1. For each of the following postfix expressions, illustrate the operation of the stack-based evaluation algorithm. Show the contents of the stack after each operand or operator symbol from the input is processed. Also indicate the value of the expression or give an error message if the expression is not syntactically valid.
2. 4 2 48 2 / - +

4

4 2

4 2 48

4 2 48 2

4 2 24

**-**18 -22

1. 2 3 2 ^ ^ 6 +

2

2 3

2 3 2

512 9

518 6

1. 2 3 ^ 2 ^ 2 –

2

2 3

8

8 2

64

64 2

62

1. 7 4 3 4 ^ + -

7

7 4

7 4 3

7 4 3 4

7 4 81

7 85

-78

1. For each of the following infix expressions, illustrate the operation of the algorithm for converting infix to postfix. Show the contents of the stack and the output stream after each token from the input is processed. If the infix expression is not syntactically valid, give an error message:
2. 2 + 3 ^ 4

Postfix: 2 3 4 ^ +

Stack: 2

2 3

2 3

2 81

83

1. (2 - 6) ^ 3

Postfix: 2 6 – 3 ^

Stack: 2

2 6

-4

-4 3

-64

1. (2 + 6) / (32 - 4 \* 7)

Postfix: 2 6 + 32 4 7 \* - /

Stack: 2

2 6

8

8 32

8 32 4

8 32 4 7

8 32 28

8 4

2

1. Using the vector<char> implementation of stacks, show what the vector looks like after each line of code below, and give the value of s.top( ). Also give the value of x. Part of the answer is given to you for the first two lines.

Stack<char> s; // [], s.top ( ) =? x not defined

char x = 'a'; // [], s.top( ) = ???, x = 'a'

s.push('w'); // ['w'], s.top( ) = 'w', x = 'a'

s.push('b'); // [‘b’, ‘w’], s.top()=’b’, x=’a’

x = s.top(); // [‘b’, ‘w’], s.top()=’b’, x=’b’

s.pop(); // [‘w’], s.top()=’w’, x=’b’

s.push('y'); // [‘y’, ‘w’], s.top()=’y’, x=’b’

s.push('z'); // [‘z’, ‘y’, ‘w’], s.top()=’z’, x=’b’

s.pop(); // [‘y’, ‘w’], s.top()=’y’, x=’b’

x = s.top(); // [‘y’, ‘w’], s.top()=’y’, x=’y’

s.push('a'); // [‘a’, ‘y’, ‘w’], s.top()=’a’, x=’y’

x = s.top(); // [‘a’, ‘y’, ‘w’], s.top()=’a’, x=’a’

s.pop(); // [‘y’, ‘w’], s.top()=’y’, x=’a’

1. Using the circular array implementation of queues, with an array of size 4, show the array after each operation in the sequence below. (Show the array, and the values of theFront, theBack, and currentSize. Also show the value of variable x.)

int x;

queue<int> q; // [ , , , ], theFront = 0, theBack = 3, currentSize=0 , x=?

q.push(2); // [2 , , , ], theFront = 0, theBack = 0, currentSize=1 , x=?

q.push(3); // [2 ,3 , , ], theFront = 0, theBack = 1, currentSize=2 , x=?

q.push(4); // [2 ,3 , 4, ], theFront = 0, theBack = 2, currentSize=3 , x=?

x = q.front(); // [2 ,3 , 4, ], theFront = 0, theBack = 2, currentSize=3 , x=2

q.pop(); // [3 , 4, ], theFront = 1, theBack = 2, currentSize=2 , x=2

q.pop() // [4 ], theFront = 2, theBack = 2, currentSize=1 , x=2

q.push(5); // [4, 5 ], theFront = 2, theBack = 3, currentSize=2 , x=2

q.push(6); // [4, 5, 6 ], theFront = 2, theBack = 0, currentSize=3 , x=2

q.push(7); // [4, 5, 6, 7 ], theFront = 2, theBack = 1, currentSize=4 , x=2

x = q.front(); // [4, 5, 6, 7 ], theFront = 2, theBack = 1, currentSize=4 , x=4

q.pop(); // [5, 6, 7 ], theFront = 3, theBack = 1, currentSize=3, x=4

q.pop(); // [6, 7 ], theFront = 0, theBack = 1, currentSize=2, x=4

q.pop(); // [7 ], theFront = 1, theBack = 1, currentSize=1, x=4

x = q.front(); // [7 ], theFront = 1, theBack = 1, currentSize=1, x=7

q.pop(); // [, , , ], theFront = 0, theBack = 3, currentSize=0, x=7

q.push(8); // [8, , , ], theFront = 0, theBack = 0, currentSize=1, x=7

1. The following code is modified from the code we talked about in class. Find what the difference(s) are and describe any difference in how the code would perform.

// Return iterator corresponding to the first node containing an item x.

// Iterator isPastEnd if item is not found.

template <class Object>

LListItr<Object>& LList<Object>::find( const Object & x ) const {

LListNode<Object> \*p = header->next;

while( p->element != x && p != NULL )

p = p->next;

return LListItr<Object>( p );

}

The difference is in that the LList<Object> is passed by reference.