Information Packet
Trinity University ACM High School Programming Competition
2011

This packet contains the significant information that you need to know for the competition including the rules, how the competition will be run, and helpful information that might aid you in some of the problems.

Schedule

Time	***Activity***	***Location***
10:00-11:00	Registration and Welcome	Chapman Auditorium
11:00-11:30	Practice Run	Halsell and Cowles
11:30-12:30	Lunch	Chapman Auditorium
12:30-3:30	Competition	Halsell and Cowles
3:30-4:30	Awards Presentation	Chapman Auditorium

Rules

The competition is being run in an ACM style where teams of up to 3 members are given a packet of problems to solve and they have 3 hours to solve as many as they can. Each team has access to one computer, one keyboard, and one mouse. All the problems are of equal value. Teams are ranked first by the number of problems that they solve and second by the number of points they got in answering those problems. Teams want to have as few points as possible for the number of problems they solve. A correct submission gets one point for every minute since the beginning of the competition and 10 points for all extra submissions. Even if you submit two working copies of a problem you will get the 10 penalty points for the extra submission.

The competition will be held on Trinity machines running the Ubuntu Linux distribution. They will have Eclipse set up for editing programs and have Java 6 installed. You can use any of the Java standard libraries as a full Java version will be provided. The environment and program used for submission are discussed in more detail below so you can easily get started without having any prior knowledge of Linux or Eclipse. One thing to know about the Ubuntu live CD is that it doesn't use the local disk drives. It puts things in a RAM disk. For this reason it is strongly suggested that you NOT reboot your computer for any reason. We are providing tools so that you can back up source files, but you should not intentionally rely on this. It will slow you down a lot to do so.

Each team will have one computer. Teams should not bring any other computing devices, nor should they bring anything in machine readable format. Bringing disks, CDs, thumbdrives, or any other machine readable devices into the competition area will be grounds for disqualification. The submission application also provides you with a mechanism for printing your code. The code will be printed in a different room and a runner will bring it to you.

Submissions will be done electronically using an application that is provided to you. On a separate sheet you are provided with your team login name and password. You will use these to log into the submission program. You can use this program to submit code for problems as well as to ask clarifications about certain problems. After you submit the source code file for a problem, the judges will compile and run the program and reply to you to let you know whether it worked. The judge reply will be one of the three following options.

Correct – Your submission produced acceptable output and was accepted. Don't submit it again.

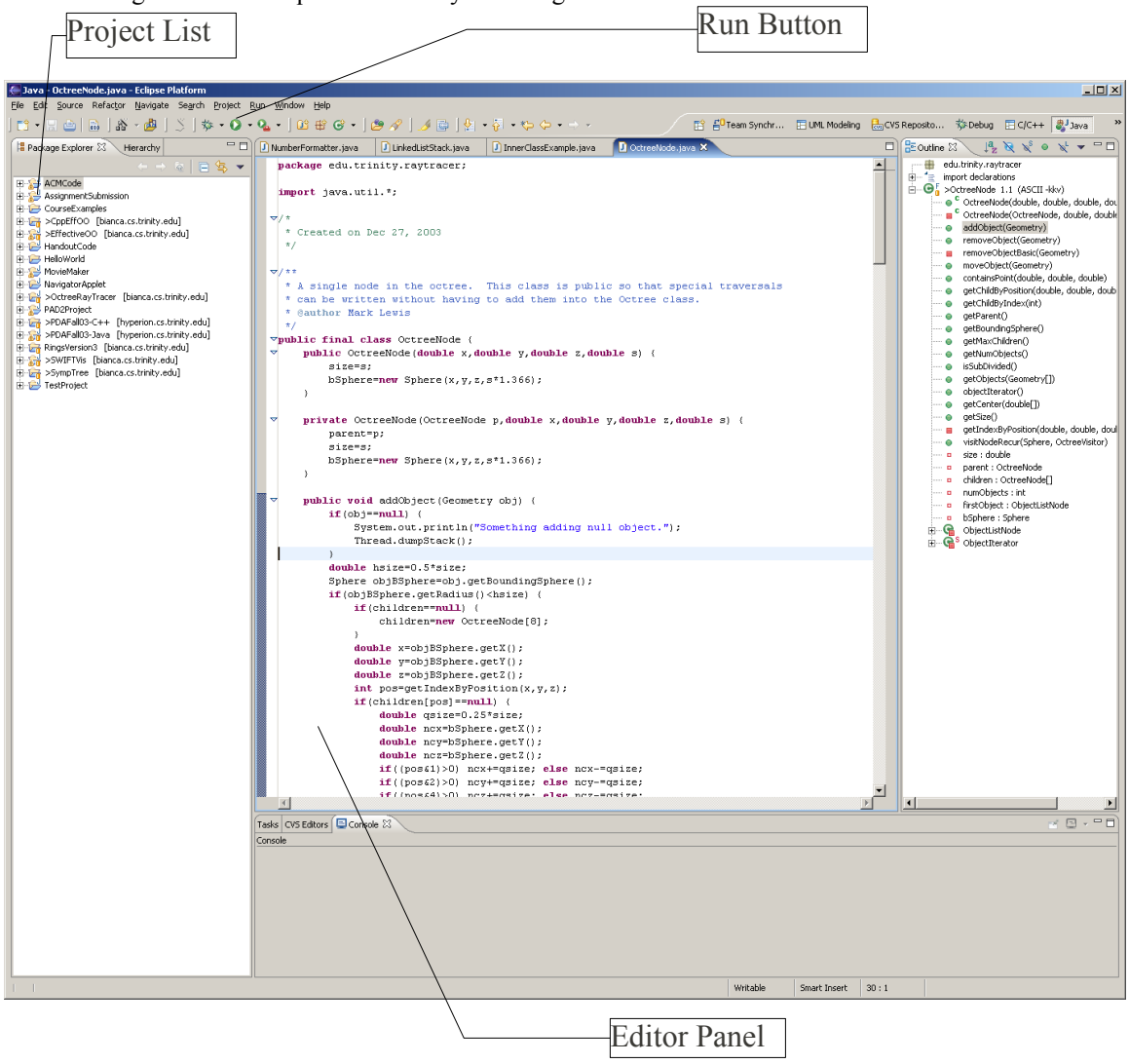
Incorrect – This implies that something is wrong with your output and that it doesn't match the expected

values.

Timeout – There are two causes for this. One is that your solution simply takes too long on our input or goes into a infinite loop. It could also mean you aren't reading the input correctly. If your program is waiting for input after all the input has been given this could cause it to timeout.

Ubuntu and Contest Environment

There really isn't much to say about the contest environment. For those who care, it is running Gnome though many of the larger applications have been stripped out (things like OpenOffice) to make room for the development tools needed for the competition. To keep things simple, we have added icons for the three applications that you will need during the competition: Eclipse, a web browser (for the Javadocs), and the submission application. The standard menu can lead you to other applications should you choose to do other things. Surfing the web isn't an allowed other task though and attempting to find code samples on the web will be grounds for disqualification if you manage to do it.



Basics of Eclipse

The Ubuntu CD that you will be booted from includes a copy of Eclipse that you can use for editing your Java source files. Being Linux you could also feel free to use a standard text editor like vi and do command line compiles, but we think you'll find it much more productive to use Eclipse (if for no other reason than features like syntax highlighting, name completion, and fast importing). In this section we want to give you a quick introduction to Eclipse so that you can quickly get over some of the early learning curve issues and then quickly find the features that will help you most during the contest.

When you bring up Eclipse you will be asked first for a workspace location. Pick the default. You will then be dropped into the workspace which has a number of different panels. The figure above shows a sample Eclipse run with different areas labeled. In order to get anything done, you want to create a new Java project. There are many ways to get a wizard or menu brought up for creating things in Eclipse including right clicking on the "Package Explorer" on the left side of the workspace, using the "File" menu, or using the icon under the "File" menu on the toolbar. You can call the project whatever you want. The new project will appear in the "Package Explorer". You can expand it. When you want to create a new program, you will right click on the project and do "New > Class". Give the class a name, but put it in the default package. We recommend that you use the names Prob0 through Prob9, just in case we have to revert to judging from disks. You only get to submit a single file and that file have to be in the default package so don't create other packages or write code for a problem that spans multiple files (use inner classes or non-public classes if you want more than one class).

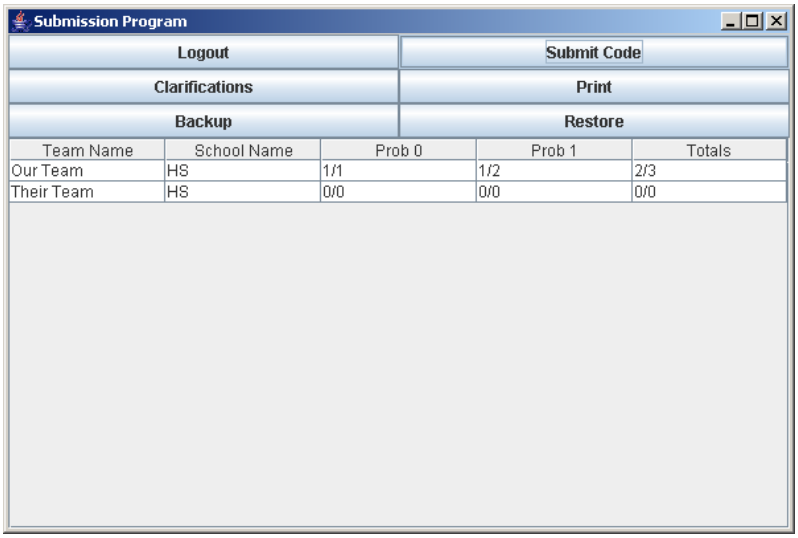
Once you have a new class file up you can simply code. Eclipse has some nice features to help speed up coding. The syntax highlighting is very good and you can click on the error icon on the left side of the edit pane for certain errors and it will suggest fixes. If Eclipse ever reports errors that you feel you have fixed, save the file and if it really is fixed it will go away. After typing a class or object name followed by a dot, you can pause and Eclipse will bring up a list of possibilities. You can also get a list of options at any time by hitting Ctrl-Space. It also lists Javadoc descriptions of those functions. If you use a class in a file that you haven't imported, put the cursor on the class name and press Ctrl-Shift-M and it will automatically add an import statement. There are lots of other features as well, but those are the ones that are likely to be able to help you during the competition.

To run and test your program, you can use the Run menu or the run icon (green circle with arrow) on the toolbar. The first time you run a given program select "Run As > Java Application". After that you can either repeat the last run by clicking on the green circle icon or pick the program from the drop down menu by the icon. The bottom of the workspace has a console that you can provide input to and see your output. You can also copy and paste into this console. We might recommend that you put a comment in your code with the input that you want to run and copy that then paste it into the console so that you don't have problems with typos or having to repeat typing stuff. Make sure to get the newline after the last line of input so that your program will terminate. We've made sure to have it so that you never read to the end of a file, but when you read a full line the program will typically wait until the linefeed is reached.

You can put all of your different programs in the one project in the default package so there is no need to create multiple different projects. We are using Eclipse 3.1 this year so everything should work fine with any Java 5.0 options that you feel like using, including generics, autoboxing, and the Scanner class.

Submission Application

We are providing you with a Java application that allows you to submit your programs to the judges and find out the results electronically. When the competition starts you should log into the submission application using the login name and password that your team was given. You can, and probably should, stay logged in the entire competition to speed up the process of submitting problems. The application will also allow you to view the current standings and to see any clarifications that the judges have posted. The standings will stop updating for you when there are about 30 minutes left in the competition. The application allows you to pick from 6 different operations: Login/Logout, Submit Code, Clarifications, Print, Backup, Restore. We'll look at the functionality of each of these in turn.



The login is rather simple and will bring up a dialog box in which you can enter your team's login name and password. Be careful not to lose the piece of paper that has your login information. Getting that information to you during the contest will take a while and will likely slow you down. Once you are logged in you will see the standings for the competition and have the ability to log off or select the other options. It is unlikely that you will have to log out during the competition. The standings show a grid with one school per row and the columns have team names, school names, problems, and the totals. The problem columns show two numbers for each team. The first number is how many times the problem has been submitted by that team. If the problem has been answered correctly then the second number shows the number of penalty points the team got for that problem. The total column has the sums of those values. Teams are displayed in sorted order of the standings.

The most important use of the program for you is to submit your solutions to the problems. When you click the submit button, a dialog box comes up that will ask you for the problem number and the file that you want to submit as shown in the figure above. Make sure that you select the correct problem number. Failure to do so will inevitably lead to your submission being judged incorrect, mostly likely with some error on reading the input in. You can only submit one .java source file in a given submission so keep everything in the solution of a problem down to a single file. After you submit you can go on to work on another problem. When a judge has judged your submission, you will be notified of the result by a dialog box that will pop-up on the submission application. If you put any debugging print statements in your code, make sure to take them out before you submit because extra print statements will cause your problem to be judged as incorrect. We suggest that you consider printing debug information to System.err as the judges won't see that. Alternately, use the debugger instead of printing.

Also potentially of significance to you during the competition is the clarifications option. This will bring up a window that lists the current clarifications and allows you to submit requests for clarifications. We have tried to insure that the problem descriptions include sufficient descriptions of the problems, but it is possible that some information has been left out. If you have questions about the intent of a question, you should submit a clarification. Clicking the “Clarifications” button brings up the window that is shown below. It has a button you can click to submit your own clarification with a table below that showing all the clarifications. Clarifications that have not been replied to might not appear. Note that replies to any

clarifications will go out to all teams.

One thing that you will notice about your work area is that you do not have a printer by your computer. If you want to print a file you can click the "print" button on the submission application. This will allow you to select a file to print. Please only print the .java files. When you do a print, a message will come back to tell you if it worked. Safeguards have been put in place to help insure you don't print binary files. After you print, a runner will bring your printout to you.

The last two options that you have for the submission application are to provide you with some ability to backup and recover your work. Linux is quite stable and hopefully you will never have a problem with this, however, we want to cover things. The reason this option exists is that Ubuntu isn't saving anything to the hard drive. Everything you are doing is happening in a RAM disk. If the machine goes down, all of that is gone. The Backup option will send all of the .java files in your workspace to the server for storage on disk. The recover option will let you pick a file to recover and will restore that file to your machine from the server. Be careful because any local version that you have will be overwritten in this process. As a hint, if your machine does go down, when you restart, name your Project in Eclipse the same thing that you named it the first time. That will make things easier for dealing with the recovery.

If for some reason the electronic submission does not work (we really hope this doesn't happen) then you will have to use floppy disk submission of your programs. The procedure for this is fairly simple. After you have your program working, make sure you save it, then click on the home icon on the bottom bar and go into workspace, then into your project directory. Next put the floppy disk in the machine and click on the floppy icon on the desktop. When the floppy drive window comes up, you can drag and drop the file you want to copy over to it and select copy (don't move because you want the original and the judges will delete the disk before it comes back to you). Make sure that the name of your program indicates the number of the problem you are submitting for (Prob0.java is a good format). After it is copied over, close the floppy window, eject the floppy, and give the disk to a runner to take to the judges. If you are familiar with Linux and decide to copy at command line make sure you umount the disk before taking it out.

Some Java Information (just in case)

This competition is being run in a way that is different than most of the competitions you have likely been in since it is modeled after the collegiate ACM competitions. As a result, it is possible that you might not feel comfortable with some of the basic aspects of this competition. If you have seen these things before that's fine, but we want to make sure everyone starts off with an even footing when it comes to the basics. The thing we want to make sure absolutely everyone can do is read from standard input. If you are used to doing file input, reading from standard input isn't really any different, you just read from the InputStream System.in. Personally, I like to wrap System.in in a stream that can read whole lines. This can be quickly done with a line like the following.

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

The advantage is that BufferedReader has a readLine method that reads everything up to the next newline character and returns the result as a string. The string will not contain the end of line. If the end of the stream has been reached, then this method returns null. With Java 5.0 there is also a Scanner class that we will discuss below. It is probably superior to BufferedReader for this competition.

Of course, there might be multiple pieces of information on a line that you need to get at. This just requires breaking up the string into multiple parts, normally based on whitespace, but possibly based on other characters. Prior to Java 1.4 this was typically done with the StringTokenizer class in the java.util package. Since 1.4 you might find that the task can be done even more easily with the split method in the String class. This method returns an array of Strings and takes a string that is a regular expression. So if you have the variable br declared above, you could get the individual words on a line with the following piece of code.

```
String[] words=br.readLine().split(" ");
```

With this invocation the array will have a length equal to the number of words on the line. If some of the words are actually numbers remember to use Integer.parseInt and Double.parseDouble to convert the Strings to ints or doubles. This code will have to go inside a try/catch that catches IOException because readLine on the BufferedReader can throw that. If you use Scanner that isn't an issue.

Because you can use Java 1.5 you could also consider using the java.util.Scanner class. This can be done with Scanner scan=new Scanner(System.in);. The Scanner class then provides methods such as next() to read a word or nextInt() to read an integer. These methods throw only RuntimeExceptions so you don't have to have them in a try/catch block. If you choose to use this, keep in mind that if you mix calls to nextLine() with other calls, you might have to throw in an extra nextLine() to finish a line you already got input from. For example, if you have one line with an int followed by a line with a full sentence, a call to readInt() followed by readLine() will get an empty string for the readLine() because it only reads the newline following the int. To get the sentence, you need an extra call to readLine().

Since your programs are judged based on what appears in the standard output, there are suggestions for how to put in debugging print statements in such a way that you are less likely to accidentally leave one or two in and cause your submission to judged incorrect when you have the right algorithm. The easiest way to deal with this is to print to System.err for debugs and only print to System.out for the problem output. The one possible problem you might run into with this is that sometimes System.err and System.out are not synchronized so print outs might appear out of order of when they are executed. If that causes you a problem you could always write a method, possibly called debug, that takes a string and prints it, but only prints it when a certain static final variable is true. That way you can remove all debug prints by just setting that variable to false.

The next page contains a sample problem. This problem has the same general format of most of the problems in the contest set. During the pre-contest trial time the submission program will be set up for you to do this problem. It is suggested that during that time you write and submit a solution to it. Note that the first line of the input tells you how many data sets will follow. Most of the problems have this format and all of the problems are set up so that you never have to detect and end of file. We suggest the if you use the Java 1.4 libraries that your main look something like this (obviously formating and variable name details are up to you).

```
public static void main(String[] args0) {
	try {
		BufferedReader br=new BufferedReader(new InputStreamReader(System.in));
		int inputs=Integer.parseInt(br.readLine());
		for(int i=0; i<inputs; ++i) {
			// Read in the input for that set.
			// Process it and do the output.
		}
	} catch(IOException e) {
		e.printStackTrace();
	}
}
```

If you use Scanner instead, your main will look something like this.

```
public static void main(String[] args0) {
	Scanner sc=new Scanner(System.in);
	int inputs=sc.nextInt();
	// might need a sc.nextLine(); here.
	for(int i=0; i<inputs; ++i) {
		// Read in the input for that set.
		// Process it and do the output.
	}
}
```

Not all the problems will fit this perfectly, but it gives you a good idea of how to set things up.

As a last point, if you aren't familiar with the HashMap class, it can make your coding for some of the problems a lot nicer. You can solve all of them without a HashMap, but the code will be easier to write with one. HashMap is an implementation of a Hashtable. It is a container that stores things by keys and values. You can efficiently look up any value based on the key. In a way you can think of it as an array where you can look things up by something other than a number. So if you had a problem where you had to keep track of how many rocks different people had gotten you might do the following.

```
HashMap<String,Integer> hash=new HashMap<String,Integer>();
hash.put("Fred",5);
hash.put("Barney",4);
hash.put("Wilma",7);
hash.put("Betty",6);
// Now look something up
String name=sc.next();
int rocks=hash.get(name);
```

Problem 0 – Sample Problem

To get you started and see if you can use the programs and do some of the basics for this competition, we want you to do a slightly more advanced version of "Hello World". The difference between this version and your normal version is that you will say hello to multiple specific people instead the whole world.

Input: The input for this problem consists of a header line that tells you how many people you will say hello, to followed by one line for each person that lists their name.

Output: For the output of this problem you should print "Hello " followed by the name of the person you are saying hello to for each name listed.

Sample Input:
4
Mark
Jason
Tyler
Andy

Sample Output:
Hello Mark
Hello Jason
Hello Tyler
Hello Andy

Problem 1 – Bits-to-ASCII

In a fit of incredibly geekiness, your friends have decided that you should start encrypting messages you send back and forth by putting them into a binary representation of the ASCII values. After all, who is going to try to find meaning in a whole bunch of seemingly random 1s and 0s? To make it even tougher for people to find, you decide you will put the messages into web pages as comments so people will only see them if they view the source of your HTML files. Your friend is going to write the code that converts the original message over to binary bits. Your job is to make a program that will take a string of ones and zeros and print out the original message.

Input: The input to the problem will begin with a single number on a line telling you how many input sets there are. Each input set will be a single line with a string of 1s and 0s. The string will have a length that is a multiple of 8 because there are 8 bits in each byte you need for ASCII values. All strings will be 800 characters or less in length.

Output: For the output, you should print one line for each input. The line will be the decoded text for the input values.

Sample Input:

```
4
01000011011000010110111000100000011110010110111101110101
01110010011001010110000101100100
01100010011010010110111001100001011100100111100 1
01000001010100110100001101001001010010010011111 1
```

Sample Output:

```
Can you
read
binary
ASCII?
```