

Programming Problems  
12<sup>th</sup> Annual Computer Science Programming Contest

Department of Mathematics and Computer Science

Western Carolina University

April 24, 2001

## Criteria for Determining Team Scores

Each program should:

1. execute without error and solve the appropriate problem,
2. execute properly using reasonable alternative input values (this includes handling possible error conditions)
3. follow good programming style conventions such as avoiding confusing and unnecessary code,
4. be documented with comments and easily understandable variable names, and
5. start with an initial comment that includes a) the name of the team's school, b) the team's number, and c) the names of the team's members. An example such comment is

*// Somewhere High School, Team 2, Susan Smith, Shelley Jones, and Sandy Hart*

The most important feature is that each program should execute properly. The total accumulated score for all four programs represents the team score. Ties in team scores will be eliminated by re-examining program robustness and style. All the programs have the same weight in the scoring.

### Notes

1. You can not use any reference materials (such as books or the Internet) when doing the problems.
2. You can use more than one computer if you wish.

### **Problem 1 (Token Detector)**

Write a program that will read in a line of text and detect whether the token "!=" exists within that line of text. The program will then output to the console whether or not it found the token in the line of text. The line of text can be arbitrarily long; you cannot assume some maximum length for the line of text.

Your solution must behave as shown in the example runs below. Output is in boldface and input is italicized.

#### **Example Program Run 1**

**Please enter a sequence of characters (terminated by a ';'):**

*sum == MAX;*

**The token does not exist.**

#### **Example Program Run 2**

**Please enter a sequence of characters (terminated by a ';'):**

*sum != MAX;*

**The token does exist.**

#### **Example Program Run 2**

**Please enter a sequence of characters (terminated by a ';'):**

*sum !!= MAX;*

**The token does exist.**

## **Problem 2 (Recursive Image Erasing)**

Write a program that will recursively erase a feature from an image. Suppose that an image is represented by a two-dimensional array of values where each value can either be black or white. Contiguous array elements that are black form a feature. Erasing a feature means to turn all the array elements of the feature to the value white.

Your solution must use a function called `eraseObject` recursively to perform the erasing. The function `eraseObject` is passed the array as an argument and row and column positions as arguments. If the row and column positions specify a position in a feature, that feature is removed by recursively calling `eraseObject`. If the row and column positions specify a position that is not in a feature, the `eraseObject` function just returns.

Your solution must behave as shown in the example runs below. Output is in boldface and input is italicized. Both example runs are using the same input image. The only difference is the initial position specified. In the first run the position is in the feature, so the feature is erased. In the second run the position is not in the feature, so the feature is not erased. The top left corner position is row 0 and column 0. Row numbers increase going downward and column numbers increase going to the right. Thus row = 1 and column = 2 is the B in the center of the feature. Row = 3 and column = 2 is one of the Ws in the last row so there is no erasure.

### **Example Program Run 1**

**The image is represented by a 4 x 4 array of Bs and Ws.**

**Enter the characters for row 0.**

*WWBW*

**Enter the characters for row 1.**

*WBBB*

**Enter the characters for row 2.**

*WWBW*

**Enter the characters for row 3.**

*BWWW*

**Enter the row of the initial erasure position:**

*1*

**Enter the column of the initial erasure position:**

*2*

**The final image is**

*WWWW*

*WWWW*

*WWWW*

*BWWW*

**Example Program Run 2**

**The image is represented by a 4 x 4 array of Bs and Ws.**

**Enter the characters for row 0.**

*WWBW*

**Enter the characters for row 1.**

*WBBB*

**Enter the characters for row 2.**

*WWBW*

**Enter the characters for row 3.**

*BWWW*

**Enter the row of the initial erasure position:**

3

**Enter the column of the initial erasure position:**

2

**The final image is**

*WWBW*

*WBBB*

*WWBW*

*BWWW*

### **Problem 3 (Unsorted Intersection Check)**

Write a program that if given two unsorted arrays of ints determines whether or not either array has an element that is also an element of the other array. Your solution must behave as shown in the example runs below. Output is in boldface and input is italicized.

#### **Example Program Run 1**

**Each array is four integers.**

**Enter the integers for the first array.**

*12 22 15 31*

**Enter the integers for the second array.**

*5 33 22 18*

**The intersection is non-empty.**

#### **Example Program Run 2**

**Each array is four integers.**

**Enter the integers for the first array.**

*12 22 15 31*

**Enter the integers for the second array.**

*5 33 21 18*

The intersection is empty.

#### **Problem 4 (Sorted Intersection Check)**

Write a program that if given two sorted arrays of ints determines whether or not either array has an element that is also an element of the other array. To receive credit for solving this problem, your solution must be significantly more efficient than a solution that would be possible in the unsorted case (which is problem 3). Your solution must behave as shown in the example runs below. Output is in boldface and input is italicized.

#### **Example Program Run 1**

**Each array is four integers.**

**Enter the integers for the first array.**

*12 15 22 31*

**Enter the integers for the second array.**

*5 18 22 33*

**The intersection is non-empty.**

#### **Example Program Run 2**

**Each array is four integers.**

**Enter the integers for the first array.**

*12 15 22 31*

**Enter the integers for the second array.**

*5 18 21 33*

**The intersection is empty.**