HW4

20200901 이효주

PROBLEM 1. AMAZING PROBLEM

(1) Function and variable definition

First, MAX_ROW and MAX_COL refer to the maximum row and column sizes of the maze, respectively. MAX_STACK_SIZE represents the size of the stack, and TRUE and FALSE are constants representing true and false, respectively.

Next, we define a structure called element and offsets. elements store rows, columns, and directions of the current location in the maze, and offsets store directions that can be moved. The directions that can be moved are stored in the array move.

After that, we declare the stack array and top variables globally. A mark array is used to display the locations visited when navigating the maze. EXIT_ROW and EXIT_COL indicate the exit position of the maze. The maze array stores information about the maze.

The push function is a function that adds elements to the stack, and the pop function is a function that takes elements out of the stack.

The path function is a function that explores a maze. Move from the start of the maze to the exit, adding the current position to the stack. If an exit is found, set the found variable to TRUE and output the path stored in the stack.

The main function calls the path function after receiving the size and information of the maze. The size of the maze is stored in the EXIT_ROW and EXIT_COL variables, and the maze array stores information about the maze. After checking the start and exit positions of the maze, initialize the mark array and call the path function.

(2) Approach

By default, the maze is represented by a two-dimensional array, with a starting point of (1,1) and an exit of (EXIT_ROW, EXIT_COL).
Set the rim of the maze to 1 in full to create a two-dimensional arrangement of maze[EXIT_ROW+2][EXIT_COL+2].

The algorithm starts at the starting point and searches the maze, storing the possible paths in the stack. If an exit is found, output the route. If there is no exit, the message "There is no path in the maze" is output.
If the starting point or exit is not 0, the path cannot be found, so it can be output immediately before the path() function and then returned.

- Push the starting point to the stack and mark the starting point as visited.

– While the stack is not empty, repeat the following.
Pop the top in the stack.
From the top direction, move in the order of eight directions.
If the exit is where you moved, change found to TRUE and output the path.
If the place you moved is where you can go, push it on the stack and mark it as visited.
– If the stack is empty, output the message "No path in the maze" because found is FALSE.

(3) Pseudo Code and Flow Chart

Define stack and movement offsets

Define push and pop functions for stack

Define path finding function

Set initial position as (1, 1) and mark it

Push the initial position to the stack

While the stack is not empty and the exit has not been found, pop the top element and move in the direction specified by its direction variable

Check if the next position is the exit, if so, set found to TRUE

If the next position is valid and unmarked, mark it and push it to the stack with direction incremented by 1

If the next position is invalid or marked, increment the direction variable

If found is TRUE, print the path from the starting position to the exit, else print "The maze does not have a path"


n = EXIT_ROW
m = EXIT_COL
Enter maze size
(n, m)
EXIT(1)
n < 0 OR m <0
n+2 > MAX_ROW
OR
m+2 > MAX_COL
SET MAX_ROW and
MAX_COL 32
= MAX n, m(input) is 30
Make (n+2, m+2) maze
(surrounded by 1)
Enter maze 0 or 1
in maze[i][j]
maze[1][1] != 0 OR
maze[n][m] != 0
printf("The maze does not have a path")
Set the mark[n+2][m+2] = 0
path()
set the starting point in stack, and mark
while(top > -1 && !found)
pop the top in stack
From the top direction, move in the order of 8 directions
If the exit is where you moved, change found to TRUE and output the path
found
If the place you moved is where you can go, push it on the stack and mark it as visited
not found
If the stack is empty, output the message "No path in maze" because found is FALSE

PROBLEM 2. Infix to postfix and evaluate postfix

(1) Function and variable definition

Define a stack structure. The structure has an integer variable top and a character array arr as members, and is used to implement the stack.

Defines the initStack function. This function is used to initialize the stack. Initialize top to −1.
Defines the isEmpty function. This function returns whether the stack is empty. If top is −1, it is considered empty.
Defines the isFull function. This function returns whether the stack is full. If top is MAX_SIZE − 1, consider it full.

Defines the peek function. This function returns the top element of the stack. If the stack is empty, output an error message and exit the program.

Defines a push function. This function adds elements to the stack. If the stack is full, output an error message and exit the program.
Defines a pop function. This function removes and returns the top element from the stack. If the stack is empty, output an error message and exit the program.

Defines the getPriority function. This function returns the priority of the operator. The priority is set differently depending on the type of operator.

infixToPostfix Defines a function. This function converts a formula written in the middle notation into a posterior notation. The transformed posterior notation formula is stored in the postfix array.

definePostfix function. This function calculates formulas written in backward notation. Returns the calculation result.

The main function receives a formula written in the middle notation from the user, and calls the infixToPostfix function to convert it to the backward notation. Outputs the converted posterior notation formula, and calls the evaluatePostfix function to output the calculation results.

(2) Approach

The program aims to convert formulas expressed in infix notations into postfix notations and then calculate formulas expressed in postfix notations. The operators used in the formula are calculated according to priority, and the unary operator − operator is also considered.

To do this, we first convert the formula expressed as infix notation into postfix notation by reading one character at a time. For each character in the formula.

If operand: add directly to postfix notification.

For binary operators: Pop higher priority operators from the stack, add them to the postfix notification, and then push them to the stack.
If the opening is parentheses: push to the stack.
For closing parentheses: Pop from the stack until the opening parentheses appear and add them to the postfix notification. Open parentheses only pop in the stack.
The unary operator − operator treats as follows.

If it is a unary operator: if the current character is a − operator, and it is not a parenthesis or other operator that the immediately preceding character opens, it is treated as an unary operator. In this case, push the − operator to the stack by replacing it with #.
For binary operators: If they are not binary operators, process the following characters.
When calculating a formula labeled postfix notation, for each character in the formula.

For operands: push to the stack.
If operator: pop two operands from the stack and push the result of performing that operation to the stack.
For unary operator: pop one operand from the stack and push the result of that operation back to the stack.
After converting the formula to postfix notation, calculate the formula labeled postfix notation and output the result.

(3) Pseudo Code and Flow Chart

Initialize the stack.

Read the infix expression one letter at a time and repeat the following.

If the letters read are operands (numbers), add them to the postfix expression.

If the read character is an operator,

If it is a unary operator (−), convert it to '#' and add it to the stack.

For binary operators (+, −, *, /, %) compare the priorities of operators in the stack, and if the current operator has the same or higher priority, pop the operator in the stack and add it to the postfix expression.

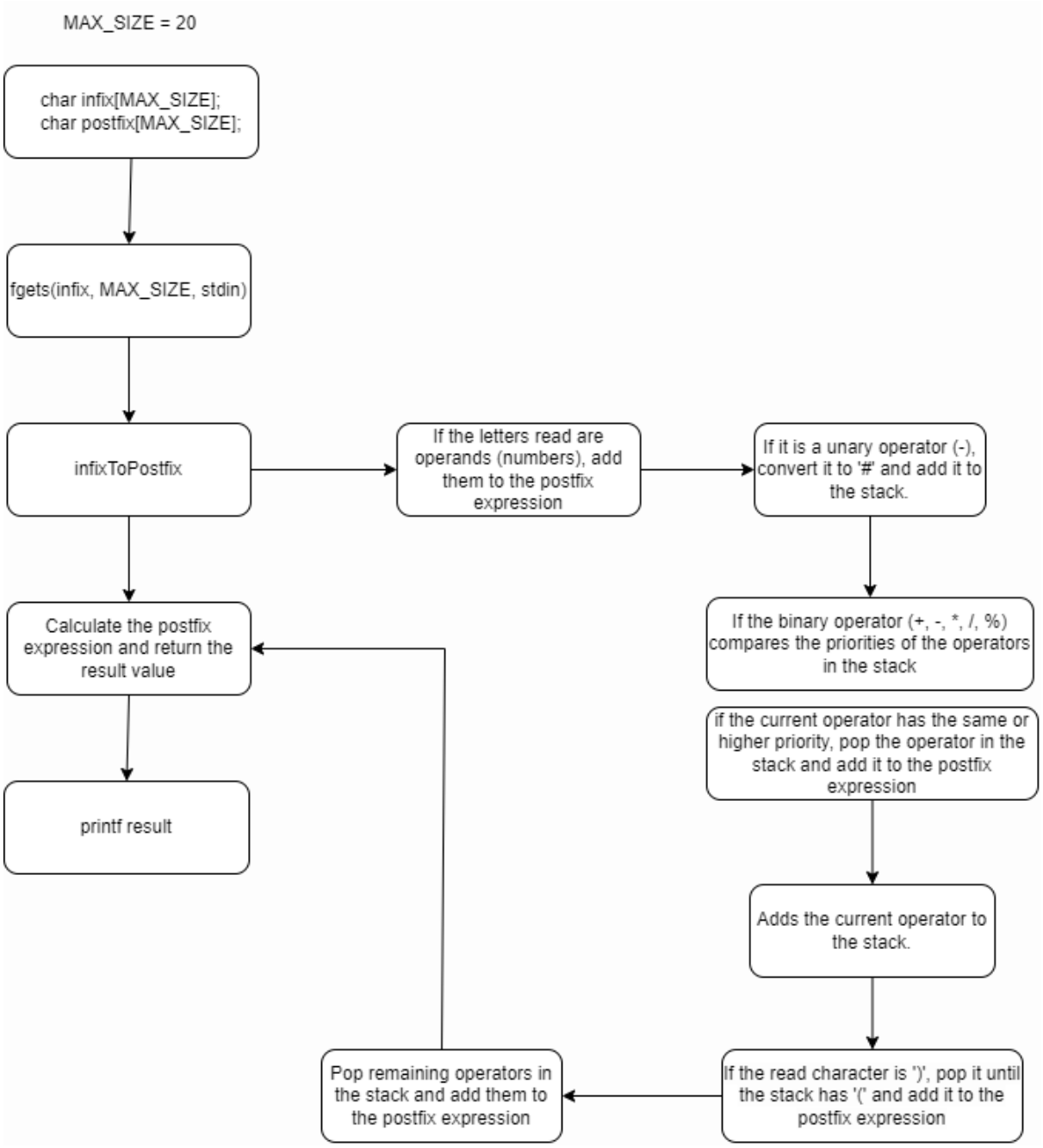
Adds the current operator to the stack.

If the read letter is '(', add it to the stack.

If the read letter is '), pop it until the stack has '(' and add it to the postfix expression.

Pop the remaining operators in the stack and add them to the postfix expression.

Returns the result value by calculating the postfix expression.


MAX_SIZE = 20
char infix[MAX_SIZE];
char postfix[MAX_SIZE];
fgets(infix, MAX_SIZE, stdin)
infixToPostfix
If the letters read are operands (numbers), add them to the postfix expression
If it is a unary operator (-), convert it to '#' and add it to the stack.
If the binary operator (+, -, *, /, %) compares the priorities of the operators in the stack
if the current operator has the same or higher priority, pop the operator in the stack and add it to the postfix expression
Adds the current operator to the stack.
If the read character is ')', pop it until the stack has '(' and add it to the postfix expression
Pop remaining operators in the stack and add them to the postfix expression
Calculate the postfix expression and return the result value
printf result