FALL 2018 CS 401 MIDTERM EXAM OCT 17-18 (Instructor Tim Hoffman)

A PITT USER NAME IS AN EMAIL ADDY w/out @PITT.EDU. LIKE THIS: **JFK63**

PRINT –**YOUR**- PITT USERNAME ➔ _____

PRINT FIRST & LAST NAME LEGIBLY ➔ _____

**There are 22 questions on this exam. 1 – 20 worth 5pts each**
5pts for Q#4 ➔ #4a=2pts #4b=2pts #4c=1pts
5pts for Q#5 ➔ #5a=1pts #5b=2pts #5c=2pts
5pts for Q#6 ➔ #6a=3pts #6b=2pts
5pts for Q#20 ➔ #20a=1pt #20b=1pt #20c=1pt #20d=1pt #20e=1pt
**Questions 21 & 21 are Extra Credit 1 point each.**

**YOU MUST HAND IN YOUR ANSWERS ONLINE BY GOING TO THE HANDIN SITE AND UPLOADING TO THE MIDTERM LINK. THE HANDIN WILL ONLY ACCEPT A FILE NAMED Midterm.java**

**AFTER HANDING IN ONLINE YOU SHOULD GET FEEDBACK JUST LIKE ANY OTHER ASSIGNMENT. ALSO YOU MUST SEE YOUR NAME IN GREEN ON THE PROJECTOR. IF YOUR NAME IS IN RED YOUR ANSWER FILE HAS MISSING ANSWERS OR INCORRECT FORMAT.**

**AS SOON AS YOU SEE YOUR NAME IN GREEN ON THE PROJECTOR YOU MUST HAND ME YOUR PAPER EXAM. THEN YOU MAY LEAVE THE ROOM.**

**IF YOU DON'T HAND ME YOUR SIGNED PAPER EXAM BEFORE LEAVING THE ROOM, YOU WILL GET A 0 FOR THE MIDTERM**

Two pieces of code are said to be “execution equivalent” if they each produce the same output. Determine if the following pairs of code fragments are execution equivalent.

**#1** This Pair (below) is EQIVALENT? T / F ?

```
for( int i=0 ; i<10 ; ++i)
       System.out.print(i + " ");
```

```
methQ1( 10 ); // in main

// - - - - - - - - - - - - -
static void methQ1( int n)
{
   if (n==0) return;
   methQ1( n-1 );
   System.out.print(n + " " );
}
```

**#2** This Pair (below) is EQIVALENT? T / F ?

```
for( int i=10 ; i>0 ; --i )
     System.out.print( i + " ");
```

```
methQ2( 10 ); // in main

// - - - - - - - - - - - - -
static void methQ2( int n)
{
       if (n>10) return;
       methQ2( n+1 );
       System.out.print( n + " " );
}
```

**#3** What is the output of this nested loop?

FALL 2018 CS 401 MIDTERM EXAM OCT 17-18 (Instructor Tim Hoffman)

```java
import java.io.*;
import java.util.*;

public class P3
{
    public static void main(String[] args)
    {
    for ( int i= 0 ; i < 3 ; ++i )
        {
            for (int j=0 ; j <= (3-i) ; ++j )
                System.out.print( "*" );
            System.out.println();
        }
    }
}
```

A:

```
****
***
**
```

B:

```
****
***
**
*
```

C:

```
*
**
***
****
```

D:

```
*****
****
***
**
*
```

FALL 2018 CS 401 MIDTERM EXAM OCT 17-18 (Instructor Tim Hoffman)

Assume **boolean b1 = true, b2 = true; int i = 79, j = 0;**

**#4a** b1 || (j>0) || (b2 && i==100) evaluates to: T / F
**#4b** b1 && (b2 || i>j) && (b1 != b2) evaluates to: T / F
**#4c** (b2 || !b2) evaluates to: T / F

Assume these definitions: **String s1,s2;**
**After each statement, put V if it is VALID (i.e. would compile) else put I if INVALID (i.e. would not compile).**

**#5a** s1 = "Hello" – "World"; V / I ?
**#5b** s2 = 3.0 + "14159"; V / I ?
**#5c** s1 = "3" + .14159; V / I ?

Assume these definitions: int i = 7; boolean b=false;
then trace this code

```
if (i>0)
{
    b = (i % 2) != 0;
    i = (i%2 == i)  || b;
}
else
{
    i +=  i + (7%i);
    b = false;
}

if (i%2==0 || !b)
{
          i = 7;
          b = true;
     }
}
else
{
    if (b)
         i = 9 - (i%5);
    else
         b = (i%2) != 0 ;
}
```

**#6a** final value of b is T / F ?
**#6b** final value of i = ____ ← a number

**#7** What is the output of this code segment?

```
1  import java.io.*;
2  import java.util.*;
3
4  public class P7
5  {
6      public static void main(String[] args)
7      {
8          int[] arr = new int[10];
9          arr = fillArr(arr);
10         System.out.println(arr[5]);
11     }
12     public static int[] fillArr(int[] arr)
13     {
14         arr = new int[5];
15         for (int i = 0; i < arr.length; i++)
16             arr[i] = i * 2;
17         return arr;
18     }
19 } // END CLASS P7
```

A 10
B 0
C index out of bounds error
D null pointer exception
E none of the above

**#8** Here is an insertInOrder method**. Assume the bSearch works perfectly.** On what line does insertInOrder do something that shows us that the writer of the code does not understand what bSearch *contributes* to the insertInOrder method?

```
1  static void insertInOrder( int[] arr, int count, int key   )
2  {
3      int index = bSearch( arr, count, key ); // ASSUME IT WORKS PERFECTLY
4      if ( index < 0) index = -(index+1);     // NOW WE KNOW WHERE BELONGS
5      int i=count-1;
6      while( i>=0 && key<arr[i] )
7      {
8          arr[i+1] = arr[i];
9          --i;
10     }
11     arr[i+1] = key;
12 }
```

Write a line number: ____ ← ( between 1 and 12 incl.)

**#9** With a correctly written bSearch that is correctly used in a correctly written insertInOrder, what is the maximal number of times we should have to compare the key to an array element in terms of N? (Assume one single solitary call to insertInOrder)

A O(1)
B O(log2N)
C O(N)
D O(N*log2N)
E O(N squared)

**#10** Same assumptions as above except now we count moving an array element as the operation to be counted.
What is the maximal number of times we ever would need to move an array element in terms of N?

A O(1)
B O(log2N)
C O(N)
D O(N*log2N)
E O(N squared)

**#11** What is the output of the following program?

```
1  import java.io.*;
2  import java.util.*;
3
4  public class P11
5  {
6      public static void main(String[] args)
7      {
8          System.out.println( mystery( 1234 ) );
9      }
10     static int mystery( int n )
11     {   if (n==0) return 0;
12         if ( n%2 == 0 ) return n + mystery( n/10 );
13         return mystery( n/10 );
14     }
15 }
```

_____ ← write the number

**#12** Looking at problem #11 above, what is the maximum number of copies of the mystery method that will be on the stack **at any one time**?

_____ ← write the number

**#13** What is the output of the following program?

```
1  import java.io.*;
2  import java.util.*;
3  public class P13
4  {   public static void main(String[] args)
5      {   int[] arr = { 5,1,7,3,77,22,55,99,31,54,10 };
6          System.out.println( mystery(arr, 0 ) );
7      }
8      static int mystery( int[] a, int idx )
9      {   if ( idx == a.length-1 ) return a[idx];
10         if ( a[idx]>a[idx+1] )
11         {   int tmp=a[idx];
12             a[idx]=a[idx+1];
13             a[idx+1]=tmp;
14         }
15         return mystery( a, idx+1 );
16     }
17 }
```

_____ ← write the number

**#14** According to the "array discipline" the correct way store values into the array is:

A keep the initialized values packed tight to the front of the array
B use the count variable to determine where the next value gets stored
C increment the count variable as soon as / same time you add an element into the array
D all of the above

**#15** What is the LOGICAL length of arr. i.e. according to the array discipline how many valid elements are in it?

```
1  import java.io.*;
2  import java.util.*;
3
4  public class P15
5  {
6      public static void main(String[] args)
7      {
8          int[] arr = { 1,2,3,4,5,6,7,0,0,0 };
9          int count = 5;
10     }
11 }
```

_____ ← write the number

**#16** Look at the following program. It scans an array of ints looking for the index of the last actual initialized valid element (i.e. put there by the user) in the array. This array conforms to the array discipline. The count is accurate and the actual initialized values put in by the user are packed tight to the front. If there are no actual initialized values in the array then it is supposed to return -1.

```
1  import java.io.*;
2  import java.util.*;
3
4  public class P16
5  {
6      public static void main(String[] args)
7      {
8          int[] arr = { ?, ?, ?, ?, ?, ?, ?, ?, ?, ? }; // 10 int values but you have NO IDEA what they are
9          int count = 5; // ASSUME THE COUNT ACCURATE & ALL INITIALIZED VALS PACKED TIGHT TO FRONT
10     }
11     static indOfLastInitializedElement( int[] arr, int count )
12     {
13         for (int i=0 ; i<arr.length ; ++i )
14         {
15             if (arr[i]==0) return i-1; // find first zero. elem before it must be last 'actual' value
16         }
17     } // IF WE RETURN -1 IT MEANS THERE WERE NO ACTUAL INIALIZED VALUES IN THE ARRAY
18 }
```

A there is an off by one error
B throws an index out of bounds error
C this code is correct, the first occurrence of a zero is a number put there by compiler
D this code completely misses the whole point of the array discipline
E none of the above

**#17** According to the "array discipline" the correct way to delete/remove the last initialized value in a plain array:

A overwrite that value with 0 or -1 or some marker/sentinel value
B resize the array to have a new .length of exactly count-1
C copy the value to the right of that element over top of the one you want to delete
D decrement your count variable

**#18** According to the "array discipline" the count variable represents:

A the number of initialized values in the array
B the maximum capacity of the array
C the number of unused cells in the array
D the index of the last initialized element

**#19** According to the "array discipline" the correct way to append a new value to the end of the array is: *(you may assume the array has an empty slot for the new value)*
A resize the array to .length + 1
B shift all the elements one place to the left
C copy the new value into array[0]
D copy the new value into array[count] and increment your count variable

**#20a** Choose the **best** description for this array:
int[] arr = { 3,6,21,8,7 };

A a null array
B a zero length array
C an empty array
D a partially filled array
E a full array

**#20b** Choose the **best** description for this array:
int[] arr = new int[0];

A a null array
B a zero length array
C an empty array
D a partially filled array
E a full array

**#20c** Choose the **best** description for this array:

```
int[] arr = new int[5]; arr[0]=5; int count=1;
```

A a null array
B a zero length array
C an empty array
D a partially filled array
E a full array

**#20d** Choose the **best** description for this array:

```
int[] arr = new int[5]; int count=0;
```

A a null array
B a zero length array
C an empty array
D a partially filled array
E a full array

**#20e** Choose the **best** description for this array definition:

```
int[] arr;
```

A a null array
B a zero length array
C an empty array
D a partially filled array
E a full array

**#21 OPEN RESPONSE** Why is it sub-optimal to solve the jumbles problem by generating all the permutations of a word?

**#22 OPEN RESPONSE** When we upSized our arrays we created a new array that was twice the length of the one that got full. Why didn't we just increase it by one … and avoid having an array that has a lot of wasted extra unused capacity when we reach end of file?