자료구조
실습 레포트

제출일	21.05.16	**전 공**	컴퓨터소프트웨어공학과
과 목	자료구조 1	**학 번**	20184612
담당교수	홍 민 교수님	**이 름**	김동민

| 목 차 |

1. 후위표기변환과 계산 프로그램

후위표기변환과 계산 프로그램

- **중위 표기식으로 저장된 데이터를 data.txt파일에서 입력받아, 후위 표기식으로 변환하여 출력하고, 다시 후위 표기식으로 표현된 식을 계산하여 출력하는 프로그램을 작성한다. 단, 숫자는 한자리가 아니라 여러 자리가 입력되어서 계산되어야 한다.**

1.1 문제 분석

조건 1 중위 표기식으로 저장된 데이터를 data파일에서 입력받는다.
조건 2 입력받은 중위 표기식을 후위 표기식으로 변환한다.
조건 3 후위 표기식을 이용하여 계산하고 결과를 출력한다.
조건 4 숫자는 한자리가 아니라 여러 자리가 입력되어야 한다.

- 이 문제는 data파일에 중위 표기식이 저장되어있다, 문자열 변수 2개를 만들고 하나는 중위 표기식, 다른 하나는 후위 표기식을 저장한다. 먼저 data로부터 식을 입력받은 후 소괄호의 개수가 맞는지 확인한다. 만약 괄호의 개수가 맞지 않다면 바로 프로그램을 종료한다.

후위 표기식으로 변경하기 위해 피연산자라면 바로 후위 문자열에 담고, 연산자 또는 (일 땐 스택에 push, 연산자의 우선순위가 더 크거나 같고) 문자 또는 중위 문자열의 끝에 도달한다면 스택 pop을 하여 후위 문자열에 담는다. 한자리의 숫자가 아니기 때문에 연산자와 피연산자 사이에는 공백문자를 넣어 구분한다.

후위 표기식으로 변경이 완료되었다면 피연산자일 땐 문자열을 숫자로 변환하여 스택에 push하고 연산자라면 피연산자를 pop하여 계산을 실시한다. 계산을 완료했다면 다시 스택에 push한다.

후위 표기 수식 계산 알고리즘

후위 표기 수식은 식 자체에 우선순위가 표현되어 있기 때문에, 연산자의 우선순위를 고려할 필요가 없다. 후위 표기 수식을 계산하려면 수식을 스캔하여 피연산자이면 스택에 저장하고 연산자이면 필요한 수만큼의 피연산자를 꺼내 연산을 실행하고 연산의 결과를 다시 스택에 저장한다.

```
calc_posfix:
        스택 s를 생성하고 초기화한다.
        for item in 후위 표기식 do
                if(item이 피연산자이면)
                        push(s, item)
                else if(item이 연산자op이면)
                        second←pop(s)
                        first←pop(s)
                        result←first op second
                        push(s, result)
        final_result←pop(s);
```

1.2 소스 코드

```c
/*
    학번 : [illegible]
    학과 : 컴퓨터소프트웨어공학과
    이름 : [illegible]
    프로그램명 : 여러자리 수로 이루어진 중위표기식을
                후위표기로 변경하고 계산하는 프로그램
*/
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#pragma warning(disable : 4996)

#define TRUE 1
#define FALSE 0
#define MAX_STACK_SIZE 100

typedef int element;

typedef struct{
    element stack[MAX_STACK_SIZE];
    double dstack[MAX_STACK_SIZE];
    int top;
}StackType;

void init(StackType *s);                                    //스택초기화
int is_empty(StackType *s);                                 //스택empty 검사
int is_full(StackType *s);                                  //스택 full 검사
void push(StackType *s, element item);                      //스택push
element pop(StackType *s);                                  //스택pop
element peek(StackType *s);                                 //스택peek
int prec(char op);                                          //연산자 우선순위 계산
int check_matching(char *in);                               //괄호 검사 함수
void infix_to_postfix(char exp[], char *inpost);            //중위표기식을 후위표기식으로 변환
double eval(char exp[]);                                        //후위표기식을 계산하는 함수
double getValue(int *i, char exp[]);                            //문자열을 숫자로 변환하는 함수
double dpop(StackType *s);
void dpush(StackType *s, double item);

int main(){
    char *buf, temp;
    char *buf2;
    int i=0;
    FILE *fp = fopen("data.txt", "r");                      //읽기 전용 파일 오픈
    if(fp==NULL){
        printf("파일이 열리지 않았습니다.\n");
        return 0;
    }
    while(!feof(fp)){
        fscanf(fp, "%c", &temp);                            //%s로 입력받는다면 공백문자를 인식하지 못하기 때문에 문자 1개씩 입력받는다.
        i++;
    }
    buf = (char*) calloc(sizeof(char),i);
    buf2 = (char*) calloc(sizeof(char),i);
    rewind(fp);
    i=0;
    while(!feof(fp)){
        fscanf(fp, "%c", &buf[i]);                          //%s로 입력받는다면 공백문자를 인식하지 못하기 때문에 문자 1개씩 입력받는다.
        i++;
    }
    printf("중위표기식 %s\n", buf);                         //정상적으로 입력 받았는지 확인
    if(!check_matching(buf)){                               //괄호의 개수가 맞는지 확인
        printf("괄호가 올바르지 않습니다.\n");               //만약 괄호의 개수가 맞지 않는다면 return
        return 0;
    }
    infix_to_postfix(buf, buf2);                            //중위표기식 buf를 후위표기식 buf2에 저장
    printf("후위표기식 %s\n", buf2);                        // 후위 표기식 확인 출력
    printf("결과값: %lf\n", eval(buf2));                        // 후위표기식으로 계산된 결과값 출력
    fclose(fp);
    free(buf);
    free(buf2);
    return 0;
}


void init(StackType *s){
    s->top = -1;                                            //스택의 top 초기값을 -1로 설정
}
int is_empty(StackType *s){
    return (s->top == -1);                                  //만약 스택의 top이 -1이라면 스택이 비어있음
}
int is_full(StackType *s){
    return (s->top == (MAX_STACK_SIZE-1));                  //만약 스택의 top이 최대 스택값 -1 이라면 스택이 꽉 찼음을 나타냄
}
void push(StackType *s, element item){                      //매개변수로 스택과 스택에 push할 값
    if(is_full(s)){                                         //스택이 다 찼는지 확인
        printf("stack overflow\n");
        return;
    }
    s->stack[++(s->top)] = item;                            //스택에 자리가 있다면 top값을 ++하고 값을 넣음
}
element pop(StackType *s){
    if(is_empty(s)){                                        //스택이 비어있다면 더이상 pop을 할 수 없으므로 강제 종료
        printf("stack underflow\n");
        exit(1);
    }
    return s->stack[(s->top)--];                            //스택의 top값을 pop하고
}

element peek(StackType *s){
    if(is_empty(s)){
        printf("stack underflow\n");
        exit(1);
    }
    return s->stack[s->top];
}
double dpop(StackType *s){                                  //계산할수 꺼낼시 소수 팝
    if(is_empty(s)){
        printf("stack underflow\n");
        exit(1);
    }
    return s->dstack[(s->top)--];
}
void dpush(StackType *s, double item){                      //계산할수 꺼낼시 소수 push
    if(is_full(s)){
        printf("stack overflow\n");
        return;
    }
    s->dstack[++(s->top)] = item;
}
int prec(char op){                                          //연산자의 우선순위 결정 *,/가 가장 우선순위가 높고 +-가 그다음, ()가 가장 낮다
    switch(op){
    case '(': case ')':                                     //()의 경우는 우선순위가 가장 낮으므로 0리턴
        return 0;
    case '+': case '-':                                     //+-의 경우는 우선순위가 2번째이므로 1리턴
        return 1;
    case '*': case '/':                                     //*/의 경우는 우선순위가 1번째이므로 2리턴
        return 2;
    }
    return -1;                                              //연산이 불가하다면 -1리턴
}
```

```c
int check_matching(char *in){                        //괄호 검사 함수
    StackType s;                                     //괄호를 넣을 스택변수
    char ch, open_ch;
    int i, n = strlen(in);
    init(&s);                                        //스택 초기화
    for(i=0; i<n; i++){
        ch = in[i];
        switch(ch){
        case '(':                                    //'('문자라면 push
            push(&s, ch);
            break;
        case ')':                                    //')'문자를 만나면
            if(is_empty(&s))                         //스택이 비어있다면 '(', ')'의 개수가 달라 오류므로 오류
                return FALSE;
            else{
                open_ch = pop(&s);                   //스택에서 pop
                if(open_ch=='(' && ch != ')')        //ch와 open_ch가 같은 짝이 아니라면 오류
                    return FALSE;
                break;
            }
        }
    }
    if(!is_empty(&s))                                //괄호의 개수가 같다면 스택이 비어있어야 함, 비어있지 않다면 오류
        return FALSE;
    return TRUE;
}
void infix_to_postfix(char exp[], char *inpost){                  //중위표기식을 후위표기식으로 변환, 매개변수 1:중위, 2:후위를 담을 문자
    int i=0;
    char ch, top_op;
    int len = strlen(exp);
    int cnt =0;
    StackType s;
    init(&s);

    for(i=0; i<len; i++){
        ch = exp[i];
        if(ch!=' '){                                               //ch가 공백문자가 아닐때만 후위에 출력
            switch(ch){                                            //만약에 연산자라면 + - * /
            case '-':
                if(!isdigit(exp[i+1])){                            //만약 -다음문자가 공백이 아니고 숫자라면 음수임을 나타내므로
                    inpost[cnt++] = ch;
                    break;                                         //연산자가 아니므로 후위표기식에 출력하고 switch를 break
                }
            case '+':
            case '*':
            case '/':
                while(!is_empty(&s) && (prec(ch) <= prec(peek(&s)))){   //연산자는 기본적으로 스택에 push하지만
                    inpost[cnt++] = pop(&s);                            //스택에 있는 연산자의 우선순위가 더 크거나 같으면 pop해서 출
                    inpost[cnt++] = ' ';
                }
                push(&s, ch);                                      //연산자 push
                break;
            case '(':                                              //'('문자라면 스택에 push
                push(&s, ch);
                break;
            case ')':
                top_op = pop(&s);
                while(top_op!='('){                    //왼쪽 괄호를 만날때까지 출력
                    inpost[cnt++] = top_op;            //왼쪽괄호가 아니므로 후위표기식에 출력
                    inpost[cnt++] = ' ';               //공백문자로 구분
                    top_op = pop(&s);                  //연산자 pop
                }
                break;
            default:                                   //피연산자(숫자)일 때
                while(1){
                    ch = exp[i];
                    if(ch == ' '|| i == len)break;              //공백문자를 만났다면 피연산자 출력 종료
                    inpost[cnt++] = ch;                //공백문자가 아니라면 피연산자이므로 후위표기식에 출력
                    i++;
                }
                inpost[cnt++] = ' ';                       //공백문자로 구분
                break;
            }
        }
    }
    while(!is_empty(&s)){                              //문자열의 끝에 도달하였다면 스택에 남은 연산을 모두 pop
        inpost[cnt++] = pop(&s);
        inpost[cnt++] = ' ';
    }
}
```

```c
double eval(char exp[]){                                          //후위표기식을 이용하여 계산하는 함수
    double op1, op2, value;
    int i = 0;
    int len = strlen(exp);
    char ch;
    StackType s;
    init(&s);

    for (i = 0; i < len; i++) {
        ch = exp[i];
        if(ch!=' '){
            if (ch != '+' && ch != '-' && ch != '*' && ch != '/'){   // 입력이 피연산자이면
                value = getValue(&i, exp);                         //피연산자를 문자열에 담아 숫자로 리턴하는 함수 [illegible]
                dpush(&s, value);                                  //값을 스택에 push
            }
            else if( ch == '-'){                                   //만약 문자가 -인 경우는 음수일때 또는 연산자일때이다
                if(isdigit(exp[i+1])){                             //-문자 다음문자가 공백이 아니라 숫자라면 음수임을 나타내므로 숫자로 바꾸는 함수를 호출한다.
                    value = getValue(&i, exp);
                    dpush(&s, value);                              //값을 리턴받으면 push
                }
                else{                                              //-문자 다음 공백이라면 연산자 -이므로 스택을 pop하여 -연산한다.
                    op2 = dpop(&s);
                    op1 = dpop(&s);
                    dpush(&s, op1 - op2);
                }
            }
            else{                                                  //연산자이면 피연산자를 스택에서 제거
                op2 = dpop(&s);
                op1 = dpop(&s);
                switch (ch) {                                      //연산을 수행하고 스택에 저장
                case '+': dpush(&s, op1 + op2); break;
            //  case '-': push(&s, op1 - op2); break;              //-앞에 위에서 처리
                case '*': dpush(&s, op1 * op2); break;
                case '/': dpush(&s, op1 / op2); break;
                }
            }
        }
    }
    return dpop(&s);
}
double getValue(int *i, char exp[]){                          //문자열을 숫자로 변환해주는 함수 매개변수 1:후위문자열배열의 index를 나타내는 i 포인터, 2:후위 문자열
    int j=0;                                                  //1은 문자열의 index
    char ch;
    char str[50]={0};
    while(1){
        ch = exp[*i];                                         //포인터 i의 값을 배열의 index로 하여 문자 ch에 담는다
        if(ch==' ')break;                                     //만약 ch가 공백문자라면 break
        str[j++]=ch;                                          //ch를 str문자열에 담고 j를 ++한다.
        (*i)++;                                               //포인터의 값을 ++
    }
    return atof(str);                                         //만약 ch가 공백문자를 만나 break되었다면 atof함수를 이용해 str문자열을 숫자로 변환하여 이를 리턴한다.
}
```

1.3 소스 코드 분석

```
/*
    학번 : 20184612
    학과 : 컴퓨터소프트웨어공학과
    이름 : 김동민
    프로그램명 : 여러자리 수로 이루어진 중위표기식을
                 후위표기로 변경하고 계산하는 프로그램
*/
#include<stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>
#pragma warning(disable : 4996)

#define TRUE 1
#define FALSE 0
#define MAX_STACK_SIZE 100

typedef int element;

typedef struct{
    element stack[MAX_STACK_SIZE];
    double dstack[MAX_STACK_SIZE];
    int top;
}StackType;
```

1. 소스코드를 작성한 날짜, 이름, 프로그램명을 작성한다.

2. 필요한 헤더를 포함한다.

3. c 코드에서는 true, false 값이 없기 때문에 define을 해주고 스택의 최댓값을 100으로 설정한다.

4. 스택 구조체를 선언한다. 최대 100개의 스택을 저장할 수 있는 stack 배열과 현재 스택의 위치를 알려주는 top을 선언한다.

5. double dstack은 계산함수에서 소수가 나올 수도 있기 때문에 double 형으로 스택을 하나 더 만들었다. 이 스택은 계산함수를 수행할 때에만 사용한다.
typedef를 이용하여 스택을 만들 때 선언문을 단순하게 사용할 수 있다.

```
void init(StackType *s);
int is_empty(StackType *s);
int is_full(StackType *s);
void push(StackType *s, element item);
element pop(StackType *s);
element peek(StackType *s);
int prec(char op);
int check_matching(char *in);
void infix_to_postfix(char exp[], char *inpost);
double eval(char exp[]);
double getValue(int *i, char exp[]);
double dpop(StackType *s);
void dpush(StackType *s, double item);
```

6. 필요한 함수들을 선언한다.

- init 함수는 스택을 초기화 할 때 사용한다. top 값을 -1로 설정한다.
- is_empty함수는 스택이 비어있는지 확인하는 함수이다.
- is_full함수는 스택이 차있는지 확인하는 함수이다.
- push함수는 스택에 값을 추가하는 함수이다.
- pop함수는 스택의 top값에 있는 함수를 반환하는 함수이다.
- peek함수는 top값에 있는 값을 확인만 하는 함수이다 top값은 --하지 않는다.
- prec함수는 연산자의 우선순위를 판별하여 반환하는 함수이다.
- check_matching함수는 괄호의 개수가 맞는지 검사하는 함수이다.
- infix_to_postfix함수는 중위표기식을 후위표기식으로 변경하는 함수이다.
- eval함수는 후위표기식으로 변경된 식을 계산하여 반환하는 함수이다.
- getValue함수는 문자열로 입력된 숫자를 숫자로 변환하여 반환하는 함수이다.
- dpop함수는 계산함수 진행 시 dstack스택에 소수가 저장되어있을 때 소수를 반환할 수 있는 함수이다.
- dpush함수는 계산함수에서 dstack스택에 소수를 push할 수 있는 함수이다

```c
int main(){
    char *buf, temp;
    char *buf2;
    int i=0;
    FILE *fp = fopen("data.txt", "r");
    if(fp==NULL){
        printf("파일이 열리지 않았습니다.₩n");
        return 0;
    }
    while(!feof(fp)){
        fscanf(fp, "%c", &temp);
        i++;
    }
    buf = (char*) calloc(sizeof(char),i);
    buf2 = (char*) calloc(sizeof(char),i);
    rewind(fp);
    i=0;
    while(!feof(fp)){
        fscanf(fp, "%c", &buf[i]);
        i++;
    }
    printf("중위표기식 %s₩n", buf);
    if(!check_matching(buf)){
        printf("괄호가 올바르지 않습니다.₩n");
        return 0;
    }
    infix_to_postfix(buf, buf2);
    printf("후위표기식 %s₩n", buf2);
    printf("결과값: %lf₩n", eval(buf2));
    fclose(fp);
    free(buf);
    free(buf2);
    return 0;
}
```

7. 필요한 변수들을 선언한다.

- temp는 파일로부터 입력을 받기위한 임시변수이고 buf는 중위표기식, buf2는 후위표기식을 저장하는 포인터이다.
- i는 파일로부터 문자를 입력받을 때 사용하고 파일포인터 fp는 읽기 전용으로 오픈한다.

8. 먼저 파일의 끝까지 파일을 입력받으면서 임시변수 temp에 담는다. 이와 함께 몇 개의 문자열로 동적할당할지 개수를 같이 샌다.

9. 배열로 문자열을 입력받는 것보다 동적할당이 메모리를 적게 차지하므로 파일을 전부 입력받았다면 문자열의 개수인 i만큼 calloc함수를 이용하여 buf를 동적할당한다. calloc함수는 할당받은 기억공간을 모두 0으로 초기화 시켜서 코드의 길이를 줄이고 쓰레기 값이 들어가는 것을 방지할 수 있다. buf2또한 최대로 i만큼의 문자열을 가질 수 있기 때문에 i로 동적할당했다.

10. rewind를 이용하여 다시 파일의 처음으로 가서 다시 입력받는다. %s를 이용하여 문자열을 입력받는다면 공백문자를 인식하지 못하기 때문에 문자 1개씩 파일 끝까지 입력하여 buf에 저장한다.

11. 중위표기식이 잘 입력되었는지 확인해보고, 괄호의 개수가 맞는지 check_matching함수를 호출하여 검사한다. 만약 괄호의 개수가 맞지 않다면 바로 프로그램을 종료한다.

12. infix_to_postfix함수를 호출하여 중위표기식을 후위표기식으로 변환한 후 후위표기식이 잘 입력되었는지 출력하여 확인한다.

13. 마지막으로 eval함수를 호출하여 후위표기식을 이용하여 계산한 결과를 출력한다. 계산 결과를 호출하는 eval함수는 double형을 반환하기 때문에 %lf를 이용하여 출력한다.

14. 파일사용이 모두 끝났으므로 fclose를 한다.
동적할당한 buf와 buf2또한 메모리를 해제한다.

```
void init(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)){
        printf("stack overflow₩n");
        return;
    }
    s->stack[++(s->top)] = item;
}
element pop(StackType *s){
    if(is_empty(s)){
        printf("stack underflow₩n");
        exit(1);
    }
    return s->stack[(s->top)--];
}
element peek(StackType *s){
    if(is_empty(s)){
        printf("stack underflow₩n");
        exit(1);
    }
    return s->stack[s->top];
}
```

15. init함수는 스택의 top값을 -1로 초기화 하는 함수이다.
스택을 선언하면 init함수를 실행시켜서 반드시 top의 초기 값을 -1로 설정해야한다.

16. is_empty함수는 스택이 비었는지 확인하는 함수이다.
만약 스택이 비어있다면 1을 반환하고 스택에 값이 있다면 0을 반환한다.

17. is_full함수는 스택이 다 찼는지 확인하는 함수이다.
배열의 끝 값은 전체 값(MAX_STACK_SIZE) -1 이므로 top값이 이와 같다면 스택이 full이므로 1을 반환하고 아니라면 0을 반환한다.

18. push함수는 입력받은 item값을 스택에 push하는 함수이다.
먼저 스택이 차있는지 확인하는 is_full함수로 검사하고 만약 스택에 자리가 있다면 top값을 ++한 후에 값을 대입한다.

19. pop함수는 스택의 top에 있는 값을 반환하는 함수이다.
스택이 비어있는지 확인하는 is_empty함수를 호출한 후 만약 스택이 비어있다면 exit함수로 강제종료하고, 스택에 남아있는 값이 있다면 값을 리턴하고 top값을 --한다.

20. peek함수는 스택의 top에 어떤 값이 있는지 확인 하는 함수이다.
먼저 스택이 비어있는지 확인하는 is_empty함수를 호출한 후 스택에 값이 남아있다면 스택의 top에 있는 값을 반환한다.
peek함수는 스택의 top에 어떤 값이 있는지 확인만 하는 함수이기 때문에 --를 하지 않고 반환만 한 후 종료한다.

```
double dpop(StackType *s){
    if(is_empty(s)){
        printf("stack underflow\n");
        exit(1);
    }
    return s->dstack[(s->top)--];
}
void dpush(StackType *s, double item){
    if(is_full(s)){
        printf("stack overflow\n");
        return;
    }
    s->dstack[++(s->top)] = item;
}
```

21. dpop함수는 계산함수 수행에만 사용되는 함수이다. 계산 시 소수가 나올 수도 있기 때문에 double형 스택인 dstack에서 소수를 반환할 수 있도록 반환형이 double인 함수이다. pop 후엔 top값을 --한다.

22. dpush함수는 매개변수로는 스택과 double형 item을 전달받는다. 계산함수를 수행할 때 소수가 나올 시 s의 dstack스택에 double형 item값을 push하는 역할을 한다. top값을 ++한 후 item을 스택에 넣는다.

```
int prec(char op){
    switch(op){
    case '(': case ')':
        return 0;
    case '+': case '-':
        return 1;
    case '*': case '/':
        return 2;
    }
    return -1;
}
```

23. prec함수는 연산자의 우선순위를 비교하여 반환하는 함수이다
매개변수 char값으로는 연산자 또는 '(', ')'값이 전달된다.
* 이나 / 연산자의 경우 우선순위가 가장 높기 때문에 2를 반환하고
+ 이나 - 연산자의 경우는 그 다음이므로 1을 반환하고
(,)의 경우는 우선순위가 가장 낮은 연산자로 취급하므로 0을 반환한다.
만약 모든 조건을 만족하지 않는다면 -1을 반환한다.

```
int check_matching(char *in){
    StackType s;
    char ch, open_ch;
    int i, n = strlen(in);
    init(&s);
    for(i=0; i<n; i++){
        ch = in[i];
        switch(ch){
        case '(':
            push(&s, ch);
            break;
        case ')':
            if(is_empty(&s))
                return FALSE;
            else{
                open_ch = pop(&s);
                if(open_ch=='(' && ch != ')')
                    return FALSE;
                break;
            }
        }
    }
    if(!is_empty(&s))
        return FALSE;
    return TRUE;
}
```

24. check_matching 함수는 괄호의 개수가 맞는지 검사하는 함수이다.
매개변수로는 중위표기식 문자열을 전달받는다.
조건 1. 왼쪽 괄호의 개수와 오른쪽 괄호의 개수가 같아야 함
조건 2. 같은 종류의 괄호에서 왼쪽 괄호는 오른쪽 괄호보다 먼저 나와야함
조건 3. 서로 다른 종류의 왼쪽 괄호와 오른쪽 괄호 쌍은 서로를 교차하면 안 됨

25. 스택 s를 선언 후 초기화를 한 후 strlen함수로 문자열의 길이를 구하고 for문을 반복한다.
만약 왼쪽 괄호가 나온다면 스택에 push하고 오른쪽 괄호가 나온다면 pop을 한다.

26. 오른쪽 괄호일 때 스택에 값이 없다면 왼쪽괄호가 존재하지 않는다는 것이므로 FALSE를 반환한다. (조건 2)
스택에 값이 있다면 open_ch변수에 스택을 pop하고 open_ch에 담는다.
ch의 값이 '('인데 open_ch의 값이 ')'가 아니라면 괄호의 짝이 맞지 않기 때문에 FALSE를 반환한다. (조건 3)
마지막 괄호까지 모두 조사했는데도 스택에 괄호가 남아있다면 괄호의 개수가 맞지 않으므로 FALSE를 반환한다. (조건 1)
남아있는 괄호가 없다면 개수가 맞으므로 TRUE를 반환한다.

```
void infix_to_postfix(char exp[], char *inpost){
    int i=0;
    char ch, top_op;
    int len = strlen(exp);
    int cnt =0;
    StackType s;
    init(&s);

    for(i=0; i<len; i++){
        ch = exp[i];
        if(ch!=' '){
            switch(ch){
            case '-':
                if(isdigit(exp[i+1])){
                    inpost[cnt++] = ch;
                    break;
                }
            case '+':
            case '*':
            case '/':
                while(!is_empty(&s) && (prec(ch) <= prec(peek(&s)))){
                    inpost[cnt++] = pop(&s);
                    inpost[cnt++] = ' ';
                }
                push(&s, ch);
                break;
```

27. infix_to_postfix함수는 중위표기식을 후위표기식으로 변경하는 함수이다. 매개변수 exp는 중위표기식, inpost는 후위표기식을 담을 변수이다.

28. i를 exp문자열의 길이만큼 for문으로 반복한다.
문자 ch가 연산자 -,+,*,/일 때는 스택에 push를 한다. 만약 ch가 -인 경우는 숫자가 음수일 때 또는 연산자 -일 때이다.
ch가 -이고 다음에 있는 문자가 숫자이면 음수라는 것을 알 수 있고, 공백문자라면 연산자라는 것을 알 수 있다.
ctype헤더파일에 있는 isdigit함수를 이용하여 -다음에 있는 문자가 숫자라면 연산자에 push하지 않고 후위표기식 문자열에 넣은 후 break한다.

29. ch가 연산자라면 스택에 push를 해야 하는데, 연산자의 우선순위를 고려하여 출력해야한다. 스택에 존재하는 연산자가 현재 처리중인 연산자보다 우선순위가 높거나 같다면 일단 스택에 있는 연산자들 중에서 우선순위가 높거나 같은 연산자들을 먼저 후위문자열에 출력한다. 출력한 뒤엔 공백문자를 추가하여 구별할 수 있도록 한다.

30. 우선순위가 높거나 같은 연산자들을 후위문자열에 출력한 후에는 현재 연산자를 스택에 push한다.

```
            case '(':
                push(&s, ch);
                break;
            case ')':
                top_op = pop(&s);
                while(top_op!='('){
                    inpost[cnt++] = top_op;
                    inpost[cnt++] = ' ';
                    top_op = pop(&s);
                }
                break;
            default:
                while(1){
                    ch = exp[i];
                    if(ch == ' '|| i == len)break;
                    inpost[cnt++] = ch;
                    i++;
                }
                inpost[cnt++] = ' ';
                break;
            }
        }
    }
    while(!is_empty(&s)){
        inpost[cnt++] = pop(&s);
        inpost[cnt++] = ' ';
    }
}
```

31. 두 번째로는 괄호를 처리해야한다. 괄호의 개수는 이미 검사가 완료되었기 때문에 신경 쓰지 않아도 된다.
왼쪽괄호는 무조건 스택에 삽입한다. 왼쪽괄호는 우선순위가 가장 낮은 연산자로 취급하기 때문에 다른 연산자들은 모두 스택에 삽입될 수 있다.

32. 오른쪽괄호를 만나게 되면 왼쪽괄호를 만날 때까지 스택에 쌓여있는 연산자들을 출력해야한다. while문을 반복하여 왼쪽 괄호를 만나기 전까지 스택에 있는 연산자들을 pop하여 후위 문자열에 넣는다.

33. 마지막으로 dafault는 문자가 피연산자일 때를 나타낸다. 피연산자는 여러 자리 숫자이므로 while문을 무한반복 하는데, 반복문 안에선 피연산자를 후위문자열에 넣고 중위문자열의 인덱스 값인 i를 증가시킨다. 만약 ch 값이 공백문자 또는 문자열의 끝(i==len)에 도달하면 피연산자가 끝났다는 말이므로 break로 반복문을 빠져나온다.
반복문을 break했다면 공백문자를 추가해 연산자와 피연산자를 구분한다.

34. 중위표기식의 끝에 도달하여 for문을 나온다면, 스택에 남아있는 연산자들을 후위표기식에 모두 출력할 수 있도록 한다.

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

    for (i = 0; i < len; i++) {
        ch = exp[i];
        if(ch!=' '){
            if (ch != '+' && ch!= '-' && ch != '*' && ch != '/'){
                value = getValue(&i, exp);
                dpush(&s, value);
            }
            else if( ch == '-'){
                if(isdigit(exp[i+1])){
                    value = getValue(&i, exp);
                    dpush(&s, value);
                }
                else{
                    op2 = dpop(&s);
                    op1 = dpop(&s);
                    dpush(&s, op1 - op2);
                }
            }
            else{
                op2 = dpop(&s);
                op1 = dpop(&s);
                switch (ch) {
                case '+': dpush(&s, op1 + op2); break;
            //  case '-': push(&s, op1 - op2); break;
                case '*': dpush(&s, op1 * op2); break;
                case '/': dpush(&s, op1 / op2); break;
                }
            }
        }
    }
    return dpop(&s);
}
```

35. eval함수는 후위표기식을 이용하여 계산하고, 그 결과를 출력하는 함수이다. 매개변수로는 후위표기식 문자열 exp를 전달받는다.
계산결과는 소수가 나올 수도 있기 때문에 double형으로 변수를 선언하고, 스택s에서 double형 스택을 접근하는 dpush함수와 dpop함수를 이용한다. 그리고 함수의 반환형을 double로 하여 소수 값을 반환할 수 있도록 한다.

36. 변수 i를 후위 문자열의 길이 len만큼 반복한다.
만약 ch가 피연산자라면 문자열을 숫자로 변환하여 반환해주는 getValue함수를 호출한 후 반환받은 값을 double형 스택 dpush에 push하도록 한다. getValue함수는 인덱스 i의 주소와 후위문자열을 전달한다.

37. 만약 문자가 -라면 음수 또는 연산자 -를 나타낸다.
앞에서 후위연산자로 변환할 때 음수인 경우는 -후 공백문자 없이 바로 숫자를 붙였기 때문에 이번에도 isdigit함수를 이용하여 -다음 문자가 숫자라면 문자열을 숫자로 변환하는 getValue함수를 호출하도록 한다.

38. 만약 -다음에 오는 문자가 숫자가 아니라면 연산자 -임을 알 수 있다.
연산자 -라면 스택에 들어있는 값을 2개를 반환받아 -연산을 수행한 후 그 값을 다시 스택에 push한다.

39. 나머지 else는 ch가 연산자일 때를 나타낸다. ch가 연산자라면 값 2개를 스택에서 꺼내 연산을 실행하고, 연산의 결과를 스택에 push한다.
여기서 -연산자일 땐 이미 위에서 처리했으므로 주석처리 하였다.

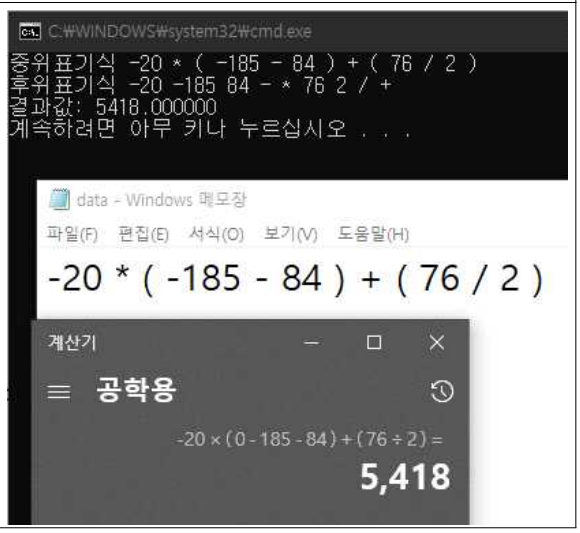
```
double getValue(int *i, char exp[]){
    int j=0;
    char ch;
    char str[50]={0};
    while(1){
        ch = exp[*i];
        if(ch==' ')break;
        str[j++]=ch;
        (*i)++;
    }
    return atof(str);
}
```

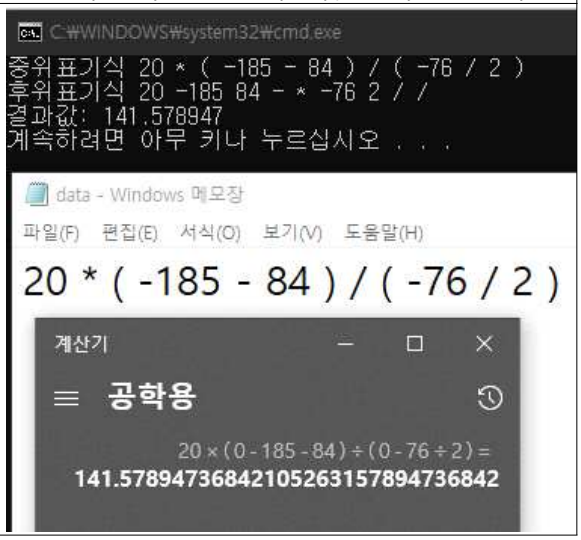
40. 문자열을 숫자로 변환하여 반환하는 함수이다.
반환형은 double이고 매개변수로는 i의 포인터(exp의 인덱스 값), 후위 연산자 문자열을 전달받는다.

41. str문자열에 연산자를 입력받고, 이를 숫자로 변환하여 반환한다.
무한반복을 돌며 exp의 인덱스 포인터 i의 값을 증가시키며 str문자열에 저장한다. 이 함수에서 i를 포인터로 선언했기 때문에 eval함수의 i변수와 같은 주소로 같은 값을 가진다.

42. ch값이 공백문자를 만나게 된다면 피연산자가 입력이 다 되었으므로 피연산자문자열의 입력을 종료한다. 예를 들어 -20.1이 피연산자라면 ch문자열에는 { '-', '2', '0', '.', '1' }이 존재하게 된다.
피연산자문자열의 입력이 끝나면 문자열을 double형으로 변환해주는 stdlib 헤더파일의 atof함수를 호출한다. 그러면 -20.1이 반환된다.

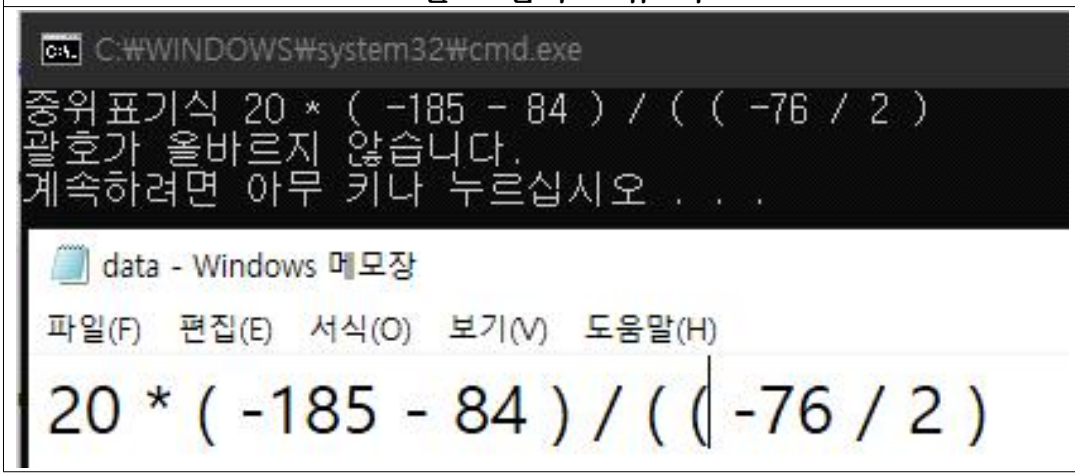
1.4 실행 창 - 아래의 값은 실제 계산 값

양수 계산, 결과 값 정수일 때	음수 계산, 결과 값 정수일 때
	

양수 계산, 결과 값 소수일 때	음수 계산, 결과 값 소수 일 때
	

소수일 때 계산

괄호 검사 오류 시



1.5 느낀 점

자료구조 실습 강의 때 중위표기식을 후위표기식으로 변환하는 프로그램과 후위표기식을 이용하여 계산을 수행하는 프로그램을 만든 적이 있었습니다. 그래서 이번 과제는 쉽게 할 수 있을 것이라고 생각했었는데, 여러 자리 숫자로 계산을 하니 한자리로 계산하는 것보다 더 복잡했고, 제가 생각했던 것만큼 쉽지 않았습니다. 다행히 교수님께서 공백문자를 이용해 연산자와 피연산자를 구분할 수 있게 해주셔서 코드가 더 복잡해지지 않았던 것 같습니다.

처음 코드를 짰을 땐 결과 값이 정수로만 출력이 되었습니다. 그 이유를 찾아보니 스택을 정수형으로 선언하여 계산과정에서 스택에 소수 값이 전달되지 않는다는 것을 알았습니다. 그래서 double형으로 스택을 하나 더 만들었고 계산함수의 반환형들 double로 설정했습니다. 그 결과, 결과 값이 소수일 때를 출력할 수 있었습니다.
그 다음으로는 음수를 처리할 방법을 생각해 보았습니다. 음수는 -로 표현하는데 연산자 -와 겹치기 때문에 이를 어떻게 구분할지 생각했습니다. 그래서 제가 생각한 방법은 -문자 다음문자가 숫자면 음수이고 공백문자이면 연산자임을 이용하여 프로그래밍 하였습니다. 그래서 문자가 숫자인지를 판별하는 ctype헤더파일의 isdigit함수를 사용하여 음수일 때와 연산자일 때를 구분하였습니다.
결과 값이 소수일 때와 음수일 때를 처리하고 나니 소수의 연산도 해볼 수 있을 것 같다는 생각이 들었습니다. 그래서 data파일에 소수를 넣고 연산을 하니 숫자변환 함수에서 소수일 때 소수점 밑의 자리를 무시하고 반환한다는 것을 알게 되었습니다. 그 이유를 찾아보니 atoi함수 때문이었습니다. 저는 atoi함수만 알고 atoi로 프로그래밍 했었는데 찾아보니 stdlib헤더파일의 atoi함수는 문자열을 int형으로 변환하는 함수였습니다. 그래서 문자열을 double형으로 변환하는 atof함수를 찾아내서 이를 수정하였습니다. 이를 계기로 헤더파일에는 제가 알고 있는 함수보다 훨씬 더 많은 함수들이 존재한다는 것을 알게 되었습니다.
이번 기회를 계기로 후위표기식으로 하는 계산에서 더 나아가 결과 값이 소수일 때와 음수일 때, 피연산자가 소수일 때의 연산을 처리할 수 있었습니다. 처음엔 어렵게 느껴졌었는데 막상 프로그래밍을 하고나니 생각보다 할 만하다고 생각했던 것 같습니다.

중위표기식과 후위표기식을 동적할당으로 프로그래밍해보면 어떨지 생각해보았습니다. 식의 길이가 길지 짧을지는 모르기 때문에 문자형 배열로 선언한다면 일단 배열크기를 크게 해야 식이 길 때를 처리할 수 있었을 것입니다. 그래서 동적할당을 하여 낭비되는 메모리가 없도록 처리했습니다. 처음엔 malloc함수를 이용하여 동적할당 했었습니다. 그런데 동적할당한 문자열을 초기화를 하지 않자 마지막에 쓰레기 값이 붙게 되었습니다. 그래서 for문을 이용하여 문자열을 초기화하려고 했지만 코드의 길이가 길어지기 때문에 고민하던 중, 할당받은 기억공간을 모두 0으로 초기화하는 calloc함수가 생각났습니다. calloc함수를 사용하면 할당을 받는 공간을 모두 0으로 초기화하기 때문에 반복문을 이용하여 일일이 초기화해야 하는 수고를 덜어줄 수 있었습니다.
처음엔 문자배열로 1024의 크기로 공간을 설정했었지만 동적할당을 하며 메모리의 공간도 아끼고 calloc을 사용하며 코드의 길이도 줄일 수 있었습니다.

이번 과제는 제가 지금까지 배워왔던 파일입력이나 동적할당, 포인터, 함수, 스택 등을 모두 사용해서 할 수 있었던 과제였던 것 같습니다. 오래전에 배운 것들도 있었기 때문에 기억이 잘 나지 않을 때도 있었지만 옛날에 배운 책들을 다시 보며 배운 것들을 기억할 수 있었던 계기가 되었습니다.
그리고 처음엔 정수계산밖에 되지 않았지만 음수계산, 그 다음은 소수계산 등 점점 더 정확한 계산이 되는 제 코드를 보며 성취감도 느낄 수 있었습니다.
처음엔 스택이라는 알고리즘을 처음 봐서 이해가 잘 되지 않았지만 이번 과제를 계기로 스택에 대해 많은 이해를 할 수 있었고 앞으로 스택에 대한 문제가 나온다면 자신 있게 풀 수 있을 것이라고 다짐했습니다.

이상 “후위표기식 변환과 계산” 레포트를 마치겠습니다.
- 감사합니다. -