Contest Coding

Alex Chen

October 14, 2011

1 Introduction

Over this year, we will be focusing on the USA Computing Olympiad (USACO), a series of monthly programming contests. In this "lecture," we will go over some tips to consider when writing programs for USACO and similar contests.

Contest programming is different from "regular" programming in many ways. You can get away with being slightly sloppy, such as forgetting try-catch loops and putting everything into one class, things you would not want to do when programming for real-life applications. However, that does not mean contest programming is worthless. The foundations learned in contest programming will only lead you to becoming a better programmer for the future. Contest programming is generally timed—with practice, programming becomes very intuitive and will come naturally.

2 Coding Tips

The USACO languages are **Java**, **C++**, and **Pascal**. Very few people around here use Pascal, so this handout will focus on Java and C++. The two languages are similar in many ways, in fact, so if you've already learned Java I would suggest giving C++ a try. When reading the following tips, keep in mind that they are merely the opinions of the author. Feel free to agree or disagree.

2.1 General Tips

- Good contest code is **legible**. There is nothing more frustrating than debugging a program that you cannot comprehend. Name your variables logically and use tabs and blank lines strategically. Add comments when necessary to remind yourself what certain bits of code did.
- Good contest code is **concise**. This does not mean that you should squeeze all your code into as few lines as possible, but rather that you should avoid coding more than you have to. If your code gets excessively long, you are probably missing out on a better way to solve the problem. USACO solutions tend to be very short and pretty.
- Good contest code can be **modular**. Modular code has more subroutines. Some coders, including me, like to create a separate method for each subroutine in the program. Others like to put all the code into one large method. I personally prefer modularity because modular code is easier to debug and to understand.
- The most important aspect of contest code is its **correctness**, and not how quickly you code or how elegant the solution is.
- Good contest code is not necessarily professional. In class, you have probably learned to handle exceptions when opening and closing files in Java (try-catch statements). With contest coding, at least for timed contests such as USACO, don't bother. Just declare throwing an exception when initializing a method and code the rest as though you were sure that the relevant files exist. For USACO, never print error messages (except for debugging purposes) or messages that prompt the user for input, because the grading system is automated.

2.2 Java Tips

Many of you will switch to C++ in the future, but for now Java is a perfectly acceptable language to compete in. In general, you should take a look at the sample Java and C++ programs on the Grader for examples of good contest code.

- Use a BufferedReader for input. Check the API for more details.

  ```
  BufferedReader input = new BufferedReader (new FileReader ("filename.in"));
  StringTokenizer st = new StringTokenizer (input.readLine(), " ");
  String string = st.nextToken();
  int number = Integer.parseInt(st.nextToken());
  ```

  A combination of a BufferedReader and a StringTokenizer is more efficient than using a Scanner or using a BufferedReader with String.split(). This will probably not matter for Bronze, but as the problems get harder the input tends to get more complicated, and efficient input methods actually matter.

- Use a PrintWriter for output. Check the API for more details.

  ```
  PrintWriter output = new PrintWriter (new BufferedWriter (new FileWriter ("filename.out")));
  output.println ("This is my output.");
  ```

  This is usually a better idea than System.setOut() because the latter prevents you from printing debugging code to standard output while you are running the program.

- For the most part, don't bother with polymorphism, extending classes, creating abstract classes or interfaces, and complex structures. These are typically not necessary for contest programming. Often, classes will serve as little more than a way to group variables and sort them.

- `java.util.Arrays.sort()` sorts arrays for you! This is very convenient.

2.3 C++ Tips

- Initialize your arrays globally. This way, they are automatically set to default values rather than to random garbage.

- Use fscanf and fprintf for input. Look them up online for guidelines. If efficiency is not a concern, ifstream and ofstream should work as well.

3 What Else?

Using USACO as an example, a "programming contest" isn't all about programming. During a 3-hour contest, many of the country's top programmers often spend 30 to 60 minutes sitting in front of the computer, thinking but not typing. USACO is more about creating algorithms than coding, but it is impossible to succeed without knowing how to code your algorithms. Thus, although it is good practice to look over problems and solve them, it's still necessary to practice coding your algorithms every once in a while. The coding, debugging, and testing process is often harder than one would expect.

4 Have Fun!

Don't forget to register for the grader at `activities.tjhsst.edu/sct/grader` and try out the problems. In addition, the first USACO contest is most likely next week. Register for the training pages, the *best* online training resource, at `train.usaco.org` and check out their problems!

Meanwhile, feel free to join the computer team mailing list. Visit the website (`activities.tjhsst.edu/sct`) and click on the pink link on the front page to find the mailing list. Alternatively, go to `http://lists.tjhsst.edu/cgi-bin/mailman/listinfo/scteam`. The SCT website is regularly updated and holds all the lectures from each meeting.

5 Practice Problems

1. (Grader: A+B; easyprob) Given N and a list of N numbers, find and print their sum.
2. (Grader: Week 3: fibo) Given N, find the N^{th} Fibonacci number.
3. (Grader: Week 2: puddles) Bessie is on a rainy farm and she wants to find the number of puddles she can jump in. Given a square grid of #'s and W's, find the number of puddles of W's. Two W's are part of the same puddle if the two square cells share an edge.
4. (Grader: Week 3: numsq) Bessie starts at the top of a grid of numbers. At every step, she moves down one row and can move right one column, left one column, or not change columns at all. Each cell of the grid contains grass that has a certain rating. Find the path from the top row to the bottom row that goes through grass with the maximum possible total rating. She can start at any cell in the top row and end at any cell in the bottom row.

6 Sample C++ Program

```
/*
PROG: easyprob
LANG: C++
*/

#include <iostream>
#include <cstdio>
#include <cstdlib>
//#include <fstream> // alternative output method
//#include <algorithm> // sorting functions and binary search
//#include <utility> // pair

using namespace std;

int N, sum = 0;

int main()
{
    // you can also use ifstream and ofstream
    //  but those are slower
    FILE * input = fopen("easyprob.in", "r");
    FILE * output = fopen("easyprob.out", "w");

    fscanf(input, "%d", &N);
    for(int i = 0; i < N; i++)
    {
        int x;
        fscanf(input, "%d", &x);
        sum += x;
    }

    fprintf(output, "%d\n", sum);
    fclose(input);
    fclose(output);
    return 0;
}
```

7 Sample Java Program

The following is a solution for problem 1.

```
/*
PROG: easyprob
LANG: JAVA
*/

// The two standard imports that cover most useful classes.
import java.util.*;
import java.io.*;
// import java.math.*; // BigInteger

public class easyprob // always use the problem name here
{
    // throw an IOException here so there is no need to handle errors later
    public static void main(String[] args) throws IOException
    {
        // These two classes are rather fast for input and output.
        BufferedReader input = new BufferedReader(new FileReader("easyprob.in"));
        PrintWriter output = new PrintWriter(new BufferedWriter(new FileWriter("easyprob.out")));

        int N = Integer.parseInt(input.readLine());
        int sum = 0;
        for(int i = 0; i < N; i++)
        {
            int next = Integer.parseInt(input.readLine());
            sum += next;
        }

        output.println(sum);
        output.close();
        System.exit(0);
    }
}
```