

School of Computer Science and Software Engineering  
Clayton Campus, Monash University  
**CSE1303 Part A**  
**Semester II, 2001**

## Tutorial 2: Stacks and Queues

Please attempt all starred questions before your tutorial class.

### Exercise 1.\*

Assume **MAXSTACK** has been defined as **5**, and **Stack** and the functions operating on **Stack** have been defined as in lectures. Show the contents of **theStack** after each **pop** and **push** in the following code. For **theStack.entry** show only the items in the array which correspond to items on the stack.

```
float  x;
float  y;
Stack  theStack;

initializeStack(&theStack);

push(&theStack, 1.2);
push(&theStack, 1.0);
x = pop(&theStack);
y = pop(&theStack);
push(&theStack, y - x);
push(&theStack, 4.9);
push(&theStack, 5.1);
x = pop(&theStack);
y = pop(&theStack);
push(&theStack, y + x);
x = pop(&theStack);
y = pop(&theStack);
y *= x;
```

What is the value of **y** at the end of the computation?

### Exercise 2.

Write a C function **float topElement(const Stack\* stackPtr)** which returns the top element of the stack and doesn't change the stack.

### Exercise 3.

Write a C function **void printStack(const Stack\* stackPtr)** which prints all the elements in the stack, from the top down, and doesn't change the stack.

**Exercise 4.**

Write a C function **int stackSize(const Stack\* stackPtr)** which returns the number of items in the stack, but doesn't change the stack.

**Exercise 5.\***

Assume **MAXQUEUE** has been defined as **5**, and **Queue** and the functions operating on **Queue** have been defined as in lectures. Show the contents of **theQueue** after each **append** and **serve**. For **theQueue.entry** show only those items in the array that correspond to items in the queue.

```
float x;
int n = 10;
Queue theQueue;

initializeQueue(&theQueue);

while (n != 0 && !queueFull(&theQueue))
{
    append(&theQueue, n);
    n /= 2;
}

while (!queueEmpty(&theQueue))
{
    x = serve(&theQueue);
    append(&theQueue, --x);
    if (x > 4)
    {
        break;
    }
}
```

What is the value of **x** at the end of the computation?

**Exercise 6.**

Write a C function **void printQueue(const Queue\* queuePtr)** which prints all the items in the queue, from front to rear, and doesn't change the queue.

**Exercise 7.**

Assume that **Queue** has been declared as above. Design and code a function **reverseQueue(Queue\* queuePtr)** that uses a stack of floats to reverse all the entries of a given queue. Use only the functions given in lectures. You may assume that the queue and the stack can contain the same number of objects.

**Exercise 8. (Preparation for the prac)**

Reverse Polish notation is a different way of entering mathematical expressions which does not have the precedence problems that “infix” notation does. The table below is also in the prac notes, and explains how “infix” notation is translated to reverse Polish.

<i>Infix</i>	<i>Reverse Polish</i>
$a + b * c$	$a \ b \ c \ * \ +$
$a + (b * c)$	$a \ b \ c \ * \ +$
$(a + b) * c$	$a \ b \ + \ c \ *$
$a * b + c$	$a \ b \ * \ c \ +$
$a * (b + c)$	$a \ b \ c \ + \ *$
$(a + b) * (c - d)$	$a \ b \ + \ c \ d \ - \ *$
$(a + b) * (c - d) / (e + f)$	$a \ b \ + \ c \ d \ - \ * \ e \ f \ + \ /$

Write the following “infix” expressions in Reverse Polish Notation

- a)  $(8 + 7) * (3 + 2) + 4 / 3$   
 b)  $(3 - 4 + 2) * 7 + 1$

Write the following Reverse Polish expressions in infix notation

- c)  $5 \ 3 \ + \ 4 \ 7 \ - \ * \ 2 \ /$   
 d)  $7 \ 2 \ 3 \ + \ * \ 5 \ -$

**Additional Exercises**

Kruse et al.

- Exercises 3.1 E6
- Exercises 4.3 E4, E8, E9,
- Review 3.1 Q1
- Review 4.1 Q1, 2, 3

Deitel & Deitel (2<sup>nd</sup> Edition)

- Self Review Exercises 12.1 a, e, j, 12.3
- Exercises 12.10, 12.11, 12.12