Name:________ KEY ________ (print, last first MI) SSN:______________

GTA:_______________________ Lab:_______ Lecture:__________

Test Instructions

**READ THIS NOW!**

Failure to read this cover page and follow the described instructions may result in serious repercussions! Failure to read these directions will NOT constitute an excuse or defense.

Fill in your NAME and ID-NUMBER accurately on the test. Failure to do this correctly, will result in an inaccurate score being recorded. Fill in your student information below, sign the pledge and fill in the last 4 digits of your social security number in the box at the bottom of each page.

Answer all questions with respect to material as discussed in class. In any questions/answers which require a distinction between integer and real values, integers will be represented without a decimal point, whereas real values will have a decimal, [1704 (integer), 1704.0 (real)].

All answers on the test must be legible. Do not make assumptions about a problem, if you believe a problem can be interpreted differently ask your instructor Respond to each question with one answer only, multiple responses and questions with omitted responses will be counted incorrect. No calculators or electronic devices may be used during this exam. GOOD LUCK!

This is a closed book, closed notes test. It is an honor code violation to discuss, (in any form: written, verbal or electronic), any portion of this test with any other students, (regardless of whether they are taking the course or not), until the tests have been returned. It is also an honor code violation to have a copy of this test, (in any form: written, verbal or electronic), in your possession outside of the test examination classroom, without the instructors written permission.

Virginia Tech honor code pledge:

“I have neither given nor received unauthorized assistance on this test.”

---

**KEY**

signature
I. ADT views

(20 points)

Label the following three views of the Drop-Out Stack data structure, discussed in class, as either being the physical, logical (abstract), or application view:

1. Used in an editor to provide a limited “undo” facility.
   
   1. **Application**

2. Implemented as a circular array of stack elements, using the modulus %, operator to increment the top & bottom indexes.
   
   2. **physical**

3. A modified stack structure such that when ‘overflow’ normally occur, the bottom element is deleted and the new element placed on top.
   
   3. **logical or abstract**

Label the following two general design descriptions of a separately compiled data structure as either being an example of information hiding or encapsulation:

1. All data structure storage types and ADT variable definitions (excluding the element type), are declared in the implementation (private .cc) section of the ADT module. The data structure is not passed to the function operations, but is referenced globally within the implementation module .cc file).
   
   1. **Encapsulation**

2. All data structure storage types (excluding the element type), are declared in the interface (public .h) section of the ADT module. The data structure is passed to the function operations whenever called. Instances of the ADT are not defined in the implementation module .cc file).
   
   2. **Information hiding**

II. Queues

(30 points)

Assume that you have access to an existing implemented queue data type that stores and operates upon the type of data elements that you need for a program on which you are currently working. However, your program actually requires a Dequeue, (double-ended queue) data type. You do not have time to implement the dequeue from scratch. You realize that all you really need to do is implement the enqFront and deqRear operations using the existing standard queue operations.

Assumptions:

Each element in the queue is unique. *(Not really necessary)*

The standard queue operations have already been implemented.
Restrictions:
Only standard queue operations may be assumed to be available for use.
enqFront and deqRear must be implementation independent,
(i.e. NO access to the underlying queue representation may be made!)
[DO NOT implement any of the standard queue operations!]

```c
void enqFront (queType& que, qElemType qElem )
{
    queType    tmpQueu;
    qElemType  tmpElem;
    InitializeQueue(tmpQue);  
    Enque(qElem, tmpQue);  
    // if (!FullQueue(que)
    while(!EmptyQueue(que)) {  
        Deque(que, tmpElem);
        Enque(tmpElem, tmpQue);  
    }  
    // InitializeQueue(que);  
    while(!EmptyQueue(tmpQue)) {  
        Deque(tmpque, tmpElem);  
        Enque(tmpElem, que);  
    }
}
}

void deqRear (queType& que, qElemType& qElem)  
{
    queType    tmpQueue;  
    qElemType  tmpElem;  
    if(!EmptyQue(que)) {  
        InitializeQueue(tmpQue);  
        Deque(que, qElem);  
        while(!EmptyQue(que)) {  
            Enque(qElem, tmpQue);  
            Deque(que, qElem);  
        }  
        // InitializeQueue(que);  
        while(!EmptyQueue(tmpQue)) {  
            Deque(tmpQue, tmpElem);  
            Enque(tmpElem, que);  
        }
    }
}
```
III. Algorithm Analysis  

Perform a **Worst-Case** Order of Magnitude Analysis on the following Function  

`ColsTurnedOn` (which returns the number of columns in the two-dimensional `Boolean` matrix that contains a **TRUE** value somewhere in the column).

```c
typedef Boolean BitRay[N, N];

int ColsTurnedOn (BitRay matrix)
{
    Boolean found;
    int row, col, ColsOn;

    (A) ColsOn = 0;
    (B) for (col = 0 ; col < N, col++) {
        (C) row = 0;
        (D) found = FALSE;
        (E) while ( ( !found ) && (row < N) ) {
            (F) found = matrix[row][col];
            (G) row++;
        }
        (H) if ( found )
            (I) ColsOn++;
    } 
    (J) return (ColsOn);
}
```

*Worst-Case* Order of Magnitude = $O(N^2)$

Give the exact running time complexity. $T(N) = \frac{3N^2 + 5N + 2}{N}$
The analysis must be shown for partial credit.

$$T(n) = A + B (C + D + E (F + G) + H + I) + J$$

$$= 1A + \sum_{i=1}^{N} (1B + 1C + 1D + \sum_{j=1}^{N} (1E + 1F + 1G) + 1H + 1I) + 1J$$

$$= 2 + \sum_{i=1}^{N} (5 + 3\sum_{j=1}^{N} 1)$$

$$= 2 + \sum_{i=1}^{N} (5 + 3N) = 2 + 5\sum_{i=1}^{N} 1 + 3N\sum_{i=1}^{N} 1 = 2 + 5N + 3N^2$$
IV. Recursion (15 points)

Trace the following recursive function and answer the questions below.

```c
int hmmm ( int a, int b )
{
    if ( b == 0 )
        return ( 1 );
    else if ( b == 1 )
        return ( a );
    else if ( (b % 2) == 0 )
        return ( hmmm(a*a, b/2) ) ;
    else
        return ( hmmm(a*a, (b-1)/2) * a ) ;
}
```

(1) What is the value returned by the call `hmmm (2, 10)`?

1024

(2) How many recursive calls occur during the execution of the above call, NOT counting the external call `hmmm (2, 10)`?

3

(3) What is the function actually computing (i.e. what would be a more descriptive name for the function)?

\( a^b \) or power

Trace

\[
\begin{align*}
\text{hmmm}(2,10) & = \text{hmmm}(4,5) \\
& = \text{hmmm}(16,2) \times 4 \\
& = \text{hmmm}(256,1) \times 4 \\
& = 256 \times 4 \\
& = 1024
\end{align*}
\]
V. Linked-Lists  (15 points)

Given the following instance of an array of structs (records) double linked-list implementation:

\[
\begin{align*}
\text{list.first} = 4 ; \quad \text{list.last} = 5 ; \quad \text{AVAIL} = 9 ; \\
\text{nodes} \\
\begin{array}{cccc}
\text{index} & \text{item} & \text{next} & \text{back} \\
0 & L & 6 & -1 \\
1 & J & 3 & 4 \\
2 & Z & 8 & -1 \\
3 & O & 7 & 1 \\
4 & E & 1 & -1 \\
5 & Y & -1 & 7 \\
6 & N & -1 & -1 \\
7 & T & 5 & 3 \\
8 & K & 0 & -1 \\
9 & X & 2 & -1 \\
\end{array}
\end{align*}
\]

(1) Give the graphic diagram for the list, (fill in and complete the following diagram)?

Execute each of the following code segments upon the structure. Reproduce the diagram structure as it would result from the execution of each code segment. Assume each code segment starts with the original array above, (i.e., each code segment is independent of the execution of the others).

\[
\text{/* 2 */} \quad \text{nodes[list.last].next} = \text{list.first} ; \\
\text{nodes[list.first].back} = \text{list.last} ; \\
\]

( makes the list circular )

\[
\text{/* 3 */} \quad \text{for (p=list.first; p > 0 ; p=nodes[p].back) \{} \\
\text{q = nodes[p].back ;} \\
\text{nodes[p].back = nodes[p].next ;} \\
\text{nodes[p].next = q ;} \\
\}
\]
list

(first last)

(reverses the list)

E

J

O

T

Y

back item next