The Programming Assignment Report Instructions
CSCE 221

1. The description of an assignment problem.

   The purpose of this assignment was to create a C++ program that would allow a user to enter an expression with or without variables in infix form. The simple calculator would then evaluate the expression by utilizing a stack, a que, and postfix form. The stack and que also used the templated linked list class created in part 1.

2. The description of data structures and algorithms used to solve the problem.

   (a) Provide definitions of data structures by using Abstract Data Types (ADTs)

   - **Templated Doubly Linked List**: a sequence of nodes where each node contains a pointer to the previous node and next node in the sequence. Since it is templated, a variety of data types can be passed in.
   - **Templated Linked Que:** a linear data structure where operations are performed based on first-in-first-out principle. Since it is templated, a variety of data types can be passed in.
   - **Templated Linked Stack:** a linear data structure where the operations are performed based on last-in-first-out principle. Since it is templated, a variety of data types can be passed in.
   - **String:** a container that holds an array of characters
   - **Vector:** a container that can hold a variety of data types. Vector has several built in functions. Also provides bounds error checking.
   - **Token:** a container that references a character to a value and/or weight.

   (b) Write about the ADTs implementation in C++.

   - **Templated Doubly Linked List:** a list of doubly linked nodes. This data type has a constructor, copy constructor, desctructor, assignment operator, get first node pointer, get after last node, check if list is empty, return first node, return last node, insert a new node to the first position, remove the first node, insert a node in the last position, and remove last node. An overloaded output stream operator was used to display the all the nodes in the list. Finally everything was templated for use with a variety of data types.
   - **Templated Linked Que:** contains a private doubly linked list variable. Functions include a constructor, copy constructor, destructor, return first element in que, check if que is empty, enqueue an item, dequeue an item, and an accessor function for the doubly linked list. An overloaded output stream operator was used to display the all the elements in the que. Finally everything was templated for use with a variety of data types.

- **Templated Linked Stack:** contains a private doubly linked list variable. Functions include a constructor, destructor, check if stack is empty, push an element to the stack, pop the top element of the stack, get top of the stack, and an accessor function for the doubly linked list. An overloaded output stream operator was used to display the all the elements in the stack. Finally everything was templated for use with a variety of data types.
- **String:** a constant char array
- **Vector:** a container that holds a list of elements that can be accessed using an iterator.
- **Token:** a struct that I created to hold an elements kind, value, and weight. Functions include a checker if element is an operator, checker if element is an operand, checker if an element is a number, checker if an element is a variable, setter functions that sets the value, function to convert character to a double, and a get operator weight function.

(c) Describe algorithms used to solve the problem.

- $Parser :: toPostfix()$ this functions converts a vector of tokens in infix form to an equivalent postfix form. This functions iterates through each token in the vector. If the token is an operand, enqueue the operand until the next item is not an operand. If the token was an close parenthesis, pop items off the stack and enqueue that item until a left parenthesis is reached. If the token is an open parenthesis, push the item to the stack. if the token is a #, pop an item of the stack and enqueue it until the stack is empty. Else, pop an item off the stack and enqueue it until the top item of the stack's weight is greater or equal to the tokens weight. Then enqueue the item. Finally, pop items off the stack and enqueue them until the stack is empty. Return the que.
- $Token :: get\_operator\_weight()$ if kind is (or), weight is 1, if kind is $+$ or $-$, weight is 2, if kind is $*$ or $/$, weight is 3, if kind is ˆ, weight is 4, if kind is ˜, weight is 5. Return weight
- $Evaluator :: evaluate$: taken in operator, v1, and v2. If operator is $+$, add v1 and v2. If operator is $-$, subtract v2 from v1. If operator is $*$, multiply v1 and v2. If operator is ˆ, raise v1 to the power of v2. If operator is $/$ and v2 is not 0, divide v1 by v2. If v2 was 0, throw a division by 0 exception.
- $Evaluator :: getValue()$

(d) Analyze the algorithms according to assignment requirements.

- $Parser :: toPostfix() : \ O(n)$
- $Token :: get\_operator\_weight() : \ O(1)$
- $Evaluator :: evaluate : \ O(1)$
- $Evaluator :: getValue()$

3. A C++ organization and implementation of the problem solution

(a) Provide a list and description of classes or interfaces used by a program such as classes used to implement the data structures or exceptions.

- Parser
- Evaluator

- **Templated Linked Que:** contains a private doubly linked list variable. Functions include a constructor, copy constructor, destructor, return first element in que, check if que is empty, enqueue an item, dequeue an item, and an accessor function for the doubly linked list. An overloaded output stream operator was used to display the all the elements in the que. Finally everything was templated for use with a variety of data types.
- **Templated Linked Stack:** contains a private doubly linked list variable. Functions include a constructor, destructor, check if stack is empty, push an element to the stack, pop the top element of the stack, get top of the stack, and an accessor function for the doubly linked list. An overloaded output stream operator was used to display the all the elements in the stack. Finally everything was templated for use with a variety of data types.
- RuntimeException

(b) Include in the report the class declarations from a header file (.h) and their implementation from a source file (.cpp).

(c) Provide features of the C++ programming paradigms like Inheritance or Polymorphism in case of object oriented programming, or Templates in the case of generic programming used in your implementation.

Templated

- LinkedQue
- LinkedStack

Inheritance

- RuntimeException

4. A user guide description how to navigate your program with the instructions how to:

(a) compile the program: specify the directory and file names, etc.

(b) run the program: specify the name of an executable file.

5. Specifications and description of input and output formats and files

   (a) The type of files: keyboard, text files, etc (if applicable).

      No input or output files

   (b) A file input format: when a program requires a sequence of input items, specify the number of items per line or a line termination. Provide a sample of a required input format.

   (c) Discuss possible cases when your program could crash because of incorrect input (a wrong file name, strings instead of a number, or such cases when the program expects 10 items to read and it finds only 9.)

6. Provide types of exceptions and their purpose in your program.

   (a) logical exceptions (such as deletion of an item from an empty container, etc.).

      - QueueEmptyException
      - StackEmptyException

   (b) runtime exception (such as division by 0, etc.)

      - *DivisionByZeroException*() in evaluator.h

7. Test your program for correctness using valid, invalid, and random inputs (e.g., insertion of an item at the beginning, at the end, or at a random place into a sorted vector). Include evidence of your testing, such as an output file or screen shots with an input and the corresponding output.