

자료구조 과제(HW3)

전공: 철학과

학년: 3학년

학번: 20180032

이름: 남기동

1. Postfix and Eval function

<Pseudo Code>

Variable: expr1(get infix expression from user), expr2(postfix expression converted from expr1), stack, top

Function:

1. void postfix(char stack[], char expr1[], char expr2[], int* top)

Int isp[] = { 0, 19, 12, 12, 13, 13, 13, 0, 18, 0 } (in stack precedence)

Int icp[] = { 20, 19, 12, 12, 13, 13, 13, 0, 18, 0 } (incoming precedence)

Stack[0] = EOS

For (token != EOS, getToken)

 If(token == '-') if *top == 0 then token = '#'

 If(token == '(')

 getToken and if the next token is '-'

 push '(' and token = '#'

 else push only '(' to stack

 If(token == operand) expr2[i++] = token

 Else if(token == ')')

While (stack[*top] != '(') expr2[i++] = pop

Pop (this is for eliminate '(' from the stack)

Else

While(isp(stack) >= icp(token)) expr2[i++] = pop

Push

While(token != EOS)

Expr2[i++] = pop(token)

Expr2[i] = EOS

2. int eval(char stack[], char expr2[], int* top)

While(token != EOS) {

 If(token == operand) push

 Else if (token == '#') op1 = pop and push -op1 to stack

 Else

 Op2 = pop, op1 = pop and calculating and push the result to stack

 getToken for next step in loop

} return pop(stack)

3. main function

get the string and store it into expr1

call postfix function and save the postfix expression into expr2

print expr2 as postfix expression

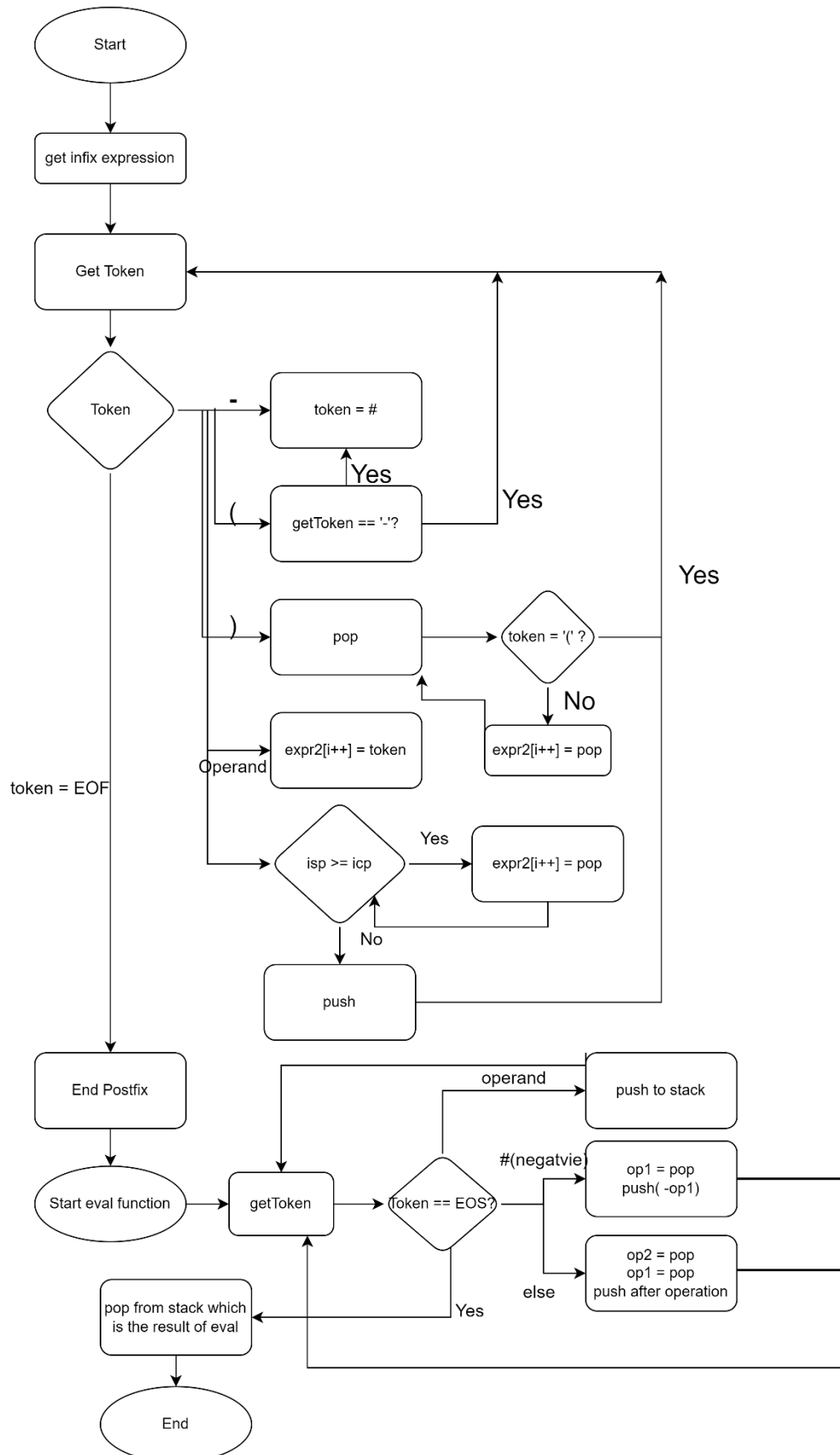
call eval function and get the result by calculating postfix expression on expr2

print result

<Test Examples>

```
cse20180032@cspro: ~/DS_Homework
cse20180032@cspro:~$ mv test DS_Homework
cse20180032@cspro:~$ ls
DS_Homework  exercise  tetrisWeek1  tetrisWeek2  week1  week2  week3  week4
cse20180032@cspro:~$ cd DS_Homework/
cse20180032@cspro:~/DS_Homework$ ls
HW3_20180032_1.c  HW3_20180032_2.c
cse20180032@cspro:~/DS_Homework$ gcc HW3_20180032_1.c
cse20180032@cspro:~/DS_Homework$ ./a.out
a.out* HW3_20180032_1.c HW3_20180032_2.c
cse20180032@cspro:~/DS_Homework$ ls
a.out HW3_20180032_1.c HW3_20180032_2.c
cse20180032@cspro:~/DS_Homework$ ./a.out
Input: -6
Postfix: 6#
Result: -6
cse20180032@cspro:~/DS_Homework$ ./a.out
Input: (1-(-3)-5)
Postfix: 13#-5-
Result: -1
cse20180032@cspro:~/DS_Homework$ ./a.out
Input: 3*2+4*(5-1)
Postfix: 32*451-*+
Result: 22
cse20180032@cspro:~/DS_Homework$
```

<Flowchart>



2. Infix to Prefix function

<Pseudo Code>

Variable: char expr[MAX_EXPR_SIZE](for storing infix expression), char operator[MAX_STACK_SIZE](stack for operator), char operand[MAX_STACK_SIZE][30] (stack for operand)

Function:

1. void prefix(char expr[], char operator[], char operand[][30])

Operator[0] = EOS

For (getToken != EOS) {

 If(token == * or / or %) op2 = getToken

 If(op2 == operand)

 Op1 = pop from operand stack

 Push (token+op1+op2) to operand stack

 Else

 Push op2 to operator stack and token to operator stack

 Else if(token == '(')

 Until pop(operator) == '('

 Op2 = pop from operand stack, Op1 = pop from operand

 Push (token+op1+op2) to operand stack

 Else

If(token == operand) push to operand stack

Else push to operator stack

}

Until (pop(operator) == EOS) {

Op2 = pop from operand stack, Op1 = pop from operand stack

Push (token+op1+op2) to operand stack

} Pop from operand stack and print it

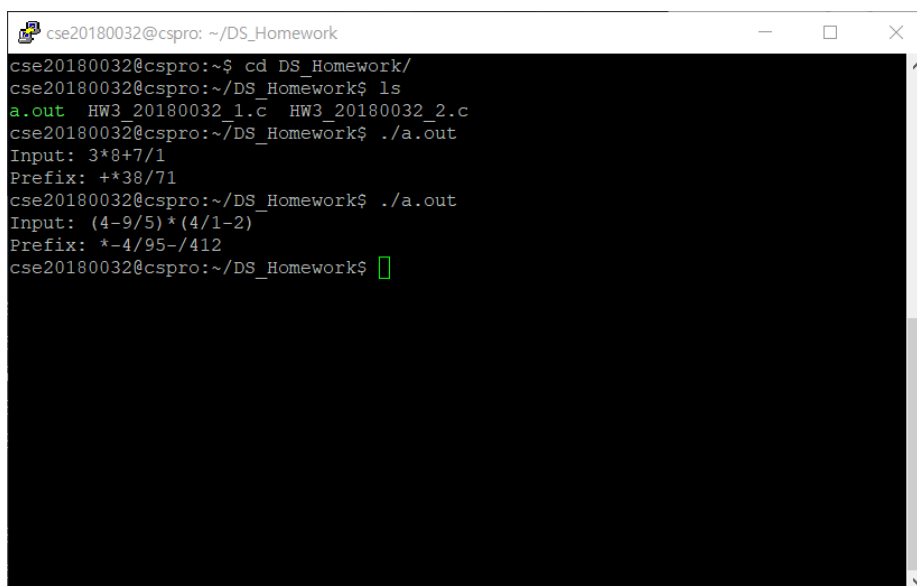
2. main function

Get infix expression and store it into expr.

Call prefix function to convert infix expression to prefix expression

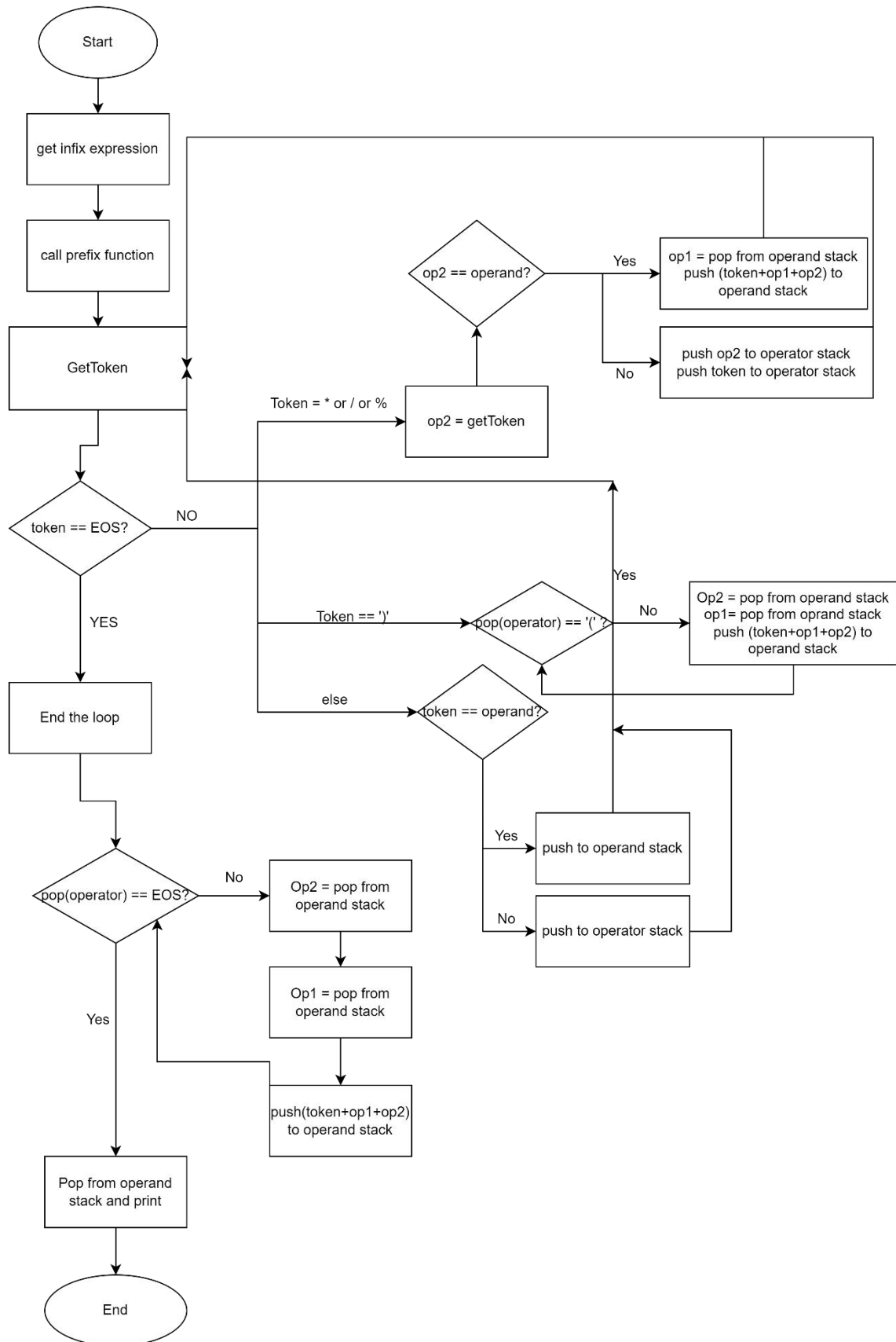
Print prefix expression

<Test Examples>



```
cse20180032@cspro: ~/DS_Homework
cse20180032@cspro:~$ cd DS_Homework/
cse20180032@cspro:~/DS_Homework$ ls
a.out  HW3_20180032_1.c  HW3_20180032_2.c
cse20180032@cspro:~/DS_Homework$ ./a.out
Input: 3*8+7/1
Prefix: ++38/71
cse20180032@cspro:~/DS_Homework$ ./a.out
Input: (4-9/5)*(4/1-2)
Prefix: *-4/95-/412
cse20180032@cspro:~/DS_Homework$
```

<Flowchart>



3. BFS in maze problem

<Pseudo Code>

Variable: fp(file pointer), int**maze(2d array that stores information of maze in input.txt), int**field(2d array that stores maze surrounded by 1 as a border), int**queue(2d array storing x, y, dir), int**move(store information of changes in x, y according to 8 different directions), int**stack(temporarily store route)

Function:

1. Main function

Open "input.txt" file and get row and col of maze

Dynamically allocate 'maze' and put each information from input.txt into maze

Dynamically allocate 'field' with row+2, col+2 and store maze surrounded by 1 as border

Call path function to solve the maze problem by BFS(Breadth First Search) method

Close the file

Free all the variables dynamically allocated

2. void path(int** field, int row_field, int col_field)

Dynamically allocate queue and stack and move

Set (1, 1) as a starting point and put into queue(add queue)

While(not found && front != rear) {


```
Queue delete(&front, &rear, queue, &row, &col)
```

```
While(dir < 8 && not found) {
```

```
    nextRow = row + move[dir][vertical]
```

```
    nextCol = col + move[dir][horizontal]
```

```
    if( (nextRow, nextCol) == (EXIT_ROW, EXIT_COL) )
```

```
        found = true,
```

```
        add_queue(&rear, queue, nextRow, nextCol, (dir+4)%8)
```

```
        pos = rear;
```

```
    else if( field[nextRow][nextCol] == 0)
```

```
        add_queue(&rear, queue, nextRow, nextCol, (dir+4)%8)
```

```
        field[nextRow][nextCol] = 1, dir++
```

```
    else dir++
```

```
    } dir = 0;
```

```
If( found ) {
```

```
    Backtrack the short route
```

```
    Print the short route
```

```
Else print "the maze does not have a path"
```

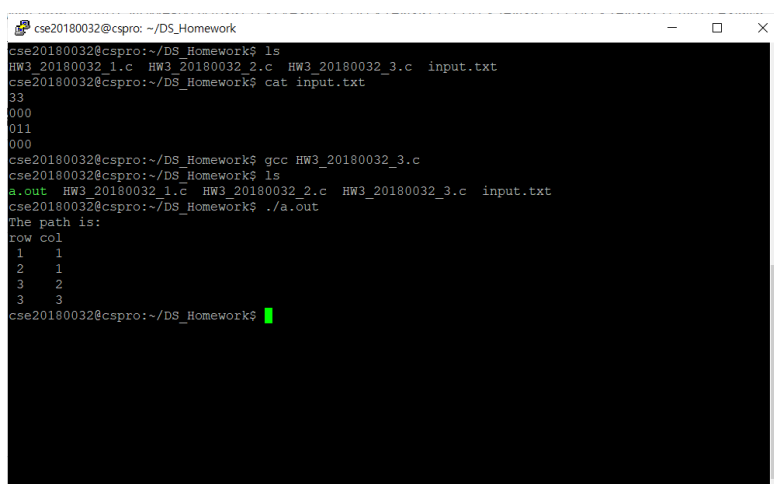
<Comparing BFS with DFS>

DFS refers to a method that starts at the root node and searches the branch

completely before moving on to the next branch. BFS is a traversal method in which the closest vertex is first visited from the starting vertex, and the farthest vertex is visited later.

Because DFS tries to search the branch completely, in the worst case, DFS needs to visit all the places in the maze. So, the worst case time complexity of the DFS method is $O(mp)$ where m and p are, respectively, the number of rows and columns of the maze. In comparison, in the worst case, BFS method also needs to visit all the places in the maze which means the time complexity of BFS method is same as DFS. However, the main point of BFS is the distance from the starting point. It expands its branch in respect to the distance. So, whenever the end point is found, BFS method can guarantee that the route BFS method find is the shortest route to the end point. Therefore, if user want to know the shortest route to the end point, BFS method is better than DFS method.

<Test Examples>



```
cse20180032@cspiro: ~/DS_Homework
cse20180032@cspiro:~/DS_Homework$ ls
HW3_20180032_1.c  HW3_20180032_2.c  HW3_20180032_3.c  input.txt
cse20180032@cspiro:~/DS_Homework$ cat input.txt
33
000
011
000
cse20180032@cspiro:~/DS_Homework$ gcc HW3_20180032_3.c
cse20180032@cspiro:~/DS_Homework$ ls
a.out  HW3_20180032_1.c  HW3_20180032_2.c  HW3_20180032_3.c  input.txt
cse20180032@cspiro:~/DS_Homework$ ./a.out
The path is:
row col
1 1
2 1
3 2
3 3
cse20180032@cspiro:~/DS_Homework$
```

<Flowchart>

