CS1020E: DATA STRUCTURES AND ALGORITHMS I

Tutorial 6 – Stacks and Queues

(Week 8, starting 3 October 2016)

1. Stack and Queue Operations

In this exercise, we will use the STL Container **adapter** classes stack<T> and queue<T> to create stack and queue instances, and execute their operations, in the program below. Draw diagrams representing the contents of stack s1, stack s2 and queue q after each operation:

```
#include <stack>
#include <queue>

using namespace std;

int main () {
    queue<int> q;
    stack<int> s1, s2;

    s1.push(3);
    s1.push(2);
    s1.push(1);

    while (!s1.empty()) {
        s2.push(s1.top());
        s1.pop();
        if (!s1.empty())
            s2.push(s1.top());
        q.push(s2.top());
    }

    s1.push(q.front());
    q.pop();
}
```

What is the **default underlying data structure** below a stack<T>? queue<T>?

Which data structures can you **choose to use** as the underlying data structure for each of the two adapters?

2. Stack and Queue Applications – Expression Evaluation

In the Lisp programming language, each of the four basic arithmetic operators appears before an arbitrary number of operands, which are separated by spaces. The resulting expressions are enclosed in parentheses. There is only one operator in a pair of parentheses. The operators behave as follows:

- `( + a b c )` returns the sum of all the operands, and `( + )` returns 0.
- `( - a b c )` returns a - b - c - … and `( - a )` returns 0 - a.
  The minus operator must have at least one operand.
- `( * a b c )` returns the product of all the operands, and `( * )` returns 1.
- `( / a b c )` returns a / b / c / … and `( / a )` returns 1 / a, using **double division**
  The divide operator must have at least one operand.

You can form larger arithmetic expressions by combining these basic expressions using a fully parenthesized prefix notation. For example, the following is a valid Lisp expression:

```
( + ( - 6 ) ( * 2 3 4 ) )
```

The expression is evaluated successively as follows:

```
( + -6.0 ( * 2.0 3.0 4.0 ) )
( + -6.0 24.0 )
18.0
```

Design and implement an algorithm that uses up to 2 stacks to evaluate a legal Lisp expression composed of the four basic operators, integer operands, and parentheses. The expression is well formed (i.e. no syntax error), there will always be a space between 2 tokens, and we will not divide by zero.

Output the result, which will be one double value.

Work hard preparing for midterms