백이 중복되게 된다. 따라서, '('을 push하지 않는 경우에만 insert(' ')가 실행되도록 한다.
2. 연산자간 공백은, 연산자가 pop되어 expr에 insert될때마다 insert(' ')가 먼저 실행되도록 하여 처리한다.

2. 각 변환 과정에서 스택의 상태를 출력하는 기능

연산자 스택의 상태를 출력하기 위해 print_st 함수를 사용하였으며, 변환된 식을 저장한 expr 을 출력하기 위해 print_e 함수를 사용하였다.

스택의 상태와 변환된 식을 출력하는 기준, 조건은 다음과 같다.

1. 피연산자간의 구분은 getchar로 인하여, 한 자리 숫자씩만 입력되어 어느 시점에서 완전한 피연산자가 expr에 insert되었는지는 연산자가 push된 시점에서만 알 수 있다. 따라서, 연산자가 push되기 전에 print_e와 print_st를 하여 피연산자가 입력된 상태임을 출력한다. 괄호의 경우에는 push되더라도 괄호 이전에는 무조건 연산자이기때문에, 해당 피연산자 구분을 위한 출력을 실행하지 않는다.

2. 현재 입력된 연산자의 우선순위가 스택 상 최상단의 연산자보다 우선순위가 높아 push 되었을 때 push 후에 print_e와 print_st를 실행하여 연산자가 스택에 push됨을 출력한다.

3. '(' 괄호가 push된 경우라면 push 후에 print_e와 print_st를 실행한다.

4. ')'괄호로 인해 '('괄호까지 pop하는 경우에는, 피연산자는 관계 없으므로 pop한 이후에만 print_e와 print_st를 실행한다.

5. 현재 입력된 연산자의 우선순위가 스택 상 최상단의 연산자보다 우선순위보다 갖거나 낮아서 스택에서 연산자가 pop될때, pop하기 전에 print_e와 print_st를 실행해 expr에 추가됨을 출력한다. 이후 현재 입력된 연산자를 push한 후에 print_e와 print_st를 실행하여 push됨을 출력한다.

6. 남은 연산자를 pop할 때, pop되는 과정을 출력하기 위해 pop 실행 전 후에 모두 print_e와 print_st를 실행한다.

3. 식을 계산하여 결과값을 출력하는 기능

공백을 기준으로 피연산자와 연산자가 구분되어 char 배열 expr에 식이 저장되어있다.

따라서, 숫자 138을 입력했더라도 "1","3","8"," ",... 식으로 저장되어 있다.

계산을 위해서는 char형태로 분리되어있는 피연산자를 int로 변환해주어야한다.

여기서 해결해야할 문제는 다음과 같다.

1) char형태의 수를 int로 변환해야한다.
2) 피연산자의 자리수가 반영되어있지 않다.

1)을 해결하기 위해서, isdigit을 통해 expr의 현재 인덱스의 원소가 숫자일 경우, '0'을 빼주어 한 자리수의 숫자인 int로 변환한다.

2)를 해결하기 위해서는 다음과 같은 방식을 따른다.

1. 1)에서 변환한 수를 int형 배열인 temp에 저장한다.
2. expr에서 공백을 만날 때 까지, temp에 변환한 수를 저장한다.
3. 공백을 만난다면, temp의 인덱스로 사용하였던 n은 곧 피연산자의 자리수가 된다.
4. 각 앞의 인덱스의 수일수록 높은 차수이므로, for문을 이용하여 반복 횟수를 제어하기위한 변수int k(초기화는 1)를 통해 `num += temp[k-1] * ipow(10, n - k);` 을 실행한다.
5. num은 자리수가 반영된 피연산자가 된다.

이후, 계산을 위해 num은 피연산자 스택에 push하고(push2함수) temp와 n을 초기값으로 초기화해준 후에 반복한다.

계산하는 방식은 다음과 같다.

1. expr에서 읽은 차례의 원소가 연산자라면, 피연산자 스택에서 pop(pop2함수)을 두 번 하여, 각 피연산자를 op1, op2에 저장한다.
2. 연산자의 종류에 따라 switch문으로 분기하여, 연산자 종류에 따라 op1와 op2를 계산한 후 다시 피연산자 스택에 push한다.(op1이 먼저 pop된 숫자임에 유의한다)

3. expr에 더 이상 읽어들일 원소가 없다면, 계산결과값은 피연산자 스택에 마지막으로 남아있는 한 개의 피연산자가 되므로, pop하여 이 값을 반환한다.

피연산자 스택 또한 계산 과정에서의 pop 및 push 되는 상황을 출력하기 위해, print_st2 함수가 정의되어 사용된다. call by reference로 받아 (사실 call by value로 받아도 상관은 없으나, 공간복잡도를 줄이기 위해 call by reference를 채택하였다) 피연산자 스택에서는 infix to postfix 변환 과정에서의 출력과는 다르게 각 피연산자가 이미 int형 변수로 구분되어있는걸 push 및 pop 하는 것이므로, pop 및 push 될때 실행되면 피연산자 스택의 변환 과정을 충분히 출력할 수 있다.