The Notre Dame Programming Contest

October 8th, 2006

Instructions
You will have 2½ hours to complete as many problems as possible. There are a total of seven problems, but you are not expected to solve them all. Instead, you should try to find the easiest problems and solve those first, then tackle the harder ones in any remaining time. Each problem is worth one point, but some problems are much easier than others.

You are free to use any books, printed materials, or any other dead-tree references, but you may not access the Internet during the contest. This contest packet is yours to tear apart, scribble upon, and take home.

If at any time you have a question about using the contest software, if you believe that the electronic judge has made an error, or if you are just plain stuck trying to get some code to compile, ask for help.

Scoring
Whoever has solved the most problems within the allotted time will be the winner. If there is a tie, it will be awarded to whoever has the lowest sum of the total times to complete every solved problem. In other words, it's good to submit correct solutions quickly. Incorrect submissions will incur a penalty of 10 minutes.

For example, if one hour after the contest starts, you get a problem right on your third try, we will consider you to have taken 60+10+10=80 minutes to solve the problem. If you get your second problem fifteen minutes later on the first try, you will have taken 75 minutes to solve that problem, and your total "minutes" is 80+75=155.
Input/Output

While there are dozens of ways to perform I/O in C and C++, in this contest, you will be fine if you only use C++'s `cin` and `cout`. For example, to output the message "Hello world", you would do this

```cpp
cout << "Hello world" << endl;
```

Most of the time, you'll want to finish up a `cout` with an `endl` to end the current line of output.

If you have an integer `x`, you can print its value in the same way

```cpp
cout << "The value of x is: " << x << endl;
```

When you want to read inputs, you use `cin`. For example, to read an integer into the variable `x`, you can use

```cpp
cin >> x;
```

Tricks you should use

You may need to know about these "tricks" to complete some of the problems in this contest.

- A single character (like the letter 'A' or the numeral '4') can be converted to an integer very easily. For example, suppose you have a character stored in the variable `myChar`, and you know it's a capital letter. The statement

  ```cpp
  myInt = myChar - 'A';
  ```

  will store a number into `myInt`, where A is 0, B is 1, C is 2, ..., and Z is 25.

- The `for` loop. Every problem should be solved with at least one loop, and some will require several nested loops. While they look complicated, a `for` loop is actually quite simple. Consider:

  ```cpp
  for ( A ; B ; C )
      something();
  ```

  No matter what A, B, and C are, here are the rules.

  1. A is executed only once, before anything else.
  2. Then B is executed. If it returns 0 or false, quit.
  3. Whatever is inside the loop (here, "something()") is run.
  4. Finally, C is executed.
  5. Return to step 2.

- The area of the region inside any three points in the plane is given by the absolute value of one half the cross product of the two vectors from a point to the other two points.
C and C++ have a built-in modulus (remainder) operator. It is the percent sign %.

\[ x = 24 \% 7; \]

**Tricks you should not use**

The problems in this contest are fairly straightforward. If you find yourself using any of these tricks, you are making things too hard for yourself and there is definitely an easier way to do it. In other words, do not use these tricks:

- Dynamic memory allocation. This means C++’s `new` keyword and it means the `malloc()` family of functions.
- Anything more than 8th grade math. You won’t need to include `<math.h>`, and you won’t need to use sines or cosines.
- Fancy data structures. Fancy algorithms. Fancy STL. While you can use maps and functors if you want, you won’t need them. You’re just showing off.
- Custom C++ classes. You may use them, but it’s probably easier not to.
- Elaborate algorithms that you learned in class last week. You won’t need quicksort, dijkstra’s algorithm, or huffman codes.

**Sample Problem**

Your program is given an integer that indicates how many letters will follow, then a list of uppercase letters (one per line). For each letter, a correct program outputs "even" or "odd" depending on whether, when converted to a number (A=1, B=2, C=3, ..., Z=26), the resulting number is even or odd.

**Sample Input**

```
3
A
N
J
```

**Correct Output**

```
odd
even
even
```