

Programming Problems

15th Annual Computer Science Programming Contest

Department of Mathematics and Computer Science

Western Carolina University

March 30, 2004

Criteria for Determining Scores

Each program should:

1. execute without error, solve the appropriate problem efficiently, and satisfy all the conditions specified in the problem statement,
2. follow good programming style conventions such as avoiding confusing and unnecessary code,
3. be documented with comments and easily understandable variable names, and
4. 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, and Sandy Hart
```

The score assigned to each program will be based upon the extent that the program satisfies the properties listed above. The total accumulated score for all four programs represents the team score. All the programs have the same weight in the scoring.

Notes

- Each team may use two reference books appropriate to their language division (e.g. two Java books for a team competing in the Java division).
- Each team may use up to **two** computers.
- **Inappropriate code comments will result in a lower score and is grounds for disqualification for an award.**

Programming notes

- General

- Where appropriate, you should do input verification. In other words, if the user is supposed to input, for example, a positive number, an appropriate error message should be displayed when the user does not enter a positive number. The user should then get additional chances to enter valid input.

- C++

- The AP classes `apmatrix`, `apqueue`, `apstack`, `apstring`, and `apvector` are available for your use. (This is not to say all these are needed for the contest.) They are located at `/home/CONTEST2003/AP_CPP_Classes`

Use the `cp` command to bring them into your home directory. The command below will copy all the AP classes into your home directory, if run from your home directory:

```
cp /home/CONTEST2003/AP_CPP_Classes/* .
```

The last dot on the above line is important. It copies the files into the current directory. The space immediately preceding this last dot is also important. Furthermore, even though this is 2004, the “2003” is not a typo.

- The College Board requires that we include the following disclaimer regarding the ap classes:

“Inclusion of the C++ classes defined for use in the Advanced Placement Program Computer Science courses does not constitute endorsement of the other material in this contest by the College Board, Educational Testing Service, or the AP Computer Science Development Committee. The versions of the C++ classes defined for use in the AP Computer Science courses included in this contest were accurate as of March 20, 2003. Revisions to the classes may have been made since that time.”

- If you wish to use the C++ standard string class, you must include the following statement along with the `#include<string>` statement:

```
using namespace std;
```

- Java

- Console I/O (i.e. keyboard input & screen output) is the expected form of input/output. Use `System.out.println` (or `System.out.print`, as appropriate) for output. The following information may be useful to implement keyboard input:

1. First, make sure that the following “import” statement is included at the very top of the code:

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

2. Before doing any keyboard input, declare a `BufferedReader` object (named “br” in the following example) tied to `System.in`:

```
BufferedReader br = new BufferedReader( new InputStreamReader( System.in ) );
```

3. Use this object to read Strings (one line at a time) from the keyboard; for example:

```
System.out.println("Enter one line of input:");
String s = br.readLine();
System.out.println("Enter another line of input:");
String t = br.readLine();
```

The above fragment would read two lines from the keyboard; `s` references the String that is the first line of input, while `t` references the second line.

4. Use `Integer.parseInt` or `Double.parseDouble` to convert any Strings to ints or doubles, respectively. Reading a line of input and converting it to a numeric type can be done in one statement; for example:

```
int i = Integer.parseInt( br.readLine() );
```

5. Finally, any methods that contain a call to `readLine` (as well as methods that call a method that calls `readLine`, etc.) need a “throws Exception” clause in the method header, unless you want to deal with Exception handling and place the `readLine` call in a `try..catch` block. (This type of Exception handling is not an expectation for the contest, so it’s recommended that you just include the “throws Exception” clause.) For example:

```
public static void main( String[] args ) throws Exception
{
    ... // call to readLine somewhere in here
}
```

- An additional note about console I/O and BlueJ: Be sure to have a `System.out.println` statement that executes before any call to `readLine`. The `System.out.println` call tells BlueJ to open a console window; it's not smart enough to do this if a call is made to `readLine` first.

1 Final Jeopardy

You're contestant #1 on the popular Jeopardy game show, it's time for "Final Jeopardy," and you're trying to decide your wager. Recall that in the Final Jeopardy round, each contestant must wager any amount between \$0 and their current winnings (inclusive), knowing only the Final Jeopardy category. Each contestant wins or loses the dollar amount of their wager if they answer the Final Jeopardy question correctly or incorrectly, respectively.

Not considering the Final Jeopardy category, write a program that accepts as input your current winnings, followed by the current winnings of your two competitors (contestants #2 and #3). The program should determine your final jeopardy wager, so that in the case you win, you do so by only \$1. (Therefore, you don't wager "too much," just enough to win if you can.) For example, if you are currently in the lead with \$9000, while your closest competitor only has \$4000, your wager should be \$999. (Because in the worst-case that your competitor wagers their whole \$4000 and answers correctly, you can answer incorrectly and still win by \$1.) Or, for example, suppose you have the lead with \$8000, while your closest competitor has \$5000; you should wager \$2001. (Because if your competitor wagers their entire \$5000 and answers correctly, you need to answer correctly and have wagered \$2001 to win by \$1.) If, however, you do not currently have the lead, you should wager your entire winnings.

An example session:

```
Welcome to the Final Jeopardy round!  
Enter your current winnings: 8000  
Enter the current winnings of contestant #2: 3000  
Enter the current winnings of contestant #3: 2000  
You can comfortably wager $1999.
```

Another example session:

```
Welcome to the Final Jeopardy round!  
Enter your current winnings: 3000  
Enter the current winnings of contestant #2: 4000  
Enter the current winnings of contestant #3: 4500  
You're not in the lead, and should wager your entire $3000.
```

2 Array Rank

The **rank** of an element in an array of integers is the number of smaller elements in the array plus the number of equal elements that appear to its left. For example, consider the array [4, 3, 9, 3, 7]. The respective ranks of these elements are 2, 0, 4, 1, and 3; thus the **rank array** is the array [2, 0, 4, 1, 3]. Write a program that prompts the user for an arbitrarily-sized array of integers, and then outputs the corresponding rank array.

An example session:

```
Enter desired size of array : 5
Enter the ints, each on a separate line:
4
3
9
3
7
The input array was:
4 3 9 3 7
The corresponding rank array is:
2 0 4 1 3
```

3 Perfect Numbers

A number (consider only positive integers) is **perfect** if it is equal to the sum of its proper divisors. For example, 6 is a perfect number, because its proper divisors are 1, 2, and 3 (note that we do not include the number itself), and $1 + 2 + 3 = 6$.

A number is **deficient** if the sum of its proper divisors is less than the number. For example, 8 is deficient, because its proper divisors are 1, 2, and 4, and $1 + 2 + 4 = 7$, which is less than 8.

A number is **abundant** if the sum of its proper divisors is greater than the number. For example, 12 is abundant, because $1 + 2 + 3 + 4 + 6 = 16$, which is greater than 12.

Write a program that prompts the user for a number, then determines whether the number is perfect, deficient, or abundant. Your program should continue to prompt the user for numbers until a 0 is provided as input.

An example session:

```
Enter an integer (0 to quit): 7
7 is deficient.
```

```
Enter an integer (0 to quit): 12
12 is abundant.
```

```
Enter an integer (0 to quit): 6
6 is perfect.
```

```
Enter an integer (0 to quit): 0
Goodbye!
```

4 Permutations

A **permutation** of an array of integers is an arrangement of the integers in the array. Consider, for example, the array of integers [5, 2, 4]. One possible permutation is the array [2, 5, 4]. Some other permutations are [2, 4, 5], [5, 4, 2], and [5, 2, 4] (the original array itself). For this program, we'll assume that no integer appears more than once in the original array. (Thus, we are not considering arrays such as [3, 4, 4].) Note that for this example of an array of three integers, there are six possible permutations. Write a program that prompts the user for an arbitrarily-sized array of integers, and outputs all the permutations of the input array.

An example session:

```
Enter number of integers in array : 3
Enter the ints, each on a separate line:
1
2
3
The permutations:
1 2 3
1 3 2
2 1 3
2 3 1
3 1 2
3 2 1
```

Another example session:

```
Enter number of integers in array : 4
Enter the ints, each on a separate line:
6
3
2
-5
The permutations:
6 3 2 -5
6 3 -5 2
6 2 3 -5
6 2 -5 3
6 -5 3 2
6 -5 2 3
3 6 2 -5
3 6 -5 2
3 2 6 -5
3 2 -5 6
3 -5 6 2
3 -5 2 6
2 6 3 -5
2 6 -5 3
2 3 6 -5
2 3 -5 6
2 -5 6 3
2 -5 3 6
-5 6 3 2
-5 6 2 3
-5 3 6 2
-5 3 2 6
-5 2 6 3
-5 2 3 6
```

Note: It is not critical that your program output the permutations in any particular order, it just needs to list every permutation exactly once.

5 Tie Breaker: Permutations, Part 2

Note: This program will only be graded if needed to break a tie.

Write a new version of the “Permutations” program that does not assume that elements in the input array are distinct. Again, this program should output each possible permutation exactly once. (Thus, you must take care not to output any duplicate permutations.)

An example session:

```
Enter number of integers in original array : 3
Enter the ints, each on a separate line:
1
1
2
The permutations:
1 1 2
1 2 1
2 1 1
```

Another example session:

```
Enter number of integers in original array : 2
Enter the ints, each on a separate line:
3
4
The permutations:
3 4
4 3
```

And, one more example session:

```
Enter number of integers in original array : 4
Enter the ints, each on a separate line:
4
3
4
3
The permutations:
4 3 4 3
4 3 3 4
4 4 3 3
3 4 4 3
3 4 3 4
3 3 4 4
```