**Software Development: Data Structures and Performance (SCQF Level 8)**

**HL9T 35**

**Assessment**

**Outcomes 2 and 3**

#### Creative Industries

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Student** | | Kamil Milej | | | | | | |
| **I.D.** | | 2264629 | | | | | | |
| **Date** | | 16/06/2025 | | | | | | |
|  | **Pass** | |  | **Fail** |  | **Remediation** |  |  |
| **Tutor** | |  | | | | | | |

#### Creative Industries

***Dijkstra’s shunting yard algorithm Documentation***

* *Algorithm Description*

The aim of the project was to create an application that converts an arithmetic expression in infix format (2 + 3 \* 4) to an equivalent postfix expression (i.e. so-called reverse Polish notation, 2 3 4 \* +). The Shunting Yard algorithm, developed by Edsger Dijkstra, was used to accomplish this task.

The application uses two custom data structures:

MyStack – implemented using std::vector<char> to manage operators

MyQueue – implemented using std::queue<std::string> to manage the output postfix expression.

Program used :

* Accepts a string input of an infix expression.
* Tokenises the input into operands, operators, and parentheses.
* Rearranges these tokens into correct postfix order using operator precedence and associativity rules.
* Displays the result using a test driver.
* *Data Structure used*

**1. MyQueue**

**Purpose:**  
Stores the output postfix tokens in correct order.

**Implementation Summary:**  
Internally uses std::queue<std::string>.

**Methods:**

* void enqueue(const std::string& val) - Adds an item to the end of the queue.
* std::string dequeue() - Removes and returns the item from the front, throws exception if empty.
* bool isEmpty() const – Returns true if queue is empty.

**2. MyStack**

**Purpose:**  
Used to temporarily hold operators and parentheses during processing.

**Implementation Summary:**  
Internally uses std::vector<char> as the underlying container.

**Methods:**

* void push(char c) – Pushes an operator onto the stack.
* char pop() – Pops the top operator; throws exception if empty.
* char top() const – Returns the top operator without removing it; throws exception if empty.
* bool isEmpty() const – Returns true if stack is empty.

Description of the toPostfix() function

The central element of the program is the toPostfix(const std::string& expression) function, whose task is to convert an arithmetic expression in infix notation (e.g. "2+(3\*4)") to the equivalent postfix form (i.e. "2 3 4 \* +"). This function is based on the Shunting Yard algorithm proposed by E. Dijkstra and has been implemented in a way that takes into account all the important rules of the order of operations.

During operation, the function can recognize multi-digit numbers and the +, -, \* and / operators, maintaining the correct order of their processing in accordance with the established precedence: multiplication and division have higher priority than addition and subtraction. Additionally, the function correctly interprets parentheses, ensuring that expressions contained in parentheses will be calculated first. Only integers are supported, without the need to process floating-point numbers or commas.

* *Test complexity analysis*

To make sure that the function works correctly in different situations, a test function runTest(const std::string& name, const std::string& expr) is provided, which allows you to check specific input cases. Example tests include simple expressions without parentheses (e.g. "2+4+8-2"), more complex expressions with mixed operators and different precedences ("2-1+3/3+12\*5"), and nested parentheses cases ("2+(2+2)\*2\*(2/2)"). The result of the function is displayed in the console as a string of tokens in postfix notation.

* *Project Notes*

The project has been divided into several separate header and source files (MyQueue.h, MyStack.h, Parser.h, Parser.cpp, main.cpp), which allows for maintaining order in the code and facilitates its development and testing. The naming of classes, methods and variables is clear, and the comments clearly describe how individual parts of the program work. To implement the solution, only the standard C++ library was used, without additional external components.

* *Conclusion*

The application successfully implements Dijkstra’s Shunting Yard Algorithm using custom data structures for stack and queue. The system meets all basic outlined assessment requirements, processes integer expressions correctly, and is thoroughly tested. The modular code structure and documentation ensure the solution is readable, reusable, and easy to maintain.