COMP SCI TEST WEEK PREP

Important topics:

1. Data Structures
2. Pseudocode
3. Testing

DATA STRUCTURES

Three types of Abstract Data Types:

Stacks:

- Operate on a FILO (First In Last Out) basis meaning that elements are "stacked" on top of each other and then processed from the top of the stack down.

Much like a stack of plates.

Examples of usage:
- Memory Management
- Expression Evaluation
- Backtracking in recursion

Adding an element to the stack is known as 'pushing' while removing from the stack is known as "popping"

Queues:

- A queue as a data structure operates much like how a queue of individuals at, for example, a bank operates.

- The individual who enters the queue first is dealt with first.
- Hence, it can be said that queues on a first in - first out (FIFO) basis

- Examples:
  - Management of files sent to printer
  - Buffers used with keyboards
  - Task Scheduling

Adding an element to a queue is known as 'enqueueing' whereas removing an element is known as 'dequeueing'.

Linked Lists:

- A linked list is a list of certain elements we call "nodes"
- Each node has two parts: the data and the pointer
- when going through a linked list, one will not necessarily go from top to bottom in a perfect order.
  - Instead, starting from the first element, the computer will move to where this first node points, to find the second node and the second node points to the third, etc.

IMPLEMENTATION

STACKS

Note: Popping an item from a stack doesn't have to be a simple deletion, it can also mean "extracting" that one item to be used elsewhere.

IMPT THINGS TO REVIEW:

- Description of the process of adding items to: ✓
  1. Linked Lists
  2. Queues
  3. Stacks
- Pseudocode of adding/removing items from linked list (from recorded lecture) ×
- Going over different types of testing
- 3 9691 p3s.
- why is modular algorithm design good.
- The remaining past papers.
- Re-read Pgs 91, 95 - 97, 190 - 194 ✓
- Verify how many different types of testing there really are

ADTS : PSEUDOCODE

STACKS

Setting it up:

```
DECLARE stackVar : ARRAY [1:10] OF INTEGER
DECLARE topPointer, basePointer, stackFull : INTEGER
topPointer ← 0
basePointer ← 1
stackFull ← 10
```

Adding an item (Pushing):

```
INPUT item
IF topPointer = stackFull
   THEN
      OUTPUT "Stack is full, cannot push"
   ELSE
      topPointer ← topPointer + 1
      stackVar[topPointer] ← item
ENDIF
```

Removing an item (Popping):

```
IF topPointer < basePointer
   THEN
     OUTPUT "Stack is empty, cannot pop"
   ELSE
     item ← stackVar[topPointer]
     topPointer ← topPointer - 1
ENDIF
```

QUEUES

Setting it up

```
DECLARE queueVar : ARRAY [1:10] OF INTEGER
DECLARE frontPointer, rearPointer, queueFull, queueLength : INTEGER

frontPointer ← 1
rearPointer ← 0
queueFull ← 10
queueLength ← 0
```

Adding an item (enqueueing)

```
IF queueLength < queueFull
  THEN
```

```
        IF rearPointer < upperBound
            THEN
                rearPointer ← rearPointer + 1
            ELSE
                rearPointer ← lowerBound
        ENDIF
        queueVar[rearPointer] ← item
        queueLength ← queueLength + 1
    ELSE
        Output "Queue is full, cannot enqueue"
ENDIF
```

Removing an item (Dequeueing)

```
IF queueLength = 0
    THEN
        OUTPUT "Queue is empty - cannot dequeue"
    ELSE
        item ← queueVar[frontPointer]
        IF frontPointer < upperBound
            THEN
                frontPointer ← frontPointer + 1
            ELSE
                frontPointer ← lowerBound
        ENDIF
ENDIF
```

TYPES OF TESTING

During development:

Walkthrough
- A formal version of manually dry running the code
- Specific test cases are first written and then run through the program
- The entire process is thoroughly documented.

White box testing
- Every route through a program is tested
- All possible ways the program can run are tested using different sets of test data

Black box testing
- Only the input and the corresponding actual vs. expected outcome is tested.