Data Structure HW3

20231515
컴퓨터공학과
김다은

<목차>

1. 문제 1
2. 문제 2
3. 문제 3

문제 1.

이 문제는 자료구조 ppt 의 기본 내용을 토대로 unary operator -에 대한 연산 기능을 추가하는 문제였다. unary operator -인지 구분할 수 있는 조건은 다음과 같이 표현할 수 있다.

```
// unary operator - 고려
if((i==0 || pretoken != operand) && token == min && pretoken != rparen) {
     token = unmin;
}
```

여기서 i==0 은 수식의 맨 첫 글자임을 의미한다. 또한 pretoken!=operand 은 이전 글자가 숫자가 아님을 의미한다. 또한 token == min 은 현재 글자가 -임을 의미한다. 또한 pretoken != rparen 은 이전 글자가)가 아님을 의미한다.

infix 를 postfix 로 바꾸는 과정에서는 연산자를 저장할 수 있는 stack 를 만들어 사용한다. 또한 postfix 의 수식을 계산하는 과정에서는 숫자를 저장할 수 있는 stack 를 만들어 사용한다.

프로그램 실행 단계:

1. main 함수에서 infix_to_postfix 함수 호출됨.
2. 연산자를 stack 에 저장하는 방식으로 infix 수식을 postfix 로 변환한다.
3. main 함수에서 calculate_postfix 함수 호출됨.
4. 숫자를 stack 에 저장하는 방식으로 postfix 수식을 계산한다.

```
#include <stdio.h>
typedef enum {lparen, rparen, plus, min, times, divide, mod, eos, operand,
unmin} precedence;

int isp[] = {0,19,12,12,13,13,13,0,0,12}; // in-stack
int icp[] = {20,19,12,12,13,13,13,0,0,12}; //incoming-precedense

int isEmpty(precedence st[], int top) {
    if(top==-1) return 1;
    return 0;
}

precedence topval(precedence st[], int top) {
    return st[top];
}

precedence pop(precedence st[], int *top) {
    return st[(*top)--];
```

```
}

int pop2(int st[], int *top) {
    return st[(*top)--];
}

void push(precedence st[], int *top, precedence value) {
    st[++(*top)] = value;
}

void push2(int st[], int *top, int value) {
    st[++(*top)] = value;
}

precedence getToken(char symbol) {
    switch(symbol) {
        case '(': return lparen;
        case ')': return rparen;
        case '+': return plus;
        case '-': return min;
        case '*': return times;
        case '/': return divide;
        case '%': return mod;
        case '#': return unmin;
        case '\0': return eos;    // EOF
        default: return operand; // 숫자
    }
}

void printToken(precedence token) {
    switch(token) {
        case lparen: printf("%c", '('); return;
        case rparen: printf("%c", ')'); return;
        case plus: printf("%c", '+'); return;
        case min: printf("%c", '-'); return;
        case times: printf("%c", '*'); return;
        case divide: printf("%c", '/'); return;
        case mod: printf("%c", '%'); return;
        // case eos: printf("%c", ' '); return;    // EOF
        case unmin: printf("%c", '#'); return;
    }
}

char trans_token_to_char(precedence token) {
    switch(token) {
        case lparen: return '(';
        case rparen: return ')';
        case plus: return '+';
```

```
        case min: return '-';
        case times: return '*';
        case divide: return '/';
        case mod: return '%';
        // case eos: return '';    // EOF
        case unmin: return '#';
    }
}


char *infix_to_postfix(char postfix[]) {
    char input[21]; // 입력받은 문자열 저장
    precedence st[21]; // 연산자를 저장할 스택
    int top = 0;

    int index = 0;

    printf("Input: ");
    scanf("%s", input);

    precedence token;

    st[0] = eos;
    int i = 0; precedence pretoken;
    for(token = getToken(input[i]); token!=eos; token = getToken(input[++i]))
{
        // unary operator - 고려
        if((i==0 || pretoken != operand) && token == min && pretoken !=
rparen) {
            token = unmin;
        }

        if(token == operand) { // 숫자인 경우
            postfix[index++] = input[i];
            // printf("%c", input[i]);
        }
        else if(token == rparen) { // 오른쪽 괄호인 경우
            while(st[top]!=lparen) {
                postfix[index++] = trans_token_to_char(pop(st, &top));
                // printToken(pop(st, &top));
            }
            pop(st, &top);
        }
        else {
            while(isp[st[top]]>=icp[token])
                postfix[index++] = trans_token_to_char(pop(st, &top));
                // printToken(pop(st, &top));
            push(st, &top, token);
```

```
        }
        pretoken = token;
    }
    while((token=pop(st, &top))!=eos) {
        // printToken(token);
        postfix[index++] = trans_token_to_char(token);
    }
    postfix[index] = '\0';
    return postfix;
}

int calculate_postfix(char postfix[]) {
    int st[21];
    int top = -1;

    precedence token;
    int op1, op2;
    int i=0;

    token = getToken(postfix[i]);
    while(token!=eos) {
        if(token == operand) {
            push2(st, &top, postfix[i]-'0');
        }
        else {
            op2 = pop2(st, &top);
            op1 = pop2(st, &top);
            switch(token) {
                case plus: push2(st, &top, op1+op2); break;
                case min: push2(st, &top, op1-op2); break;
                case times: push2(st, &top, op1*op2); break;
                case divide: push2(st, &top, op1/op2); break;
                case mod: push2(st, &top, op1%op2); break;
                case unmin:
                    push2(st, &top, op1);
                    push2(st, &top, (-1)*op2);
                    break;
            }
        }
        token = getToken(postfix[++i]);
    }
    return pop2(st, &top);
}

int main() {
    char postfix[21];
    printf("Postfix: %s\n", infix_to_postfix(postfix));
    printf("Result: %d\n", calculate_postfix(postfix));
```

```
}
```

실제 입력 결과:

```
cse20231515@cspro2:~/cse3080/HW3$ gcc HW3_20231515_1.c -o p3
cse20231515@cspro2:~/cse3080/HW3$ ./p3
Input: -6
Postfix: 6#
Result: -6
cse20231515@cspro2:~/cse3080/HW3$ ./p3
Input: (1-(-3)-5)
Postfix: 13#-5-
Result: -1
```

```
cse20231515@cspro2:~/cse3080/HW3$ ./p3
Input: 3*2+4*(5-1)
Postfix: 32*451-*+
Result: 22
```

문제 2.

입력된 수식을 맨 뒤부터 읽어오면서 연산을 진행한다. 연산자를 저장할 수 있는 stack 를 만들어 사용한다.

프로그램 실행 단계:

1. main 함수에서 infix_to_prefix 함수를 호출한다.
2. 수식을 입력받고 string_length 함수를 이용해 수식의 길이를 파악한다.
3. 연산자를 stack 에 저장하면서 리턴할 prefix 문자열을 맨 뒤에서부터 채운다.
4. prefix 문자열의 시작 주소를 리턴한다.
5. main 함수에서 infix_to_prefix 의 리턴값이 출력된다.

```c
#include <stdio.h>
typedef enum {lparen, rparen, plus, min, times, divide, mod, eos, operand}
precedence;

int isp[] = {19,0,12,12,13,13,13,0,0}; // in-stack
int icp[] = {19,20,12,12,13,13,13,0,0}; //incoming-precedense
// isp[st[top]] > icp[token]

precedence pop(precedence st[], int *top) {
    return st[(*top)--];
}

void push(precedence st[], int *top, precedence value) {
    st[++(*top)] = value;
}

precedence getToken(char symbol) {
    switch(symbol) {
        case '(': return lparen;
        case ')': return rparen;
        case '+': return plus;
        case '-': return min;
        case '*': return times;
        case '/': return divide;
        case '%': return mod;
        case '\0': return eos;    // EOF
        default: return operand; // 숫자
    }
}

void printToken(precedence token) {
    switch(token) {
```

```
        case lparen: printf("%c", '('); return;
        case rparen: printf("%c", ')'); return;
        case plus: printf("%c", '+'); return;
        case min: printf("%c", '-'); return;
        case times: printf("%c", '*'); return;
        case divide: printf("%c", '/'); return;
        case mod: printf("%c", '%'); return;
        // case eos: printf("%c", ' '); return;    // EOF
    }
}

char trans_token_to_char(precedence token) {
    switch(token) {
        case lparen: return '(';
        case rparen: return ')';
        case plus: return '+';
        case min: return '-';
        case times: return '*';
        case divide: return '/';
        case mod: return '%';
        // case eos: return '';    // EOF
    }
}

int string_length(char *str) {
    int i = 0;
    while(str[i]!='\0') {i++;}
    return i;
}

char* infix_to_prefix(char prefix[]) {
    char input[21];
    int len, i;
    int index;
    precedence st[21];
    int top = 0;
    st[0] = eos;
    precedence token;

    printf("Input: ");
    scanf("%s", input);
    len = string_length(input);
    i = len-1; index = len-1;

    for(token = getToken(input[i]); token!=eos; token = getToken(input[--i]))
{
        if(token == operand) { // 숫자인 경우
            prefix[index--] = input[i];
```

```
            // printf("%c", input[i]);
        }
        else if(token == lparen) { // 왼쪽 괄호인 경우
            while(st[top]!=rparen) {
                prefix[index--] = trans_token_to_char(pop(st, &top));
                // printToken(pop(st, &top));
            }
            pop(st, &top);
        }
        else {
            while(isp[st[top]]>icp[token])
                prefix[index--] = trans_token_to_char(pop(st, &top));
                // printToken(pop(st, &top));
            push(st, &top, token);
        }
    }
    while((token=pop(st, &top))!=eos) {
        // printToken(token);
        prefix[index--] = trans_token_to_char(token);
    }
    prefix[len] = '\0';
    return &prefix[index+1];
}

int main() {
    char prefix[21];
    printf("Prefix: %s\n", infix_to_prefix(prefix));
}
```

실제 입력 결과

```
^Ccse20231515@cspro2:~/cse3080/HW3gcc HW3_20231515_2.c -o p2
cse20231515@cspro2:~/cse3080/HW3$ ./p2
Input: 3*8+7/1
Prefix: +*38/71
cse20231515@cspro2:~/cse3080/HW3$ ./p2
Input: (4-9/5)*(4/1-2)
Prefix: *-4/95-/412
```

문제 3.

숫자를 저장할 수 있는 stack 를 만들어 계산한다.

프로그램의 진행 단계는 다음과 같다.

1. main 함수에서 num 과 k 값을 입력받는다.
2. k 가 0 이 아니고, 계산하는 자리수(tmp)가 num 의 가장 큰 자리 수부터 1 의 자리 수 사이인 경우 while 문을 통해 반복한다.
   A. stack 이 비지 않은 경우, k>0 이고 stack 의 top 이 가리키는 정보가 tmp 보다 큰 경우 pop 하고 k 를 -1 한다.
   B. 이후 tmp 를 stack 에 push 하고 tmp 를 하나 작은 자리 수로 변경한다.
3. 계산되지 않은 자리 수가 존재하는 동안
   A. 남은 각 자리의 수를 모두 stack 에 push 한다..
4. k 가 0 이 아닌 동안
   A. stack 의 원소를 pop 하고 k 를 -1 해준다.
5. main 함수에서 printStackReverse 함수를 호출한다.
6. stack 에 저장된 정보를 거꾸로 출력한다.

```c
#include <stdio.h>

// stack 이 비었다면 return 1
int isEmpty(int st[], int top) {
    if(top==-1) return 1;
    return 0;
}

int topval(int st[], int top) {
    return st[top];
}

int pop(int st[], int *top) {
    int tmp = st[*top];
    (*top)--;
    return *top;
}

void push(int st[], int *top, int x) {
    (*top)++;
    st[*top] = x;
}
```

```c
void printStackReverse(int st[], int top) {
    if(isEmpty(st, top)) {
        printf("0\n");
        return;
    }
    int flag = 1;
    for(int i=0; i<=top; i++) {
        if(flag && st[i]==0) continue;
        flag = 0;
        printf("%d", st[i]);
    }
    printf("\n");
}

int main() {
    int num, k;
    int st[100];
    int top = -1, exp, tmp;

    scanf("%d %d", &num, &k);

    for(exp=1; num/exp!=0; exp*=10) {};
    exp/=10;

    while(exp!=0 && k!=0) {
        tmp = (num/exp)%10;
        // printf("<%d> ", tmp);
        if(!isEmpty(st, top)) {
            while(k>0 && topval(st, top) > tmp) {
                // printf("pop %d\n", topval(st, top));
                pop(st, &top);
                k--;
            }
        }
        // printf("push %d\n", tmp);
        push(st, &top, tmp);
        exp/=10;
    }
    while(exp!=0) {
        tmp = (num/exp)%10;
        push(st, &top, tmp);
        // printf("push %d\n", tmp);
        exp/=10;
    }
    while(k!=0) {
        // printf("pop %d\n", topval(st, top));
        pop(st, &top);
```

```
        k--;
    }
    printStackReverse(st, top);
}
```

실제 입력 결과

```
cse20231515@cspro2:~/cse3080/HW3$ gcc HW3_20231515_3.c -o p3
cse20231515@cspro2:~/cse3080/HW3$ ./p3
1432219
3
1219
cse20231515@cspro2:~/cse3080/HW3$ ./p3
10200
1
200
cse20231515@cspro2:~/cse3080/HW3$ ./p3
10
2
```

```
cse20231515@cspro2:~/cse3080/HW3$ ./p3
10
2
0
```