CCDSALG MCO1 Specifications - Expression Evaluation (Stack and Queue Application) Term 3 AY 2024 - 2025

Ryan Austin Fernandez
2401 Taft Avenue
Manila, Philippines
ryan.fernandez@dlsu.edu.ph

ABSTRACT
This document details specifications for a software project in the form of a calculator implementation. The calculator should be implemented using stacks and queues as the key data structure.

Author Keywords
data structures, algorithm analysis, stacks, queues

CCS Concepts
•**Theory of computation** → **Data structures design and analysis;**

INTRODUCTION
Let us apply what you learned about stacks and queues. For this project, you will develop your own (simple) calculator. Read, understand, and comply with the project specs described below.

This document is divided into seven sections: your objectives for this project, a description of the software you will develop, the list of deliverables, the groupwork policies, non-submission policies, honesty and intellectual property policy, the grading rubrics, and the reference list.

OBJECTIVES
Your general objective in this study is to understand an algorithm for evaluating expressions in the C language and to implement it using stack and queue data structures.

The specific objectives are:

1. To build a working implementation of the stack and queue data structures in the C language;
2. To implement an algorithm for infix to postfix conversion based on a textual explanation;
3. To practice modular programming for large or complex systems;
4. To implement a proper software testing procedure for the developed application; and
5. To write a documentation of the developed application.

Design and implement your stack and queue data structures. You will need two stacks, one for storing operators and another for storing operands. You will also need a queue for storing the tokens corresponding to the postfix expression.

Practice modular programming. That is, compartmentalize the data structures and operations by storing the implementation codes in separate source code files. For example, the codes for the stack data structure are stored in stack.h and stack.c, while the codes for the queue data structure are stored in queue.h and queue.c. Do not use the goto statement.

Implement the algorithms for (a) infix to postfix conversion, and (b) postfix expression evaluation. Again, practice good programming. Compartmentalize each algorithm implementation in separate source code files, for example, infix-to-postfix.c, evaluate-postfix.c. The tokens comprising the postfix expression should be stored in a queue.

Integrate the modules. Create the main module, which will include the other modules, and call the appropriate functions to achieve the task.

Test your solution. Create test cases and perform exhaustive testing. You should test each operator and the grouping symbols. You should test that the operator hierarchy and operator associativity are correctly observed. Create at least 50 test expressions. Do not forget to test the division by zero error.

Hint: Use input and output redirection to minimize keyboard input. Encode your infix expression in a text file. Run your .exe file in the command line. For example:

```
CCDSALG> solution < infix.txt > result.txt
```

where solution is the exe file, infix.txt is a text file containing the test cases (infix expressions), and result.txt is the text file that will contain the corresponding output.

Document your solution. Give a brief description of how you implemented the data structures. For example, did you use arrays? Did you use a linked list (dynamic memory allocation)? Did you use a circular queue? Give a brief description of how you implemented the algorithms. Disclose what is not

ACM ISBN 978-1-4503-2138-9.
DOI: 10.1145/1235

working (for example, your solution does not handle logical operators).

Documentation Format

Abstract
Provide a summary of the paper, including a summary of the implementation details of the stack and queue data structures, the algorithms, the testing process and results, and any limitations of the implementation (what features do not work?)

Stack and Queue Implementation
Describe how your group implemented the stack and queue data structures from scratch.

Infix to Postfix Implementation
Describe how you implemented the infix to postfix conversion. What data structures did you use? How did you integrate these into the algorithm?

Postfix Expression Evaluation Implementation
Describe how you implemented the postfix expression evaluation. What data structures did you use? How did you integrate these into the algorithm?

Testing Process
How did you come up with your test cases? What types of test cases did you include? What were the results of your testing? What fixes did you have to implement after specific iterations of testing?

Limitations
What features of your application do not fully work? What challenges did you encounter in attempting to get these features to work? What attempts did you make in trying to make these work?

References
Provide a complete list of references in APA format. Failure to do so will result in a grade of 0.0 for this major course output.

Appendix A: Record of Contribution
Provide a record of contribution for each major activity for the case study.

1. Each row represents an activity, and each cell represents the percentage each member contributed to that activity. Each row's contributions must total to 100.

2. The **TOTAL** row must be normalized to add up to 100.

Have each member sign the member list above the table, above their printed full name.

Example:

Group Members:

- P1: Sonic The Hedgehog
- P2: Miles "Tails" Prower
- P3: Knuckles the Echidna

Activity	P1	P2	P3
Stack Implementation	50.0	25.0	25.0
Queue Implementation	33.3	33.3	33.3
Infix to Postfix Conversion	25.0	50.0	25.0
Postfix Evaluation	10.0	10.0	80.0
Testing	10.0	10.0	80.0
Documentation	10.0	10.0	80.0
Raw Total	138.3	138.3	323.3
TOTAL	**23.05%**	**23.05%**	**53.88%**

SOFTWARE DESCRIPTION

Input Description

The input is an INFIX expression. For simplicity, assume that the tokens making up the INFIX expression are stored in a string with at most 256 characters, including the NULL byte. To simplify the task further, assume that NO spaces separate the tokens and that the expression is syntactically valid.

Handle the following tokens:

- Operands are limited to non-negative integers, i.e., 0 and higher. There is no need to consider negative integers or floating-point values. For logical operators only: assume that the operands are limited to 0 and 1 only.
- Operators include:
  - Arithmetic operators: +, -, *, %, ^(caret is for exponentiation)
  - Relational operators: >, <, >=, <=, !=, ==
  - Logical operators: !, &&, ||
  - Parentheses as grouping symbols ( )

Example Inputs
- Input #1: 1+2*3
- Input #2: (8+2)/3
- Input #3: 5>=3
- Input #4: 1&&0
- Input #5: !(2>1&&3<2)
- Input #6: 123/0

Output Description

The required output, i.e., printed output, are:

1. POSTFIX expression - separate two consecutive tokens with exactly one SPACE character
   - For example input #1, the output should be: 1 2 3 * +
   - For example input #5, the output should be: 2 1 > 3 2 < && !
   - For example input #6, the output should be: 123 0 /
2. Evaluated value of the postfix expression
   - For example input #1, the evaluated value should be: 7
   - For example input #2, the evaluated value should be: 3

- For example input #3, the evaluated value should be: 1
- For example input #4, the evaluated value should be: 0
- For example input #5, the evaluated value should be: 1
- For example input #6, there is no evaluated value. Print instead: Division by zero error!

Notes

- The division operator is applied to integer operands. Following C language notation, the result should also be an integer. For example, 10/3 gives a result of 3 (see Example input #2).
- : If the denominator is zero, do not perform the actual division since it will cause a run-time error. Your program should check this case, and if it occurs, your program should print the constant string "Division by zero error!" (see Example input #6).
- For relational and logical operators: the result should be limited to a 0 and 1, where 0 means False and 1 means True (see Example input #3 to #5).

Required Program Interaction

Figure 1 shows an example of the required program interaction. Those in black bold font represent the user input. Those in blue bold font represent the output postfix expression. Those in green bold font represent the output evaluated value. The red bold font represents an error message due to division by zero. Those in brown are not part of the interaction—they serve as explanations to help you understand what each line means.

The program should run in a loop when executed. The user inputs the infix expression (as a string). The program will then output the postfix expression on the next line and the corresponding evaluated value on another line. To terminate the loop, the user inputs QUIT (all in capital letters).

The program should have a MINIMALIST interaction. This means that you should NOT have any prompts and that you should NOT print anything else except the required output. Non-compliance will result in point deductions.

DELIVERABLES

Submit via Canvas a ZIP file that contains:

- Source files for the stacks, queues, conversion and evaluation algorithms, main module
- Test case file named TESTCASE.TXT, which contains at least 50 sample infix expressions that you used for testing your solution. Each infix expression should be encoded in a separate line of text.
- Corresponding RESULT.TXT that contains the result.
- Documentation named <GROUPNAME>.PDF. For example "Team Sonic.PDF"

Do NOT include any EXEcutable file in your submission.

Deadline for submission of all deliverables is Monday, July 7, 2025, 5:00 PM.

It is highly recommended that you use Overleaf when typesetting your document. Your document should look like this document you are reading right now. The read-only template for this specifications document can be found here https://www.overleaf.com/read/crbcyyvhfcwy#0eee4d. You must make a copy of the Overleaf project to make edits.

Follow the file naming convention: CCDSALG_<section>_<lastnames of members>.zip, e.g., CCDSALG_S11_theHedgehogProwertheEchidna.zip

WORKING WITH GROUPMATES

You are to accomplish this project in collaboration with your fellow students. Form a group of at least two to at most three members. You may only select groupmates from the same section. Make sure that each group member makes approximately the same contribution to the project. Problems with groupmates must be discussed internally within the group and, if needed, with the teacher.

NON-SUBMISSION AND LATE SUBMISSION POLICY

If the project is not submitted three days after the due date, it will result in a score of 0 for this graded assessment.

Late submissions will receive a 10-point deduction per day (counted every second beyond 5:00 PM on July 7, 2025). No late submissions will be accepted after July 10, 2025, 5:00 PM.

HONESTY POLICY AND INTELLECTUAL PROPERTY RIGHTS

Honesty policy applies. You are encouraged to read and gather the information you need to implement the project. However, please note that you are NOT allowed to borrow and/or copy-and-paste – in whole or in part any existing related program code from the internet or other sources (such as printed materials like books, or source codes by other people not online). You should develop your own code from scratch by yourselves, i.e., in cooperation with your groupmates.

Cheating or intellectual dishonesty is punishable with a final grade of 0.0 in the course.

RUBRIC FOR GRADING

The grading rubrics are shown in Table 1.

Please post any questions or concerns in the Canvas Discussion thread. Thank you for your cooperation.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. 2009. *Introduction to Algorithms, Third Edition* (3rd ed.). The MIT Press.

[2] C.J.A. Delfinado. *Data structures and algorithms*. https://books.google.com.ph/books?id=EcBUswEACAAJ

[3] Mark Allen Weiss. 1995. *Data structures and algorithm analysis (2nd ed.)*. Benjamin-Cummings Publishing Co., Inc., USA.

```
1+2*3                          // user inputs the infix expression
1 2 3 * +                      // the program prints the postfix expression on the next line
7                              // the program prints the evaluated value on the next line

!(2>1&&3<2)                    // user inputs the infix expression
2 1 > 3 2 < && !               // the program prints the postfix expression on the next line
1                              // the program prints the evaluated value on the next line

123/0                          // you user the infix expression
123 0 /                        // the program prints the postfix expression on the next line
Division by zero error!        // the program prints the error message on the next line

QUIT            // type QUIT in all caps to quit the program
```

Figure 1. Sample Interaction for CCDSALG MCO1 AY2425 T3

Table 1. Grading Rubric for CCDSALG MCO1 AY2425 T3

CRITERIA	COMPLETE	INCOMPLETE	NO MARKS
Stack (20 points)	Correctly implemented the stack data structure. (20 points)	Implementation is not entirely correct. (10 points)	Not implemented. (0 points)
Queue (20 points)	Correctly implemented the queue data structure.	Implementation is not entirely correct. (10 points)	Not implemented. (0 points)
Infix-to-postfix (25 points)	Correctly implemented the infix-to-postfix conversion algorithm. Processed all operators and grouping symbols correctly. (25 points)	Implementation is not entirely correct. Some operators and grouping symbols were not processed correctly. (12 points)	Not implemented. (0 points)
Postfix Evaluation (20 points)	Correctly implemented the postfix evaluation algorithm. Processed all operators correctly. (20 points)	Implementation is not entirely correct. Did not process all operators correctly. (10 points)	Not implemented. (0 points)
Documentation (10 points)	Required details were covered. (10 points)	Some required details were not discussed. (5 points)	No discussion or documentation. (0 points)
Compliance with Instructions (5 points)	All instructions were complied with. (5 points)	Deduction of 1 point for every instruction not properly complied with. Examples: included an EXE file in the ZIP file, file naming not followed.	More than four instructions not complied with.
Maximum Total Points: 100			