Summative Assessment of Baccalaureate Computer Science: The Graduating Senior Test - Version 1

The Seidenberg Faculty
About this issue:

One function of the Technical Reports series is to archive faculty efforts that may be useful in the future but do not qualify as publications. An example of this came up last month, in June, at a meeting to consider organizing a programming contest or perhaps a competition on information technology suitable for a larger population of contestants. Technical Report #192, describing the format and presenting the content of an earlier CSIS programming/"trivia" contest, proved helpful. In the same vein, the present report may serve as a resource when building new instruments for program assessments.

The instrument herein was the first attempt on the part of the CS faculty at a performance-based inquiry into how well graduating seniors were learning content felt to be central to the major. Although it was not completed and administered until the spring of 2006, the test's creation began in 2004 in response to the School's then new assessment plan that called for summative program assessments. Initially, the CS Curriculum Committee reviewed the Major Field Test in computer science published by the Educational Testing Service. While the MFT was deemed unsuitable because the questions were at variance with the curriculum, in form it served as the model to emulate. Like the MFT, the homegrown test was to be a comprehensive, two-hour exam, mainly consisting of short-answer questions, that yielded an overall score as well as scores on "subfields" (i.e. the test's ten sections).

Unlike the MFT, the homegrown test was not constructed and validated by experts in educational measurement. Nor could a homegrown test yield comparisons of the performance of our seniors with those from benchmark schools. Despite these shortcomings, our test offered direct evidence of a large assortment of specific proficiencies, hence made a creditable assessment vehicle. Its chief faults were: (i) a lengthiness stemming from the committee's including virtually all contributed items (lest anyone be offended and cool toward assessment), and (ii) a coverage bias reflecting the skills and understandings deemed focal by those who participated in its construction.

Relative to the test's construction, thanks go to Mary Courtney, who was chairperson of the Computer Science Curriculum Committee during the two years over which this test was constructed; Andreea Cotoranu, the Seidenberg School's Associate Director of Assessment; and to the CS faculty members who contributed including Mehdi Badii, Constantine Coutsas, Joseph Malerba, Narayan Murthy, Richard Nemes, Sotirios Skevoulis, Sylvester Tuohy, and Carol Wolf.

Allen Stix, Editor
Department of Computer Science  Name:  Answers

Summative Assessment for the Computer Science Major

The purpose of this test is not to evaluate you, but to assess the effectiveness of our program. The results will help us to see where program improvements are needed.

Please try your best because these test results are important. The entire CS faculty and staff thanks you!

Section One: Coding Constructs

Derivation

1. true  [false] Because Java does not support multiple inheritance one class cannot have multiple descendent classes as diagrammed below:

```
        ParentClass
          /     \
       /       \
  ChildClass1    ChildClass2
```

2. [true] false Method overloading relates to versions with different parameter lists while method overriding relates to a method in a subclass with the same signature.
3. Which of the following classes definitely extends the **Object** class? Circle the letter of the best answer.

a) class A extends Object
   
   {  
       //details omitted  
   }

b) class B
   
   {  
       //details omitted  
   }

c) class C extends SomeClass
   
   {  
       //details omitted  
   }

d) class A and class B

e) all of the above
Polymorphism

4. It has always been possible for "new code to run old code." This is exemplified by methods such as Math.sqrt(). Polymorphism allows "old code to run new code."

Given the abstract Shape class and the concrete Circle class that extends it directly below:

```java
public abstract class Shape {
    public abstract double getArea();
    public abstract double getPerimeter();
}

public class Circle extends Shape {
    private double radius;

    public Circle(double radius) //constructor
    { this.radius = radius; }

    public double getArea()
    { return Math.PI * (radius*radius); }

    public double getPerimeter()
    { return Math.PI * (2*radius); }
}
```
4.a. Suppose that the Circle class is "old" but that the Square class is a brand new descendent of Shape.

i) Change the statement on line 5 so that a Square object is instantiated with a side length of two:

```java
/* 5 */   Shape shape = new Square(2.0);
```

ii) On line 8 and line 9 make whatever change(s) are needed to get object's area and perimeter:

```java
/* 8 */   no change needed
/* 9 */   no change needed
```

4.b. Write the code for this Square class:

```java
public class Square extends Shape {
    private double sideLength;

    public Square(double sideLength)  //constructor - argument must match 4.a.i.
    {
        this.sideLength = sideLength;
    }

    public double getArea()  //what's important is method's
    {                       //presence and signature
        return sideLength * sideLength;
    }

    public double getPerimeter()  //what's important is method's
    {                           //presence and signature
        return 4 * sideLength;
    }
}
```
5. Pen the changes that allow the throws clause to be removed from the heading of the following method:

```java
String getResponseFromUser() throws IOException
{
    System.out.print(">"); //prompting

    try
    {
        InputStreamReader source = new InputStreamReader(System.in);
        BufferedReader buffer = new BufferedReader(source);
        String lineIn = buffer.readLine();
    }

    catch(IOException e)
    {
    }

    return lineIn;
}
```

6. **true** false An advantage of exception recovery and not propagating the exception is that methods which call this method are relieved of the need to deal with exception handling.

7. **true** false A disadvantage of exception recovery and not propagating the exception is that methods which call this method become unable to take part in recovery operations.
8. Given the following program:

```java
import java.util.StringTokenizer;
import java.util.NoSuchElementException;

class Test1 {
    public static void main(String[] args) {
        String s = "123 456";
        StringTokenizer tokenizer = new StringTokenizer(s);
        int data[] = new data[2]; //array with two elements

        try { //block of code that is to have its exceptions handled
            String token = "";
            int index = 0;

            while (tokenizer.hasMoreTokens()) {
                token = tokenizer.nextToken();
                System.out.print("1 ");

                int numericValue = Integer.parseInt(token);
                System.out.print("2 ");

                data[index] = numericValue;
                System.out.print("3 ");
            }

            System.out.print("4 ");
        }

        catch (NoSuchElementException e) { //StringTokenizers throw this
            System.out.print("5 ");
        }

        catch (ArrayIndexOutOfBoundsException e) { //arrays throw this
            System.out.print("6 ");
        }

        catch (NumberFormatException e) { //Integer.parseInt throws this
            System.out.print("7 ");
        }

        System.out.println("8 ");
    }
}
```

Run as shown, the program's output is:

```
1 2 3 1 2 3 4 8
```
8.a. What is the output when the `String` literal on line 8 is changed to "123 4a6 789"?

123178

8.b. What is the output when the `String` literal on line 8 is changed to "123 456 789"?

1231231268

8.c. What is the output when the `String` literal on line 8 is changed to "123 456 7a9"?

123123178
9. **true**  false  The following is a legal declaration of an interface:

   ```java
   public interface AbilityA
   {
       public void abilityA();
   }
   ```

10. **true**  false  The following class declares an array of type `AbilityA` and a method for placing objects into this array. The name of an interface, just like the name of a class, can serve as a type specifier. Thus, the code below is perfectly legal.

    ```java
    class Item10
    {
        private AbilityA[] a = new AbilityA[10];
        private int cursor = -1;

        void register (AbilityA obj)
        {
            if ( cursor < (a.length - 1) )
            {
                cursor = cursor + 1;
                a[cursor] = obj;
            }
        }

        //details omitted
    }
    ```

11. Which class heading is correct for a class named `Abc` that extends a class named `Parent` and implements the `AbilityA` interface?

    a) public class Abc implements AbilityA extends Parent
    
    b) public class Abc extends AbilityA implements Parent
    
    c) public class Abc implements Parent extends AbilityA
    
    d) public class Abc extends Parent implements AbilityA
12. true false The following method is in the Item10 class on the previous page but was one of those not shown. This callThem() method is perfectly legal.

```java
void callThem()
{
    for (int i = 0; i <= cursor; i++)
    {
        a[i].abilityA();
    }
}
```

13. The construction illustrated in items 9 - 12 is reminiscent of what construction in Java?

Listeners – the arrangement made when instances of a class register with controls (e.g. JButton) for notification of events
Section Two: Complexity Analysis

1. true (false) The analysis of a sorting algorithm relates to the computing resources (time) it requires as the size of the file increases.

2. true (false) The three chief resources subject to analysis are time, space, and programming difficulty.

3. true (false) Trade-offs between good performance (time complexity) and memory requirements (space complexity) are not unusual when choosing one algorithm over another.

4. true (false) It is possible for an "N-squared" algorithm to execute faster than an "N-1g-N" for the same application (e.g. the selection sort in contrast to the quicksort) for small values of N.

5. true (false) When a problem is termed "hard" by a computer scientist, it is because the solution depends on an algorithm with an exponential time complexity. It is not because of the difficult, or time it takes, to write and test the code.

6. true (false) The execution time of an algorithm in Java can be timed down to the microsecond with System.currentTimeMillis() that returns a long. The technique is to get the time immediately before and immediately after, and then subtract the "before" time from the "after" time.

7. true (false) If a 2ⁿ algorithm takes 50 milliseconds to complete when N is 20, it will take on the order of 100 milliseconds to complete when N is 40.
8. Use big-oh notation to express the time complexity of the fragments of code that are shown or described below:

a) $O(n)$  
   
   for (int i = 0; i < n; i++)
   {
      // details omitted
   }

b) $O(n^2)$  
   
   for (int i = 0; i < n; i++)
   {
      for (int j = 0; j < n; j++)
      {
         // details omitted
      }
   }

c) $O(n^2)$  
   
   for (int i = 0; i < n; i++)
   {
      for (int j = i+1; j < n; j++)
      {
         // details omitted
      }
   }

d) $O(N)$  
   a linear search for targets known to be present through an sorted array of length N that exits with a hit

e) $O(\ln N)$  
   a binary search for targets known to be present through a sorted array of length N that exits with a hit

f) $O(\ln N)$  
   processing a look-up path from root to leaf in a height balanced binary search tree containing N nodes

g) $O(c)$  
   look-up in a hash table with a load factor of 20% (which is low)
Section Three: Data Structures with the Java Collections Framework

The API for the `java.util.Stack` class includes the following public instance methods:

```java
public boolean empty();
public Object pop();
public Object push(Object obj);
```

1. What is the value of the postfix expression below?

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

Questions 2-5 pertain to implementing a stack. Mark an item false if any detail is untrue or inaccurate.

2. **true** false
   Suppose you are building a program to evaluate postfix expressions such as the one above. Java's `Stack` class pushes and pops `Objects`, but your program needs to work with numeric primitives. Despite this, a `java.util.Stack` object may be harnessed to serve as the LIFO storage structure for operands.

3. true **false**
   The *breadth-first traversal* of a graph is another application that requires a stack. However the stack is ordinarily implicit as most coders opt for a recursive implementation.

4. **true** false
   There is no logical limit to the number of `Objects` that can be pushed onto a `java.util.Stack` object.

5. **true** false
   An object of type `java.util.Stack` may be used whenever an application program requires a stack. The limitation is that any single method may contain only one `Stack` object at a time.
The API for the `java.util.Vector` class includes the following `public` instance methods:

```java
double get(int index); //returns a reference to the entity at index
double set(int index, double entity); //replaces existing entity at index with the argument entity
double remove(int index); //element at index is removed from structure
```
// if index is less than or equal to the size of the queue,
// the argument object is inserted into the queue at the
// indicated index; otherwise the object is appended to the rear
public void insert(String obj, int index)
{
    if (index < v.size()) v.add(index, obj);
    else v.add(obj);
}

// removes the object at the head of the queue and
// returns a reference to it; but if the queue is empty,
// the method does nothing and returns null
public Object removeFront()
{
    if (v.size() > 0) return v.remove(0);
    else return null;
}

// removes the object stored at the given index and
// returns a reference to it; but if index is greater than
// or equal to the size of the queue, the method does nothing
// and returns null
public Object delete(int index)
{
    if (index >= v.size()) return null;
    else return v.remove(index);

    if (v.size() <= index) return null;
    else return v.remove(index);

    if (index < v.size()) return v.remove(index);
    else return null;

    if (v.size() > index) return v.remove(index);
    else return null;
}
} // closing the class
7. Write the code that uses an instance of the **PriorityQueue** class to perform the processing designated by the comments in the program below:

class Main
{
  public static void main(String[] args)
  {
    //get a PriorityQueue object
    PriorityQueue priorityQueue = new PriorityQueue();

    //addend the String literal "three"
    priorityQueue.append("three");

    //insert the String literal "four" into index 1
    priorityQueue.insert("four", 1);

    //insert the String literal "zero" into index 1
    priorityQueue.insert("zero", 1);

    //append the String literal "two"
    priorityQueue.append("two");

    //remove the String at the head of the list and
    //append it to the rear of the list
    priorityQueue.append( priorityQueue.removeFront() );

    //replace the entity at index 1 with the String literal "one"
    priorityQueue.delete(1);
    priorityQueue.insert("one", 1);

    //write a for loop to display the Strings in the queue,
    //one per line, from head to rear
    for (int i = 0; i < priorityQueue.size(); i++)
    {
      Object holder = priorityQueue.removeFront();
      System.out.println( holder );
      priorityQueue.append(holder);
    }
  }
}
Section Four: Applied Sorting -- Sorting with:

the `java.util.Arrays.sort()` method
the `java.lang.Comparable` interface
`java.util.Comparator` objects

1. An array is filled with random integers in the range of 0..999:

   ```java
   int[] data = new data[10000]; //ten thousand element array
   for (int i = 0; i < data.length; i++)
   {
     data[i] = (int) (Math.random() * 1000); // 0..999
   }
   ```

   Assuming that the `java.util.Arrays` class has been imported, which of the following sorts the values in the `data` array??

   a) `data.sort();`
   b) `data = data.sort();`
   c) `data = data.Arrays.sort();`
   d) `Arrays.sort(data);`
   e) `data = Arrays.sort(data);`

2. An array of objects of any kind can be sorted in the same way as an int array, provided the class from which the objects were instantiated implements the java.lang.Comparable interface:

```java
interface Comparable
{
    int compareTo(Object object);
}
```

Recall that with an invocation: a.compareTo(b)

- 1 is returned when a < b
- 0 is returned when a == b
- 1 is returned when a > b

Show the modifications needed by the class below so that arrays of Square objects may be sorted by sideLength with java.util.Arrays.sort().

```java
class Square implements Comparable
{
    public int compareTo(Object obj)
    {
        Square arg = (Square) obj;
        if (this.sideLength < arg.sideLength) return -1;
        else
            if (this.sideLength == arg.sideLength) return 0;
            else return 1;
    }

    private int sideLength;

    public Square(int forSideLength)
    {
        sideLength = forSideLength;
    }

    public String toString()
    {
        return " " + sideLength + " ";
    }

    public static void showArray(Square[] a)
    {
        for (int i = 0; i < a.length; i++)
        {
            if (i % 10 == 0) System.out.println();
            System.out.print( a[i] );
        }
    }
}
```
3. Complete the following program to display the sideLengths of the Squares in array s before and after sorting.

```java
import java.util.Arrays;

class Main1 {
    public static void main(String[] args) {
        Square[] s = new Square[8]; //Square class implements Comparable
        s[0] = new Square(25);  s[1] = new Square(12);  s[2] = new Square(16);
        s[3] = new Square(14);  s[4] = new Square(11);  s[5] = new Square(23);
        s[6] = new Square(25);  s[7] = new Square(21);

        // Your code to display the array, sort it using Arrays.sort(),
        // and display it again goes here.
        Square.showArray(s);
        Arrays.sort(s);
        Square.showArray(s);
    }
}
```
4. Another approach to sorting arrays of objects is by way of a java.util.Comparator object and the overloaded version of java.util.Arrays.sort() that accepts the array as its first argument and the Comparator object its second.

An object of type java.util.Comparator is an instance of a class that implements the java.util.Comparator interface, shown below:

```
interface java.util.Comparator
{
    int compare(Object objA, Object objB);
}
```

Recall that compare(a, b) returns:

-1 when a < b
0 when a == b
1 when a > b

Given the Rectangle class shown on the following page. Write a class named CompareByArea that implements the Comparator interface and enables arrays of Rectangles to be sorted by their areas.

class CompareByArea implements java.util.Comparator
{
    public int compare(Object left, Object right)
    {
        Rectangle leftR = (Rectangle) left;
        Rectangle rightR = (Rectangle) right;

        if(leftR.getArea() < rightR.getArea()) return -1;
        else if (leftR.getArea() == rightR.getArea()) return 0;
        else return 1;
    }
}
class Rectangle
{
    private int length, height;

    public Rectangle(int length, int height)
    {
        this.length = length;
        this.height = height;
    }

    public int getLength()
    {
        return length;
    }

    public int getArea()
    {
        return height * length;
    }

    public String toString()
    {
        //details omitted
    }

    public static void showArray(Rectangle[] a)
    {
        //details omitted
    }
}
5. Imagine that in addition to the `CompareByArea` comparator you just built, a `CompareByLength` comparator is available. Insert the statements in the program below that will sort the array by the objects' lengths and display it.

```java
import java.util.Comparator;

class Main2
{
    public static void main(String[] args)
    {
        Rectangle[] r = new Rectangle[5];
        r[0] = new Rectangle(1, 6);
        r[1] = new Rectangle(6, 1);
        r[2] = new Rectangle(3, 2);
        r[3] = new Rectangle(2, 4);
        r[4] = new Rectangle(5, 5);

        //sort the array by area and display it
        Comparator comparator = new CompareByArea();
        Arrays.sort(r, comparator);
        r.showArray();

        //sort the array by length and display it
        Comparator lengthComparator = new CompareByLength();
        Arrays.sort(r, lengthComparator);
        r.showArray();
    }
}
```

6. **false** A trap to sorting is that if a class implements the `Comparable` interface, its objects cannot be sorted with `Comparators`. These features are incompatible.

7. **true** A trap to sorting is that `Vectors` (as well as `ArrayLists` and `LinkedLists`) cannot be sorted with `java.util.Arrays.sort()`. However, they can be sorted in exactly the same ways with `java.util.Collections.sort()`. 

21
Section Five: Trees

This section looks at a number of characteristics and a number of operations on a number of different kinds of trees used for different purposes and in different applications.

Binary Search Trees and Information Storage and Retrieval

Given the trees in Figures 1, 2, and 3:

Figure 1

Figure 2
1. Show the in-order traversal of
   1.a. the tree in Figure 1:
     \[ 3 \quad 4 \quad 2 \quad 6 \quad 0 \quad 2 \quad 1 \quad 9 \quad 5 \quad 8 \quad 2 \]
   1.b. the tree in Figure 2:
     \[ 4 \quad 7 \quad 9 \quad 11 \quad 16 \quad 19 \quad 20 \quad 21 \quad 25 \quad 28 \quad 32 \quad 34 \quad 36 \quad 38 \quad 43 \]
   1.c. the tree in Figure 3:
     \[ 6 \quad 11 \quad 17 \quad 19 \quad 20 \quad 21 \quad 28 \quad 34 \quad 40 \quad 43 \]

2. What can be said about the tree in Figure 1?
   a. It qualifies as a heap.
   b. It qualifies as a binary search tree.
   c. It qualifies as a height balanced binary search tree.
   d. None of the above.
   e. All of the above.
3. If a full tree has 3 levels and 7 nodes, how many nodes may be stored on level 4?
   a. It depends on whether the root is considered level 0 or level 1.
   b. 8, because if a full tree has N nodes, the next level always has space for N+1 nodes.
   c. 14, because each leaf may have up to two children
   d. It depends upon the algorithm used to keep the tree height balanced.

4. When a binary tree is full, what is the approximate length of the path from the root to a leaf?
   a. If the tree contains approximately N nodes, the path has a length of around N also.
   b. If the tree contains approximately N nodes, the path has a length of around \( N^2 \) nodes.
   c. If the tree contains approximately N nodes, the path has a length of around \( \log(N) \) nodes, where \( \log(N) \) stands for the log of N to the base 2.
   d. If the tree contains approximately N nodes, the path has a length of around \( \ln(N) \) nodes, where \( \ln(N) \) stands for the natural log of N (the log of N to the base e).
   e. If the tree contains approximately N nodes, the path has a length of around \( \log(N) \) nodes, where \( \log(N) \) stands for the log of N to the base 10.
5. Presume that the tree in Figure 3 is being maintained as a binary search tree. Sketch the tree following the insertion of the keys 10 and 45:
6. The tree in Figure 2 is being maintained as a binary search tree.

6.a. Sketch the tree following the deletion of the node holding 34.

6.b. How many references in the given data structure needed to be changed to accomplish the sketched deletion?

6.c. Looking back at the tree in Figure 2, sketch a different but equally acceptable outcome from the deletion of the same node.
7. What can be said about height balanced binary search trees in contrast to binary search trees that are not height balanced?
   a. Height balanced binary search trees give a guaranteed look-up performance, but the search algorithm is more elaborate and more difficult to program.
   b. Height balanced binary search trees can be searched with the same algorithm that is used to search a binary search tree that is not kept height balanced.
   c. Height balanced binary search trees give a guaranteed look-up performance, but their worst case performance for insertion and deletion can be poor.
   d. When height balanced binary search trees put into production, deletions are marked but not made "in real time" because the tree has to be re-created from scratch.

8. true false
   Generally speaking, hashing gives better average case look-up performance than height balanced binary search trees.

9. true false
   Generally speaking, the binary search gives better performance than height balanced binary search trees, especially in applications where there are numerous insertions and deletions along with the look-ups.

10. What can be said about this seemingly wild historical assertion?
    "While the first binary search was published in 1946, the first published binary search without bugs did not appear until 1962."

    It is:
    a) certainly true (and I'm proud of our professional heritage; what's not to be proud of?!)  
    b) certainly true (and indicative of why I fit into the field so nicely.)  
    c) certainly true (and remind me to find out if they were working on this full time?)  
    d) All I know is that I never got my binary search debugged, but all I had was one semester.
Huffman Encoding

Given the Huffman tree in Figure 4:

![Huffman Tree Diagram]

**Figure 4**

11. Which of the following characters occurred most frequently in the text that served as the basis for constructing the Huffman tree in Figure 4?
   a. G
   b. I
   c. H
   d. !

12. Decode the following bit string using the tree in Figure 4:

   1 0 0 1 1 1 1 1 0 0 1 0 1 0 1 1

   G   O   T   ′′   I   T
Trees as Graphs

Given the tree in Figure 5. The circles represent vertices, and the numbers within a circle is its vertex number.

![Figure 5]

13. Thinking of the tree in Figure 5 as a graph, represent it in an adjacency matrix. Label the rows and columns. Show only the 1s.

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td></td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
14. Show a depth-first traversal starting at vertex 5:

5 3 2 1 7 6 4

15. Show a depth-first traversal starting at vertex 7:

7 1 2 3 4 5 6

16. Show a breadth-first traversal starting at vertex 1:

1 2 7 3 6 4 5
Recursion Trees

17. Recursion trees are not data structures built in memory. They are sketches to depict the sequence of invocations of a method that contains two or more recursive calls.

The superFibonacci series is the series of integers that begins with three 1s. Each successive term thereafter is the sum of the three previous terms. The series looks like this:

\[1\ 1\ 1\ 3\ 5\ 9\ 17\ 31\ 57\ \ldots\]

The following method computes the sum of the number of terms in the superFibonacci series entered as the argument. The call \texttt{superFibonacci(6)} returns 20.

\begin{verbatim}
static int superFibonacci(int numberOfTerms)
{
    int sum = 0;

    if (numberOfTerms == 1)
    {
        sum = 1;
    }
    else if (numberOfTerms == 2)
    {
        sum = 2;
    }
    else if (numberOfTerms == 3)
    {
        sum = 3;
    }
    else
    {
        sum = superFibonacci(numberOfTerms - 1)
            + superFibonacci(numberOfTerms - 2)
            + superFibonacci(numberOfTerms - 3);
    }

    return sum;
}
\end{verbatim}

Draw the recursion tree depicting the series of invocations when \texttt{superFibonacci()} is called with 6 as the actual argument. (The next page is provided for your diagram.)
Each oval represents an activation of \texttt{superFibonacci()}. The number in each oval is the value passed to the formal parameter, \texttt{numberOfTerms}. The value on top of each oval is the return. Activations and returns are made in the order of a depth-first traversal.
Section Six: Architecture

The following question asks you to use instructions from the conceptual assembly language illustrated below:

<table>
<thead>
<tr>
<th>instruction number</th>
<th>operation</th>
<th>operand 1</th>
<th>operand 2</th>
<th>operand 3</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>000</td>
<td>assign</td>
<td>x</td>
<td>unused</td>
<td>y</td>
<td>x = y</td>
</tr>
<tr>
<td>001</td>
<td>add</td>
<td>a</td>
<td>b</td>
<td>sum</td>
<td>sum = a + b</td>
</tr>
</tbody>
</table>
| 010                | equals?            | i         | j         | booleanResult | if (i == j)  
|                    |                    |           |           |          |     booleanResult = true  
|                    |                    |           |           |          |     else booleanResult = false               |
| 011                | branch_on_true     | booleanValue | unused | destination_Label | if (booleanValue == true)  
|                    |                    |           |           |          |     branch to the label instruction with destination_Label in operand 1 |
| 100                | branch_on_false    | booleanVar | unused    | label_123 | if (booleanVar == false)  
|                    |                    |           |           |          |     branch to the label instruction with label_123 in operand 1 |
| 101                | branch_to          | unused    | unused    | label_String | an unconditional branch to the label instruction with label_String in Operand 1 |
| 110                | label              | nameOfLabel | unused    | unused    | a "marker" in the code for branching purposes; no operation performed |
| 111                | display            | x         | unused    | unused    | value of x is displayed on the screen        |
1. Write the assembler code that displays the sum of the integers from 1 to 10 as depicted by the Java code and the flowchart:

```java
sum = 0;
count = 0;
while (count < 11)
{
    count = count + 1;
    sum = sum + count;
}
System.out.println(sum);
```

![Flowchart Diagram]

<table>
<thead>
<tr>
<th>operation</th>
<th>operand 1</th>
<th>operand 2</th>
<th>operand 3</th>
</tr>
</thead>
<tbody>
<tr>
<td>assign</td>
<td>sum</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>assign</td>
<td>count</td>
<td></td>
<td>0</td>
</tr>
<tr>
<td>label</td>
<td>loopTop</td>
<td></td>
<td></td>
</tr>
<tr>
<td>equals?</td>
<td>count</td>
<td>11</td>
<td>testResult</td>
</tr>
<tr>
<td>branch_on_true</td>
<td>testResult</td>
<td></td>
<td>displaySum</td>
</tr>
<tr>
<td>add</td>
<td>count</td>
<td>1</td>
<td>count</td>
</tr>
<tr>
<td>add</td>
<td>count</td>
<td>sum</td>
<td>sum</td>
</tr>
<tr>
<td>branch_to</td>
<td>count</td>
<td>sum</td>
<td>loopTop</td>
</tr>
<tr>
<td>label</td>
<td>displaySum</td>
<td></td>
<td></td>
</tr>
<tr>
<td>display</td>
<td>sum</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

34
2. Use only four places for the binary representations that will accomplish the following subtraction in a CPU that "subtracts by adding":

\[
\begin{array}{c}
6 \\
-2
\end{array}
\] (minuend) (subtrahend)

2.a. Express the minuend in binary.

\[
\begin{array}{cccc}
0 & 1 & 1 & 0
\end{array}
\]

2.b. Express the subtrahend in binary (the number without the sign).

\[
\begin{array}{cccc}
0 & 0 & 1 & 0
\end{array}
\]

2.c. Express the one's complement of the subtrahend.

\[
\begin{array}{cccc}
1 & 1 & 0 & 1
\end{array}
\]

2.d. Express the two's complement of the subtrahend.

\[
\begin{array}{cccc}
1 & 1 & 1 & 0
\end{array}
\]

2.e. Evaluate the sum of the minuend plus the two's complement of the subtrahend, discarding overflow.

\[
\begin{array}{cccc|cccc}
& & & & & 1 & 1 & 1 \\
& & & & & 0 & 1 & 1 & 0 \\
\hline
\text{two's complement of subtrahend:} & + & & & & 1 & 1 & 1 & 0 \\
& & & & & 1 & 0 & 1 & 0 & 0
\end{array}
\]

3. true [false] The largest integer that can be represented in the unsigned binary system in 4 bits is 16.

4. true [false] The logical "OR" operation is used to clear bits.

5. true false The "Fetch-Execute Cycle" describes the operation of the CPU.
Questions 6 - 10 pertain to the circuit shown below:

6. What is the output when: \[ \begin{cases} 
\text{bit 1} = 0 \quad \text{(false)} \\
\text{bit 2} = 0 \quad \text{(false)} \\
\text{bit 3} = 0 \quad \text{(false)}
\end{cases} \]?

[Answer: circuit 1]

7. What is the output when: \[ \begin{cases} 
\text{bit 1} = 0 \quad \text{(false)} \\
\text{bit 2} = 1 \quad \text{(true)} \\
\text{bit 3} = 0 \quad \text{(false)}
\end{cases} \]?

[Answer: circuit 2]
8. How many distinct inputs are possible? \[ \text{three bits worth} = 2^3 = 8 \]

9. What kind of circuit is this?
   a. adder
   b. half-adder
   c. instruction decoder
   d. a variant flip-flop
   e. none of the above

10. \textcolor{red}{true} false This circuit excites philosophers because it enables symbolic representations to bring forth action without an external agent to interpret them (Hume's problem).
1. Suppose that two "producer" threads are appending objects onto the end of a FIFO queue, and that a "consumer" thread is taking these objects from the front of the queue for processing.

Suppose the queue is implemented in an array. \texttt{tail} is an \texttt{int} storing the subscript of the element to receive the next object from a producer. The heart of the append action, in bold below, is to store the incoming object and increment the queue's cursor:

\begin{verbatim}
queue[tail] = incomingObject;
tail = tail + 1;
\end{verbatim}

Identify a mutual exclusion problem here relative to the two producer threads.

\textbf{producer1} is interrupted immediately after executing the first statement; that is, after storing its object in the array but before incrementing the cursor.

\textbf{producer2} proceeds to store its object, destructively assigning it to the same compartment. The result is that the object stored by \textbf{producer1} is lost.

2. \textbf{true} \hspace{1cm} \textbf{false} To solve the mutual exclusion problem (sometimes known as a "race condition"), the producers and the consumer may put a "lock" on the queue. If a producer's time slice expires when it holds the lock on the queue, the other producer and the consumer will be blocked (i.e. denied access).
3. **true** false  In Java, object locking is accomplished with **synchronized** methods.

4. **true** false  Besides solving the mutual exclusion problem, synchronization eliminates the issue of deadlock.
1. Which of the following strings is not in the language defined by the following deterministic finite automaton?

![](image)

a) a b b
b) b a b b
c) b a a b a

d) a b b a b b

2. Remove the left recursion from the following grammar:

```
S ::= S "a" A | "b" B
A ::= "a" B | "c"
B ::= B "b" | "d"
```

```
S ::= ("b" B) {"a" A}
A ::= "a" B | "c"
B ::= "d" {"b"}
```
Section Nine: Computer Networking and the Internet

1. A subnet has the address 223.1.1.0/25 (the /25 means that 25 bits are used for the network part of the address). How many hosts can the subnet include?

With 25 bits used for the network's part of the 32 bit total, there are seven bits left for the host's part. With seven bits, there are a total of \(2^7 = 128\) different address; however, the all 0s and the all 1s have special meaning. This allows for \(128 - 2 = 126\) different hosts.

2. In the following network, the host with IP address 111.111.111.111 is sending a datagram to the host with IP address 222.222.222.222. Show the sender and receiver MAC addresses at layer 2 for every link the datagram travels.

```
link 1: host with IP address: 111.111.111.111 to router with IP address: 111.111.111.110
MAC address of sender (the host): 74-29-9C-EB-FF-55
MAC address of receiver (the router): E6-E9-00-17-BB-4B

link 2: router with IP address: 222.222.222.220 to host with IP address: 222.222.222.222
MAC address of sender (the router): 1A-23-F9-CD-06-9B
MAC address of receiver (the host): 49-BD-D2-C7-56-2A
```
Section Ten: Mathematics

1. Imagine the following problem:

Each of five cities is connect to each of the others with a direct highway. How many highways are there?

The numerical answer as well as the general formula can be derived by inspection from the adjacency matrix.

Sketch a matrix, draw a loop around "the location of the answer," and give the general formula.

The cities are named A, B, C, D, and E.

The highways are undirected edges, which means that the same highway that connects city i to city j connects city j to city i.

The result is that each highway appears twice in the adjacency matrix, above the main diagonal and in the mirror-image position below.

No entries are on the main diagonal because no highway connects a city to itself.

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>C</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>D</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

These 10 entries represent each highway once and only once.

The adjacency matrix has $5 \times 5 = 25$ elements. Subtracting the main diagonal, which consists of one element on each row, leaves $25 - 5 = 20$ elements. Because each highway is represented twice, there is space for $20/2 = 10$ highways. The isomorphism between the adjacency matrix and physical geography indicates that 10 highways are needed to connect each of the five cities directly with each of the others. In general:

\[
\text{numOfHighways} = \frac{(\text{numOfCities}^2 - \text{numOfCities})}{2}
\]
2.a. Math.random() supplies doubles drawn pseudo-randomly from the standard uniform probability density distribution.

Complete the method for returning uniformly distributed pseudo-random integers within the range of 1 to 10 inclusive.

```java
public class Randoms {
    public static int OneToTen() {
        double standardRandom = Math.random();
        // Your code goes here
        double enlargedRange = standardRandom * 10;  // 0..10, non-inclusive
        int tenDiscreteValues = (int) enlargedRange;  // 0..9, inclusive
        int r = tenDiscreteValues + 1;  // 1..10, inclusive
        return r;
    }
    /* or, as a one-liner:
    return ((int) (standardRandom * 10)) + 1;
    */
}
```

2.b. Which comment below best describes what is wrong with the following method built to return uniformly distributed pseudo-random integers within the range of 1 to 10 inclusive?

```java
public int pick_A_Number_Between_One_And_Ten()
{
    double accumulator = 0;
    for (int i = 1; i <= 10; i++) // iterates ten times
    {
        accumulator += Math.random();
    }
    return (int) accumulator + 1;
}
```

a. Slow performance: ten different values from Math.random() are needed for each generated random integer

b. Incorrect performance: the generated random integers are not uniformly distributed; there are many more 5s and 6s than 1s, 2s, 8s, and 9s

c. Syntax error in the line accumulator += Math.random();

d. Semantic error in the line (int) accumulator + 1; due to the precedence of casting
<table>
<thead>
<tr>
<th>Item Number</th>
<th>Content - General Classification</th>
<th>Content - More Detail</th>
</tr>
</thead>
<tbody>
<tr>
<td>Section 1, Number 1</td>
<td>derivation diagram</td>
<td>multiple inheritance</td>
</tr>
<tr>
<td>Section 1, Number 2</td>
<td>derivation</td>
<td>method overloading versus overriding</td>
</tr>
<tr>
<td>Section 1, Number 3</td>
<td>derivation</td>
<td>implicit extension of the Object class</td>
</tr>
<tr>
<td>Section 1, Number 4a</td>
<td>polymorphism</td>
<td>coding a polymorphic application</td>
</tr>
<tr>
<td>Section 1, Number 4b</td>
<td>polymorphic subclassing</td>
<td>coding a full class (polymorphic subclass)</td>
</tr>
<tr>
<td>Section 1, Number 5</td>
<td>exceptions</td>
<td>coding the try-catch clause, the throws clause</td>
</tr>
<tr>
<td>Section 1, Number 6</td>
<td>exceptions</td>
<td>properties of exception propagation</td>
</tr>
<tr>
<td>Section 1, Number 7</td>
<td>exceptions</td>
<td>properties of exception propagation</td>
</tr>
<tr>
<td>Section 1, Number 8a,b,c</td>
<td>exceptions</td>
<td>code reading - flow of control with an exception</td>
</tr>
<tr>
<td>Section 1, Number 9</td>
<td>interfaces</td>
<td>declaring an interface</td>
</tr>
<tr>
<td>Section 1, Number 10</td>
<td>interfaces</td>
<td>interface a data type specifier to declare a reference</td>
</tr>
<tr>
<td>Section 1, Number 11</td>
<td>interfaces</td>
<td>declaring a class that implements an interface</td>
</tr>
<tr>
<td>Section 1, Number 12</td>
<td>arrays</td>
<td>calling a method on an object in an array</td>
</tr>
<tr>
<td>Section 1, Number 13</td>
<td>parsers</td>
<td>listeners - that they work as callbacks</td>
</tr>
<tr>
<td>Section 2, Number 1</td>
<td>complexity analysis</td>
<td>complexity - what it means</td>
</tr>
<tr>
<td>Section 2, Number 2</td>
<td>complexity analysis</td>
<td>complexity - the computing resources it relates to</td>
</tr>
<tr>
<td>Section 2, Number 3</td>
<td>complexity analysis</td>
<td>complexity - the performance-space trade-off</td>
</tr>
<tr>
<td>Section 2, Number 4</td>
<td>complexity analysis</td>
<td>complexity - interpretation</td>
</tr>
<tr>
<td>Section 2, Number 5</td>
<td>complexity analysis</td>
<td>complexity - hard problems</td>
</tr>
<tr>
<td>Section 2, Number 6</td>
<td>complexity analysis</td>
<td>coding - measuring execution time</td>
</tr>
<tr>
<td>Section 2, Number 7</td>
<td>complexity analysis</td>
<td>complexity - interpretation of exponential big oh</td>
</tr>
<tr>
<td>Section 2, Number 8a,b,c</td>
<td>complexity analysis</td>
<td>complexity - big oh of segments of code</td>
</tr>
<tr>
<td>Section 2, Number 8d</td>
<td>complexity analysis</td>
<td>complexity - big oh of the linear search look-up</td>
</tr>
<tr>
<td>Section 2, Number 8g</td>
<td>complexity analysis</td>
<td>complexity - big oh of the binary tree search look-up</td>
</tr>
<tr>
<td>Section 2, Number 8f</td>
<td>complexity analysis</td>
<td>complexity - big oh of binary search tree look-up</td>
</tr>
<tr>
<td>Section 2, Number 8g</td>
<td>complexity analysis</td>
<td>complexity - big oh of hash table look-up</td>
</tr>
<tr>
<td>Section 3, Number 1</td>
<td>architecture</td>
<td>postfix expression evaluation</td>
</tr>
<tr>
<td>Section 3, Number 2</td>
<td>data structures</td>
<td>storing in a collection (list to Object)</td>
</tr>
<tr>
<td>Section 3, Number 3</td>
<td>data structures</td>
<td>breadth-first traversal</td>
</tr>
<tr>
<td>Section 3, Number 4</td>
<td>data structures</td>
<td>Java collections framework - using java.util.Stack</td>
</tr>
<tr>
<td>Section 3, Number 5</td>
<td>data structures</td>
<td>Java collections framework - using java.util.Stack</td>
</tr>
<tr>
<td>Section 3, Number 6</td>
<td>coding - adapter pattern</td>
<td>methods - argument passing</td>
</tr>
<tr>
<td>Section 3, Number 7</td>
<td>coding - using an API</td>
<td>methods - using an API</td>
</tr>
<tr>
<td>Section 4, Number 1</td>
<td>coding - using Arrays.sort()</td>
<td>sorting an int array with Arrays.sort()</td>
</tr>
<tr>
<td>Section 4, Number 2</td>
<td>coding - using Comparable</td>
<td>Comparable interface - a class that implements</td>
</tr>
<tr>
<td>Section 4, Number 3</td>
<td>coding - using Arrays.sort()</td>
<td>Comparable interface - sorting objects that implement</td>
</tr>
<tr>
<td>Section 4, Number 4</td>
<td>coding - class to be a Comparator</td>
<td>Comparator interface - a class that implements</td>
</tr>
<tr>
<td>Section 4, Number 5</td>
<td>coding - using a Comparator</td>
<td>Comparator - sorting with a Comparator object</td>
</tr>
<tr>
<td>Section 4, Number 6</td>
<td>coding - Comparable and Comparator</td>
<td>Comparable interface and Comparator objects for same class</td>
</tr>
<tr>
<td>Section 4, Number 7</td>
<td>coding - using Collections.sort()</td>
<td>sorting a Vector with Collections.sort()</td>
</tr>
<tr>
<td>Section 5, Number 1</td>
<td>data structures</td>
<td>inorder traversal of binary trees - performing</td>
</tr>
<tr>
<td>Section 5, Number 2</td>
<td>data structures</td>
<td>heap</td>
</tr>
<tr>
<td>Section 5, Number 3</td>
<td>data structures</td>
<td>binary trees - number of nodes on next level of a full tree</td>
</tr>
<tr>
<td>Section 5, Number 4</td>
<td>data structures</td>
<td>binary trees - length of path from root to leaf in a full tree</td>
</tr>
<tr>
<td>Section 5, Number 5</td>
<td>data structures</td>
<td>binary search tree - performing insertions</td>
</tr>
<tr>
<td>Section 5, Number 6</td>
<td>data structures</td>
<td>binary search tree - performing deletions</td>
</tr>
<tr>
<td>Section 5, Number 7</td>
<td>data structures</td>
<td>binary search trees - not height balanced versus height balanced</td>
</tr>
<tr>
<td>Section 5, Number 8</td>
<td>data structures</td>
<td>binary search trees - height-balanced trees versus hashing</td>
</tr>
<tr>
<td>Section 5, Number 9</td>
<td>data structures</td>
<td>binary search trees - height-balanced trees versus binary search</td>
</tr>
<tr>
<td>Section 5, Number 10</td>
<td>data structures</td>
<td>binary search - historical trivia (for fun)</td>
</tr>
<tr>
<td>Section 5, Number 11</td>
<td>data structures</td>
<td>Huffman tree - structure</td>
</tr>
<tr>
<td>Section 5, Number 12</td>
<td>data structures</td>
<td>Huffman encoding of a message</td>
</tr>
<tr>
<td>Section 5, Number 13</td>
<td>data structures</td>
<td>adjacency matrix - using an adjacency matrix to represent a tree</td>
</tr>
<tr>
<td>Section 5, Number 14</td>
<td>data structures</td>
<td>depth-first traversal of a binary tree - performing it</td>
</tr>
<tr>
<td>Section 5, Number 15</td>
<td>data structures</td>
<td>depth-first traversal of a binary tree - performing it</td>
</tr>
<tr>
<td>Section 5, Number 16</td>
<td>data structures</td>
<td>breadth-first traversal of a binary tree - performing it</td>
</tr>
<tr>
<td>Section 5, Number 17</td>
<td>code reading - drawing a recursion tree</td>
<td>recursion - drawing a recursion tree for a method with three recursive calls</td>
</tr>
<tr>
<td>Section 6, Number 1</td>
<td>architecture</td>
<td>assembler - coding a while loop in a conceptual assembly language</td>
</tr>
<tr>
<td>Section 6, Number 2a,b</td>
<td>architecture</td>
<td>binary number system - expressing an integer</td>
</tr>
<tr>
<td>Section 6, Number 2c</td>
<td>architecture</td>
<td>binary number system - expressing a two's complement</td>
</tr>
<tr>
<td>Section 6, Number 2d</td>
<td>architecture</td>
<td>binary number system - expressing a one's complement</td>
</tr>
<tr>
<td>Section 6, Number 3</td>
<td>architecture</td>
<td>binary number system - expressing an integer</td>
</tr>
<tr>
<td>Section 6, Number 4</td>
<td>architecture</td>
<td>logical AND and OR</td>
</tr>
<tr>
<td>Section 6, Number 5</td>
<td>architecture</td>
<td>fetch-execute cycle (the i-fetch)</td>
</tr>
<tr>
<td>Section 6, Number 6</td>
<td>architecture</td>
<td>reading a diagram with AND and NOT gates</td>
</tr>
<tr>
<td>Section 6, Number 7</td>
<td>architecture</td>
<td>reading a diagram with AND and NOT gates</td>
</tr>
<tr>
<td>Section 6, Number 8</td>
<td>architecture</td>
<td>that n bits can encode 2-to-the-n instructions</td>
</tr>
<tr>
<td>Section 6, Number 9</td>
<td>architecture</td>
<td>instruction decoder</td>
</tr>
<tr>
<td>Section 6, Number 10</td>
<td>architecture</td>
<td>instruction decoder</td>
</tr>
<tr>
<td>Section 6, Number 11</td>
<td>operating systems</td>
<td>mutual exclusion problem (producers and consumer)</td>
</tr>
<tr>
<td>Section 7, Number 2</td>
<td>coding - using Threads/synchronization</td>
<td>locking - lock retained when time-slice expires</td>
</tr>
<tr>
<td>Section 7, Number 3</td>
<td>coding - using Threads/synchronization</td>
<td>synchronized methods and locking</td>
</tr>
<tr>
<td>Section 7, Number 4</td>
<td>coding - using Threads/synchronization</td>
<td>synchronized methods - properties</td>
</tr>
<tr>
<td>Section 8, Number 1</td>
<td>languages and compilers</td>
<td>finite state automaton - Strings defined by</td>
</tr>
<tr>
<td>Section 8, Number 2</td>
<td>languages and compilers</td>
<td>grammar - performing the removal of left recursion</td>
</tr>
<tr>
<td>Section 9, Number 1</td>
<td>networks</td>
<td>IP addresses - number of hosts in a subnet</td>
</tr>
<tr>
<td>Section 9, Number 2</td>
<td>networks</td>
<td>MAC addresses of diagram senders and receivers</td>
</tr>
<tr>
<td>Section 10, Number 1</td>
<td>mathematics</td>
<td>graph - number of edges in a non-directed graph with each vertex adjacent to each other</td>
</tr>
<tr>
<td>Section 10, Number 2a,b</td>
<td>mathematics</td>
<td>linear transformation to adjust the standard uniform distribution</td>
</tr>
</tbody>
</table>
The Seidenberg School of Computer Science and Information Systems
Pace University
Technical Report Series

EDITORIAL BOARD

Editor:
Allen Stix, Computer Science, Pace—Westchester

Associate Editors:
Constance A. Knapp, Interim Dean, The Seidenberg School of Computer Science and Information Systems — Pace University
Susan M. Merritt, Computer Science, Pace—Westchester

Members:
Howard S. Blum, Computer Science, Pace—New York
Mary F. Courtney, Computer Science, Pace—Westchester
Nicholas J. De Lillo, Mathematics and Computer Science, Manhattan College
Daniel Farkas, Information Systems, Pace—Westchester
Fred Grossman, Information Systems; Doctor of Professional Studies, Pace—New York and White Plains
Fran Goertzl Gustavson, Information Systems, Pace—Westchester
Joseph F. Malerba, Computer Science, Pace—Westchester
John S. Mallozzi, Computer Information Sciences, Iona College
John C. Molluzzo, Information Systems, Pace—New York
Pauline Mosley, Technology Systems, Pace—New York
Narayan S. Murthy, Computer Science, Pace—New York
Catherine Ricardo, Computer Information Sciences, Iona College
Judith E. Sullivan, CSIS Alumna, MS in CS from Pace—Westchester
Sylvester Tuohy, Computer Science, Pace—Westchester

The Seidenberg School of Computer Science and Information Systems, through the Technical Report Series, provides members of the community an opportunity to disseminate the results of their research by publishing monographs, working papers, and tutorials. Technical Reports is a place where scholarly striving is respected.

All preprints and recent reprints are requested and accepted. New manuscripts are read by two members of the editorial board; the editor decides upon publication. Statements of policy and mission may be found in issues #29 (April 1990) and #34 (September 1990).

Please direct submissions as well as requests for single copies to:

Allen Stix
The Seidenberg School of Computer Science and Information Systems
Goldstein Academic Center
Pace University
861 Bedford Road
Pleasantville, NY 10570-2799