MAJOR COURSE OUTPUT 1

In partial fulfillment of the requirements in

DATA STRUCTURES AND ALGORITHMS

by

Group 9

Nicole Ashley L. Corpuz

Princess Loraine R. Escobar

Herise Janah F. Visto

S13

June 21, 2024

Major Course Output 1
Solution and Documentation

I. Project Overview and Specifications

1. Infix expressions will be converted to postfix expression
2. Evaluation postfix expressions to obtain their numerical results
3. It should handle arithmetic, logical, and relational operators as specified in the specification
4. Output the postfix expression and the evaluated result.

II. Documentation

A. Precedence of Operators

Priority	Operators
1	(,)
2	^
3	!
4	*, /, %
5	+, -
6	<. >, <=, >=
7	==, !=
8	||, &&

B. Infix Expression

- The operators are written between their operands
- Example: a + b

C. Postfix Expression

- The operators are written after their operands
- The order of evaluation of operations is left to right, with no brackets in the expression to change the order
- Example: ab+

D. Using a Stack to Convert Infix to Postfix

a. ***Algorithm***

➔ Scan symbols from left to right, and for each symbol, do the following:
- ◆ If the symbol is an operand
    - Enqueue the symbol to the postfix queue
- ◆ If the symbol is a left parenthesis
    - Push it onto the stack

    - If the symbol is a right parenthesis
        - Pop all operators from the stack up to the first left parenthesis, and enqueue it to the postfix queue
        - Discard the left and right parenthesis
    - If the symbol is an operator
        - If the precedence of the operator at the top of the stack is greater than or equal to the current operator, then pop the operator from the stack and enqueue it to the postfix queue
        - Push the current operator onto the stack
    - If no more symbol left in the infix expression
        - Dequeue the contents of postfix queue and print them on the screen

b. ***Example Conversions***

- Infix Expression: 1+2*3
    - Postfix Output: 1 2 3 * +

E. Evaluating Postfix Expression

a. ***Algorithm***

- Scan the symbols of the given postfix expression from left to right, and for each symbol, do the following
    - If the symbol is an operand
        - Push it onto the stack
    - If the symbol is an operator
        - Pop two symbols from the stack and apply the operator to these symbols
        - Push the result onto the stack
- After scanning all the symbols of the postfix expression
    - Pop the remaining symbol from the stack and print it on the screen
    - The remaining symbol is the result obtained after evaluating the postfix expression

b. ***Example Evaluation***

- Infix Expression: 1+2*3
- Evaluated value: 7

III. Implementation Details

A. Stack and Queue Data Structures

1. Stack Data Structure Implementation (stack.c and stack.h)
    - These files define and implement by using a dynamic array to store character pointers representing operators or operands.
    - Used to manage operators and operands during both infix-to-postfix conversion and postfix evaluation
    - Utilized memory allocation (malloc, free) to manage memory
    - Supports operations such as:

        - createStack: initializes a stack with a specified capacity
        - push: pushes an item onto the stack
        - pop: pops and returns the top item from the stack
        - top: returns the top item of the stack without removing it
        - isEmptyStack: check if the stack is empty
        - isFullStack: checks if the stack is full
    - Handled resizing as needed to accommodate varying numbers of elements
    - The stack follows LIFO (Last In, First Out) principles
2. <u>Queue Data Structure Implementation (queue.h and queue.c)</u>
    - These files implements a circular queue structure to store tokens of the postfix expression during infix-to-postfix conversion and handle them for postfix evaluation
    - Uses malloc and free for memory management
    - Supports the following operations:
        - createQueue: initializes a circular queue with a specified capacity
        - enqueue: allocates memory for the item and adds it to the tail of the queue
        - dequeue: removes and returns the head item from the queue
        - queueHead: retrieves the head item of the queue
        - queueTail: retrieves the tail item of the queue
        - queueEmpty: checks if the queue is empty
        - queueFull: checks if the queue is full
    - Queue follows FIFO (First In, First Out) principles

B. Algorithm Implementation

1. <u>Infix to Postfix Conversion (infix-to-postfix.c)</u>
    - This file contains the implementation of the algorithm to convert an infix expression to a postfix expression.
    - It utilizes the stack for operator management and handles parentheses to ensure correct order of operations
    - Uses a stack to manage operators and parentheses
        - Iterates through each character of the infix expression
        - Outputs operands directly to the postfix queue
        - Handles operators and parentheses based on precedence and associativity rules
        - Ensures correct grouping and order of operations
    - Supports the following functions:
        - precedence: returns the precedence of an operator
        - isOperator: checks if the character is a valid operator (single char operators, e.g: +, -, /, *)
        - isMultiCharOperator: checks if the character is a valid operator (multi-char operators, e.g. &&, ||, !=)
        - toExpression: converts the postfix queue to a string

- ◆ infixToPostfix: converts an infix expression to postfix expression

2. Postfix Expression Evaluation (evaluate-postfix.c)
   - ➔ This file implements the algorithm to evaluate a postfix expression.
   - ➔ It uses a stack to compute the result by popping operands and applying operators as per their precedence and associativity rules.
   - ➔ Supports the following functions:
     - ◆ applyOperator: performs the operation between two operands and returns the answer
     - ◆ evaluatePostfix: evaluates the postfix expression using queues and stacks and returns the final answer

C. Main Integration (main.c)

This file integrates the modules for infix-to-postfix conversion and postfix evaluation. It provides the main program flow, including input reading, processing infix expressions, generating postfix expressions, evaluating them, and printing results

IV. Limitations and Considerations

- ➔ The current implementation of the group does not support the full range of logical operators. The group only implemented the basic arithmetic and logical operators as per the specifications.
- ➔ The program currently supports integer operands only and is limited to basic arithmetic operations and comparisons.
- ➔ The program has minimal error handling and has limited feedback to the user.
- ➔ The program doesn't check for mismatched parentheses.

V. Project Coordination

A. Github [https://github.com/Ashcorpuz/CCDSALG-S13-Group9]

VI. References

GeeksforGeeks. (2023, August 21). *Pattern programs in C*. GeeksforGeeks.

https://www.geeksforgeeks.org/pattern-programs-in-c/

Queue data structure and implementation in Java, Python and C/C++. (n.d.).

https://www.programiz.com/dsa/queue

VII. Contribution and Affirmation of Original Work

The members of Group 9 from CCDSALG S13 affirm their individual contributions to the project and assert the authenticity and originality of their work. By signing below, each member acknowledges their responsibility for the submitted code and documentation .

[illegible]	[illegible]	[illegible]
06/21/2024	06/21/2024	06/21/2024
Nicole Ashley L. Corpuz	Princess Loraine R. Escobar	Herise Janah F. Visto