CS1102: Data Structures and Algorithm
Course Marker Guide
(Part 2 of Lab 0 Walkthrough)

Please note that:

Information for Lab 0:

- It is a purely take home lab, i.e. **no actual lab sessions will be conducted**.
- No deadline of submission. But you should try to finish it by **3rd February to get ready for your subsequent lab.**
- The total weightage is 2%:
    - Exercise 1 (A simple Java problem) 2 %

Hello again! This is the second and final part of the CS1102 Lab 0 walkthrough. This document is to guide you on the following topics:

- Using the ***CourseMarker*** system
- Trying out Lab 0, which comprises of ***one exercise***

Section 1 Overview

We use the CourseMarker system for two very specific purposes:

1. For *retrieving laboratory problems*. When you do this, various files are copied to your sunfire accounts from the CourseMarker server. These files may include a copy of the question asked, sample data files, an initial program, a makefile and so on. The files will be placed in a (new) directory under your sunfire account.

2. For *submitting laboratory problems*. When you do this, various files are copied from your machine to the CourseMarker server, where they are given an automatic assessment. You will immediately see the results of this automatic assessment.

In between retrieving and submitting problems, you can work on the local copy of your problem just like a normal Java program (i.e. the exact same steps in Walkthrough Part 1 can be used).

Section 2 Using the CourseMarker command-line client interface

The following steps are written as a walkthrough for your CS1102 lab 0. Before you start, make sure you know the following:

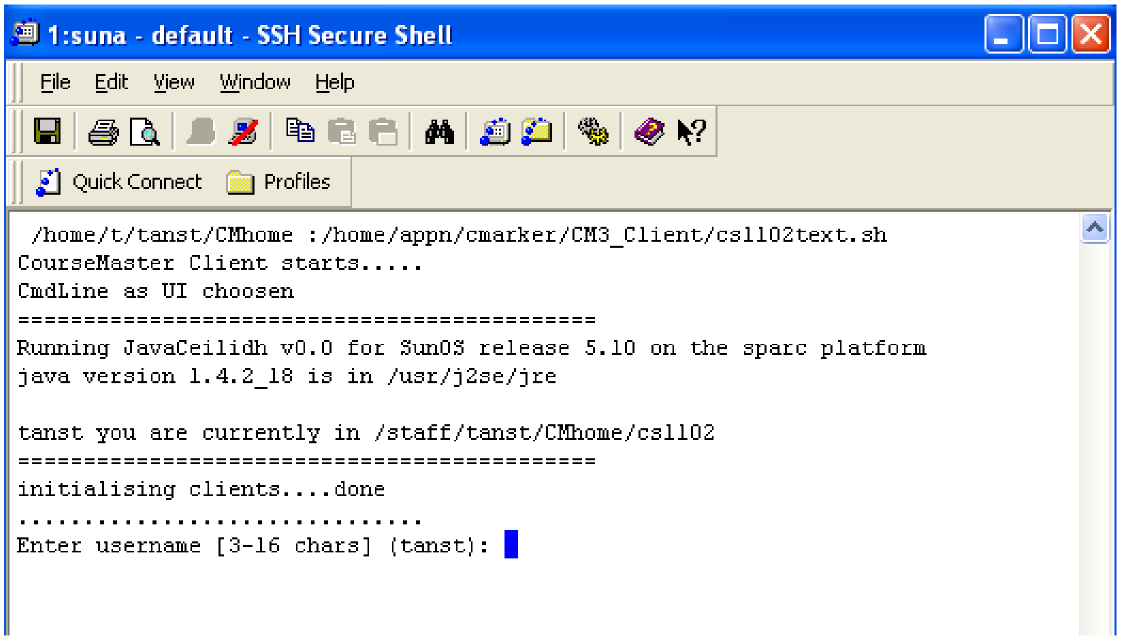
1. How to connect to your sunfire account (see Walkthrough Part 1 if you don’t)
2. CourseMarker account id and password. You should have received an email containing your CourseMarker account id and password in your NUSNET email account.

2.1 Starting the CourseMarker Client

Make sure you are connected to sunfire and type the following at the command prompt.

```
/home/appn/cmarker/CM3_Client/cs1102text.sh
```

You should see the following screen if you have typed in the command correctly.


1:suna - default - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

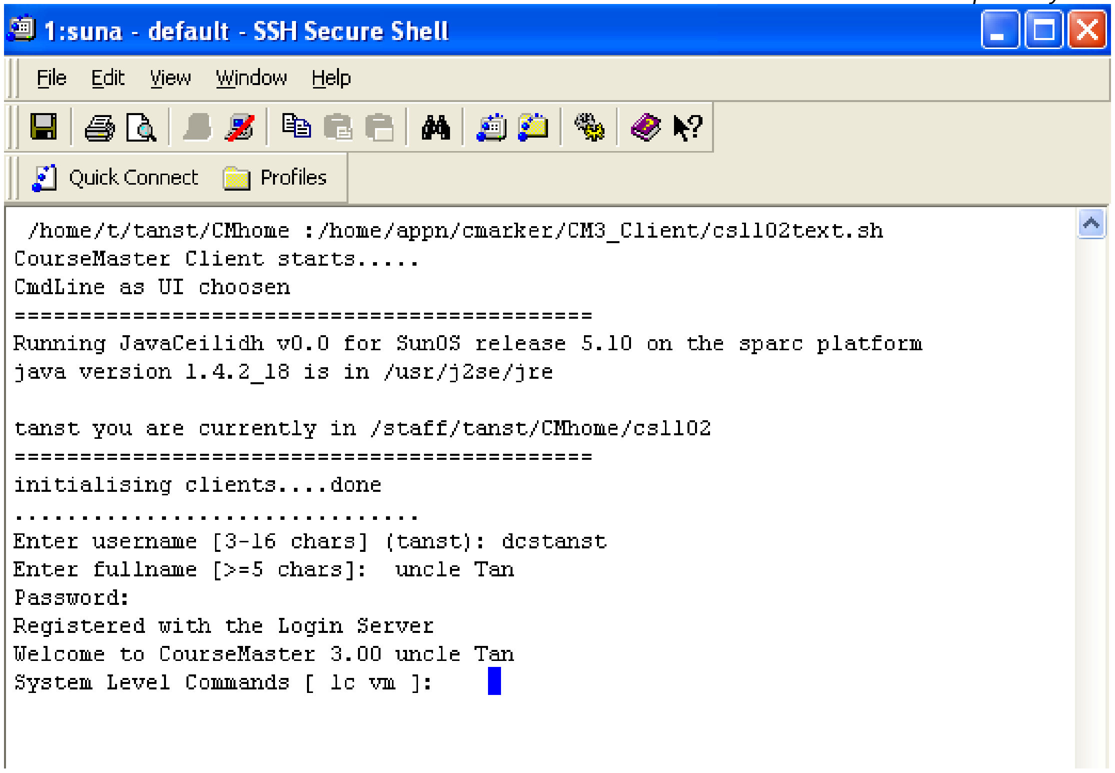
```
 /home/t/tanst/CMhome :/home/appn/cmarker/CM3_Client/cs1102text.sh
CourseMaster Client starts.....
CmdLine as UI choosen
============================================
Running JavaCeilidh v0.0 for SunOS release 5.10 on the sparc platform
java version 1.4.2_18 is in /usr/j2se/jre

tanst you are currently in /staff/tanst/CMhome/cs1102
============================================
initialising clients....done
...............................
Enter username [3-16 chars] (tanst): 
```

The CourseMarker system will prompt you for the following:

1. **`Username`**: Enter your CourseMarker id
2. **`Full name`**: Give your full name. This is only used for the CourseMarker system to greet you ☺. So, you can even type in a nickname if you like. This question will be asked only once. Yon will not see this prompting in subsequent usage of CourseMarker
3. **`Password`**: Enter your CourseMarker password

Note that item (1) and (3) should have been sent to your NUSNET email account. Please note that both username and password are case sensitive. Also, if your username shown in the bracket is correct, you can simply type ENTER instead of keying in again.

After CourseMarker verified your id and password, you will be greeted by a short welcome message.


1:suna - default - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
 /home/t/tanst/CMhome :/home/appn/cmarker/CM3_Client/cs1102text.sh
CourseMaster Client starts.....
CmdLine as UI choosen
============================================
Running JavaCeilidh v0.0 for SunOS release 5.10 on the sparc platform
java version 1.4.2_18 is in /usr/j2se/jre

tanst you are currently in /staff/tanst/CMhome/cs1102
============================================
initialising clients....done
...............................
Enter username [3-16 chars] (tanst): dcstanst
Enter fullname [>=5 chars]:  uncle Tan
Password:
Registered with the Login Server
Welcome to CourseMaster 3.00 uncle Tan
System Level Commands [ lc vm ]:
```

The next thing that stares at you would be the CourseMarker command prompt, waiting for your input. Before we look at the actual commands, let's have a high level view of the system. The CourseMarker system is organized as a hierarchy of the following levels:

- **`System Level`**
  Represents the whole system. This level contains:
  - **`Course Level`**
    Represents the courses that utilize the course marker. In our case, this would be cs1102. Each Course contains a number of units.
    - **`Unit Level`**
      Represents each of the labs. As we progress through the semester, you should see more and more units (i.e. labs) at this level. Each unit contains a number of exercises. For most labs, you will get only one exercise (one problem).

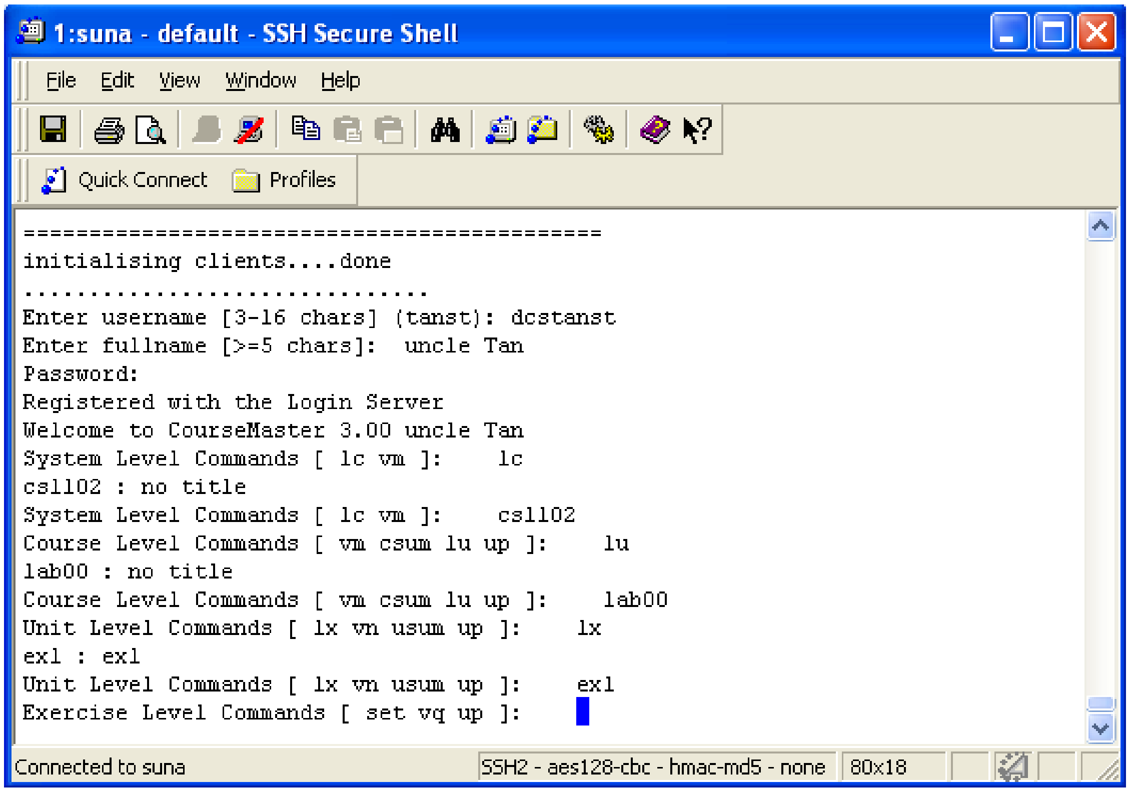
As shown in the screen shot, you will start out at the system level. At each level, the valid commands are listed in the [] brackets. For example, at the system level, the valid commands are "**`lc`**" and "**`vm`**". Note that only those commands covered in this walkthrough is useful for your lab. You can safely ignore those not mentioned in this guide. For the time being, let's enter "**`lc`**" (short for **`L`**`ist` **`C`**`ourses`), which shows all the courses you have under CourseMarker system. In our case, there will be only one course "**`cs1102 : no title`**". The "**`cs1102`**" is the course name and " **`: no title`**" is the course title.

To navigate to the course level, enter the course name "**`cs1102`**" (case sensitive). You will be presented with another set of commands [**`vm csum lu up`**]. The "**`up`**" command moves you back one level (i.e. back to the system level in this case). The "**`lu`**" command lists the available units (labs).

Go ahead and enter "**`lu`**". You should see one unit listed as "**`lab00 : no title`**". The meaning is similar to the course information: "**`lab00`**" is the unit name, " **`: no title`**" is the description of the unit. In subsequent labs, you will be presented with more units at this point and you have to choose the relevant one (usually the last one). Enter the unit name ("**lab00**" in this case) to get to the unit level.

At unit level, there are four commands [**lx vn usum up**]. Only "**lx**" and "**up**" are relevant to our purpose. "**up**" moves you back to the course level and "**lx**" lists the available exercises for this lab. Enter "**lx**" and you should see the exercise "ex1: ex1" listed as the only choice.

Please enter "**ex1**" (the exercise name) to reach the last level.


1:suna - default - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles
Connected to suna
SSH2 - aes128-cbc - hmac-md5 - none
80x18

```
============================================
initialising clients....done
...............................
Enter username [3-16 chars] (tanst): dcstanst
Enter fullname [>=5 chars]:  uncle Tan
Password:
Registered with the Login Server
Welcome to CourseMaster 3.00 uncle Tan
System Level Commands [ lc vm ]:    lc
cs1102 : no title
System Level Commands [ lc vm ]:    cs1102
Course Level Commands [ vm csum lu up ]:    lu
lab00 : no title
Course Level Commands [ vm csum lu up ]:    lab00
Unit Level Commands [ lx vn usum up ]:    lx
ex1 : ex1
Unit Level Commands [ lx vn usum up ]:    ex1
Exercise Level Commands [ set vq up ]:
```

This level is the most important level, so you should probably spend some time understanding the various commands. Initially, you will only see three commands : [**set vq up**]. "**set**" will download the relevant files (question, skeleton Java files etc) from the system to your sunfire account. "**vq**" (View Question) displays the lab problem and "**up**" transport you back to the unit level.

Enter "**set**" to download the necessary files to your account. If the command is successful, you will see quite a few more commands at the prompt.

Important Note:

The "**set**" command **overwrites** any existing file in the exercise directory. So, you should use this command **once** per exercise only. If you have written some code and use the "**set**" command again, your work may be overwritten! Be careful!

The additional commands do not concern us at the moment, you can enter "**`quit`**" to exit the CourseMarker program. You will get back to your sunfire account.

As a quick recap, the various steps described up to this point are needed to retrieve the lab question and download the necessary files to your sunfire account. We will see how to work on the lab question in the next section.

Tips:

It can get quite tiresome to type the long string of command to start CourseMarker. If you are feeling adventurous, you can try the following to create a "shortcut" in your sunfire account:

1. Go to the home directory under sunfire
2. Edit your "`.bashrc`" using any editor (e.g. "`vim .bashrc`")
   Don't forget the ".".
3. You should see a number of lines that start with "alias ………"
4. Add the following line at the end:
   `alias cmarker='/home/appn/cmarker/CM3_Client/cs1102text.sh'`
   Note that the "**'**" is usually located just beside the "**`enter`**" key. Do not confuse it with the backquote "`" key, which usually on the same key as the "**~**" symbol.
5. Save the file and exit the editor.
6. Type "**`source  .bash_profile`**"
   You should not see any error message. If you get any error message, please redo from step 2.
7. Type "`cmarker`"
   You should see the CourseMarker starting up. In future, you can just type "`cmarker`" instead of the full command to start up CourseMarker.

2.2 Working on the lab problem

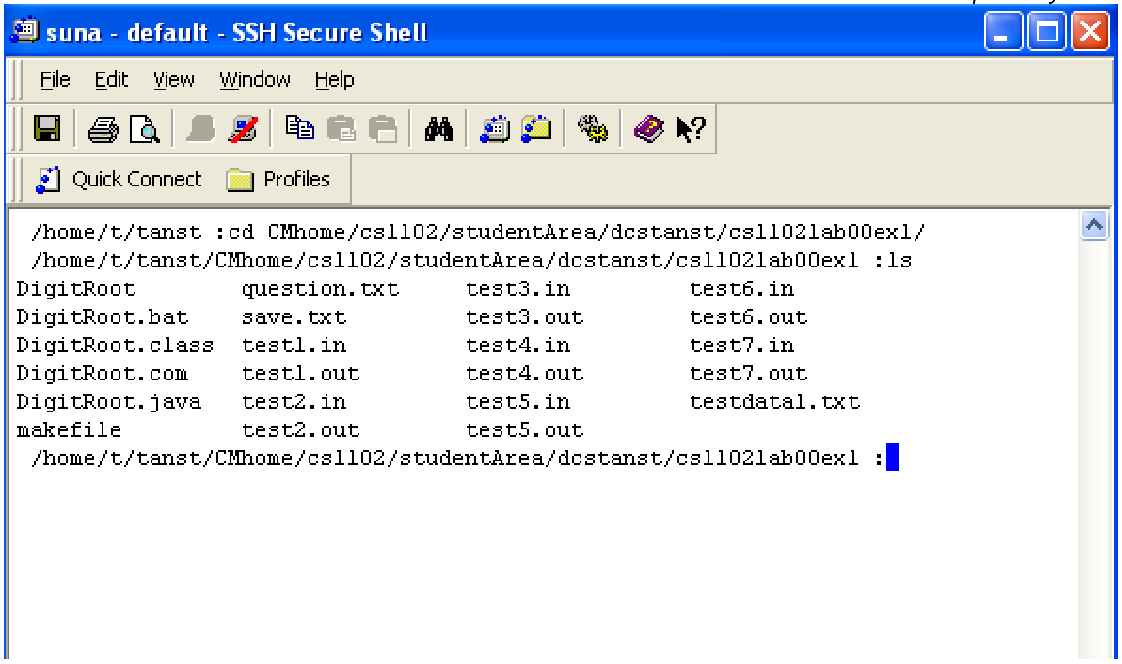
The various file downloaded from CourseMarker are stored under the following directory:

`CMhome/cs1102/studentArea/[Your CourseMakerId]/cs1102lab00ex1`

The last directory "**`cs1102lab00ex1`**" is simply the concatenation of the course name, unit name and exercise name. Navigate to this directory using the "**`cd`**" unix command. If you are not sure how to do it, here's one way:

1. Enter "**`cd`**" to go back to your home directory
2. Enter
   "**`cd CMhome/cs1102/studentArea/[Your CourseMakerId]/cs1102lab00ex1`**" to change to the desired directory.

After you arrived at the correct directory, enter "**`ls`**" to see the downloaded files.



```
 /home/t/tanst :cd CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1/
 /home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :ls
DigitRoot        question.txt     test3.in         test6.in
DigitRoot.bat    save.txt         test3.out        test6.out
DigitRoot.class  test1.in         test4.in         test7.in
DigitRoot.com    test1.out        test4.out        test7.out
DigitRoot.java   test2.in         test5.in         testdata1.txt
makefile         test2.out        test5.out
 /home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :
```

Do not be alarmed by the number of files, only a few are important to us:

`question.txt` :
This is a text file containing the lab problem description

`DigitRoot.java` :
This is the given skeleton Java program file. You are supposed to place your solution code in this file. In subsequent labs, you can enter "ls *.java" to show all given .java file.

`test*.in`:
These are the test cases used by the CourseMarker dynamic tests

`test*.out`:
The corresponding standard answer to the test cases

Important Note:

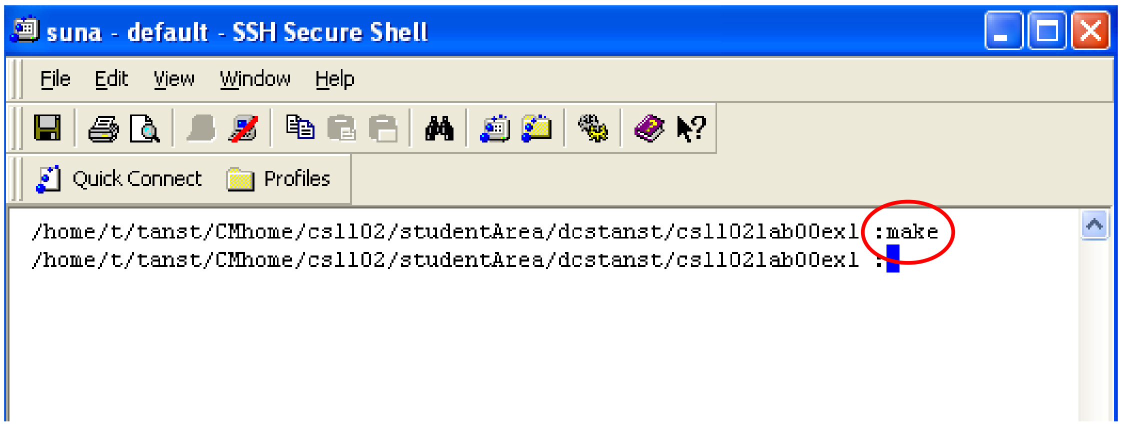
In subsequent labs, you may be given more than one .java files. The purpose of the various .java files is to promote good code organization. You should split your code accordingly. Also, it is important to note that during submission, CourseMarker **will only retrieves the downloaded .java files**. **If you introduce additional .java file(s), your program will fail to compile on the CourseMarker server!**

Let us first try to read the question. You can use any text displaying command in the unix. Here we use the "**`less`**" command:

1. Enter "**`less question.txt`**" to display the question.
2. Use arrow key to scroll around
3. Type "**q**" to quit the viewing.

I hope you find it a simple question to code ☺. Before you start coding, make sure you read the specification on the input and output carefully.

Please proceed to code the program in the given .java file (**`DigitRoot.java`**) using your favorite editor. Note that instead of using "**`javac DigitRoot.java`**" to compile your program, you should use the "**`make`**" command instead. Try it!



```
/home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :make
/home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :
```

If you look closely, you can see that the "**`make`**" command basically execute "**`javac`** ……" compilation for you. In subsequent labs, your may be given a number of files and it may be quite troublesome to compile them using the "**`javac`**" compiler directly. So, we have setup the "**`make`**" to manage the compilation process for you.

Debug your program until it can be compiled successfully. Note that the executable is produced as "**`DigitRoot.class`**".

Important Note:

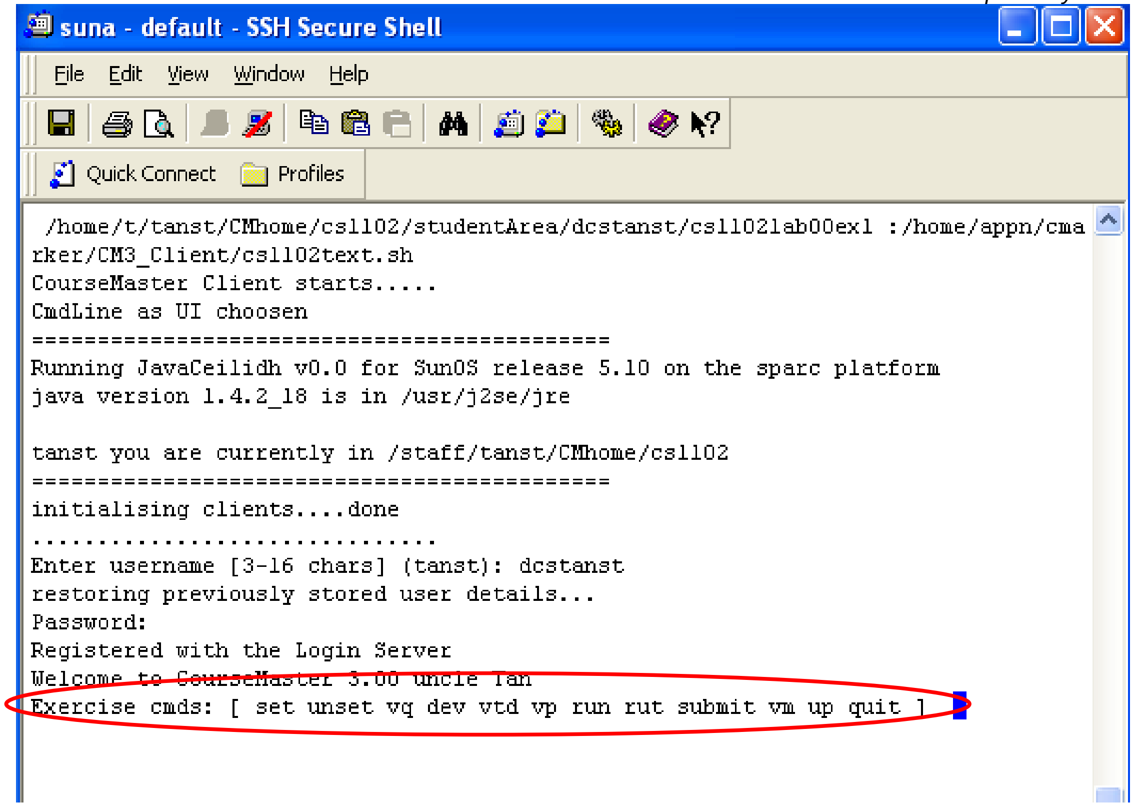
If you dislike the "**`make`**" command for some reasons, you can still use the "**`javac`**" compiler directly. However, make sure you produce the class file with the correct name, e.g. "**`DigitRoot.class`**" in this case. CourseMarker will try to detect a specific executable file before allowing submission!

Execute your program to test it out by typing "**`java DigitRoot`**". You should "torture" your program with various tests until you are sure it is correct. A more efficient way of testing will be covered in section Section 2.4.

After you are satisfied with your solution program, it is time to start up CourseMarker again to submit your solution.

2. 3 Submitting your program through CourseMarker

Start up CourseMarker using the steps given in section Section 2.1. Note that the CourseMarker system automatically resumes the last session (i.e. it will navigate to the last level you visited in the previous session). If you have followed closely to the steps given in this walkthrough, it should start you up in the "**`ex1`**" exercise level, as shown below:



```
 /home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :/home/appn/cma
rker/CM3_Client/cs1102text.sh
CourseMaster Client starts.....
CmdLine as UI choosen
============================================
Running JavaCeilidh v0.0 for SunOS release 5.10 on the sparc platform
java version 1.4.2_18 is in /usr/j2se/jre

tanst you are currently in /staff/tanst/CMhome/cs1102
============================================
initialising clients....done
...............................
Enter username [3-16 chars] (tanst): dcstanst
restoring previously stored user details...
Password:
Registered with the Login Server
Welcome to CourseMaster 3.00 uncle Tan
Exercise cmds: [ set unset vq dev vtd vp run rut submit vm up quit ]
```

If you are not at this level, don't panic. Just use the command given in Section 2.1 to navigate to the "**ex1**" level.

You can see that there are a few more commands available now. In particular, you should see the "**submit**" command listed. This command is enabled automatically as soon as CourseMarker detects the executable for the particular exercise (in our case "**DigitRoot.class**"). If this command is missing from the choices, then the most likely reason is that you failed to compile the program. Please exit CourseMarker and redo Section 2.2 to edit and compile the Java program for the lab problem.

Go ahead and type "**submit**". If the submission is successful (may take some time depends on the system load, please be patient), CourseMarker will give you an assessment of the submission. The result is usually a bit too long to fit into one screen, use the scroll bar on SSH client to view the full report. The full report looks like this (only partially shown):

```
(L1)===========================
Mark:60.0
Grading:C
Cweight:100.0
Description:General Grade
Feedback:Average
==============================
(L2)===========================
Mark:100.0
Grading:A
Cweight:0.0
Description:Compilation Tool
Feedback:Awesomely Excellent
==============================
(L3)===========================
Mark:100.0
```

```
Grading:A
Cweight:100.0
Description:Search for errors in the compilation
Feedback:No errors found during complilation
==============================

//... ... Snipped ... ... ... ... ...  ... ... //

(L2)===========================
Mark:0.0
Grading:F
Cweight:0.0
Description:Feature Tool
Feedback:Not so good
==============================
```

At first glance the report is quite confusing as it is all text. However, if you notice the label (L1), (L2), etc, you can see that it is again organized as a hierarchy (pretty much like a folder listing in windows). The hierarchy looks like this (only description of each level is shown):

```
(L1)===========================
Description:General Grade
==============================
      (L2)===========================
      Description:Compilation Tool
      ==============================
            (L3)===========================
            Description:Search for errors in the compilation
            ==============================
            (L3)===========================
            Description:Search for warnings in the compilation
            ==============================
      (L2)===========================
      Description:Typographic Tool
      ==============================
            (L3)===========================
            Description:Avg. chars per line
            ==============================
            (L3)===========================
            Description:Avg. spaces per line
            ==============================
            (L3)===========================
            Description:% of blank lines
            ==============================
      (L2)===========================
      Description:ALL Dynamic Tests
      ==============================
            (L3)===========================
            Description:Test 1 -
            ==============================
                  (L4)===========================
                  Description:Test #1a
                  ==============================
            (L3)===========================
            Description:Test 2 -
            ==============================
                  (L4)===========================
                  Description:Test #2a
                  ==============================
            // ... Similarly for Test 3 to Test 5 ... ... //
      (L2)===========================
      Description:Feature Tool
      ==============================
```

As you can see, the report is organized into levels (or sections). The topmost level "General Grade" gives you the summary score for the whole lab. The subsections underneath give the detailed breakdown of the marking. You should concentrate on the two subsections "**`Compilation Tool`**" and "**`All Dynamic Tests`**" only. Ignore all the other sections such as **`Typographic Tool and Feature Tool.`**

The "**`Compilation Tool`**" section shows whether your program is free of compilation error and warning. Of course, if your program fails to compile, you will get pretty bad score as no tests can be run! Programs with warning are compilable, but most likely with wrong runtime behavior. So, make sure you nail down every single error and warning in your program before submitting.

The "**`Dynamic Test`**" is the most important section. As mentioned in the lecture, the score of this section constitute **70%** of the overall lab score. Under this sections, there are a number of test cases (shown as "**`Test 1`**", "**`Test 2`**", …, etc). In most labs, you should see around 10 test cases. The number of correct test cases indicates the overall correctness of your program. Although take-home labs are not graded, you should still pay attention to the dynamic tests, which give you an idea of the overall quality of the solution.

One common frustration from previous semesters is that the test cases **were not given,** i.e. you only know that the program failed on some test cases, but have no idea **what are those** test cases! To reduce the "**`hair pulling`**" stress level, the test cases will **now be provided for every labs (yes, even during sit-in labs)**. Before you celebrates ☺, please note that only ~80% of the test cases will be given.

2.4 Using the Sample Test Cases

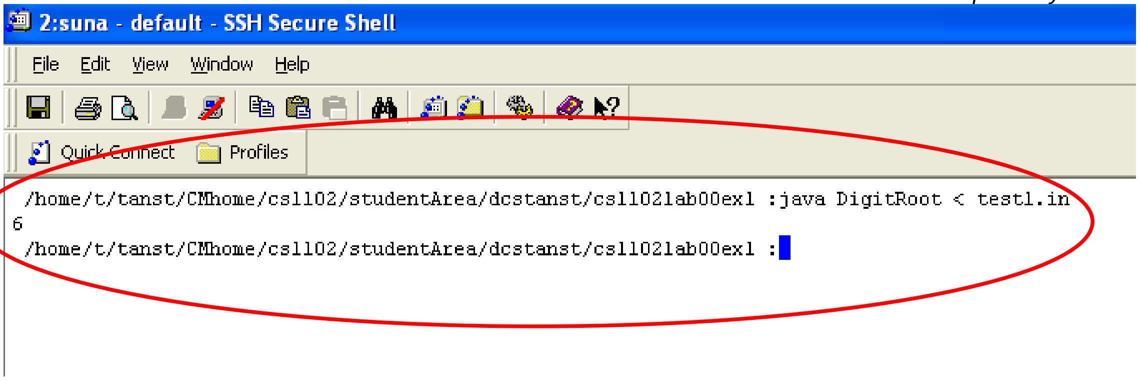
Let's exit CourseMarker and go back to the directory that contains your program file and executable.

You should have noticed the "**`test1.in`**", "**`test3.out`**", etc files in the exercise directory by now. These files contain the input and output used by the CourseMarker for dynamic tests. Files with the extension "**`.in`**" are the test inputs. Files with "**`.out`**" extension are the corresponding correct output. Each test case is numbered. So, "**`test1.in`**" is the input for test case 1, and the correct output is stored in "**`test1.out`**".

You can view the content of these files using the text viewing command. Enter "**`less test1.in`**" to view the first test case, you should see that the first test case is "**`24`**" (Remember to use '**`q`**' to end the "**`less`**" command). Try entering "**`less test1.out`**" to see the corresponding output, which is shown as "**`6`**".

You can then try starting up your executable "**`java DigitRoot`**" and enter "**`24`**" as input. If your output is "**`6`**", then your program is correct for this particular test case.

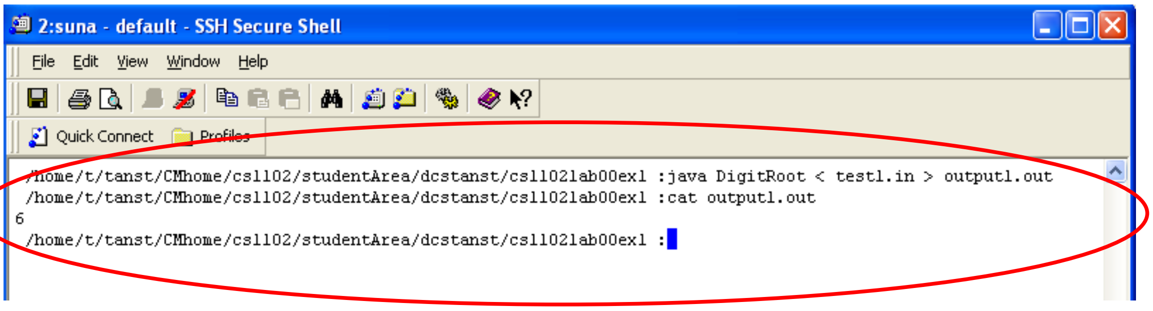
Typing in test cases by hand is pretty slow and error prone (you may mistype the input). The input can consists of tens (even hundreds ☺) of lines in subsequent labs. So, manual input is definitely not feasible. Fortunately, there is a way to "trick" your executable into reading a file *as if it is reading from the keyboard*. Try entering "**`java DigitRoot < test1.in`**", you should see the output "**`6`**" *without you typing in the input!*


2:suna - default - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
 /home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :java DigitRoot < test1.in
6
 /home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :
```

This "trick" is known as *redirection* in unix. Now we have a simple way to run various tests, try using the redirection with "**`test3.in`**". It is a lot more efficient now, isn't it? However, you still need to manually check the output against various "**`.out`**" file, which is again tedious. Let us show you another way to get around this. The *redirection* trick mentioned above works equally well for **output of a program**. You can capture all the output in a file instead of showing them to monitor.

Try typing in "**`java DigitRoot < test1.in > output1.out`**". This line basically says "execute **`DigitRoo`**, read input from **`test1.in`**, and place all output in **`output1.out`**". Pay attention to the direction of the arrow "**<**" and "**>**", and make sure you use a new output file name instead of **`test1.out`**. You should find that the executable seems to end without waiting for input and without showing any output! This is normal as all inputs are read from "**`test1.in`**", while all outputs are captured in the file "**`output1.out`**". Note that "cat **`output1.out`**" is another unix command to display the content of a text file on the screen. Here we use it to see the output produced by your program.


2:suna - default - SSH Secure Shell
File Edit View Window Help
Quick Connect
Profiles

```
 /home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :java DigitRoot < test1.in > output1.out
 /home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :cat output1.out
6
 /home/t/tanst/CMhome/cs1102/studentArea/dcstanst/cs1102lab00ex1 :
```

Important Note:

Be careful of the output file name when you redirect the output of a program. Redirection will destroy the original content of the file!

Note that if you run the command "**`java DigitRoot < test1.in > output1.out`**" again, you will get an error as unix disallows you to redirect output to an existing file. Simply remove the output file ("**`output1.out`**" in this case) by the command "**`rm output1.out`**" if you want to rerun tests.

Since we have now captured the output in a file "**`output1.out`**", we can use a simple unix command to compare it against the expected output. Enter "**`diff output1.out test1.out`**". The "**`diff`**" unix command compares two files (**`output1.out`** against the expected output in `test1.out`) and reports any discrepancy! If you see nothing on the screen (remember the saying "no news is good news"), then you can rest assured that your program works **perfectly for test case 1**. If "diff" spotted some discrepancies between the two files, those offending lines will be shown on screen. Let say your program is incorrect and produced the result "9" instead of "6", you will see something like this when "diff" is performed:

```
1c1
< 9
---
> 6
```

The "`diff`" command shows you the corresponding line from the two files so that the difference can be clearly seen. lines that start with "<" came from the 1st file (in this case `output1.out`), while lines start with ">" came from the 2nd file (`test1.out`). You should use this information to figure out the possible error in your logic and correct them. Then rerun this test to make sure there is no difference in the output.

Use the same idea for the rest of the test cases. In particular, check the assessment report and concentrate on those test cases that failed the dynamic tests on CourseMarker.

A summary of efficient testing is given below:

1. Make sure you have the executable ready
2. Execute the executable by
   `Java something < testX.in > outputX.out`
   where X is the test case number
3. Compare the output with the expected answer
   `diff outputX.out testX.out`
4. If there is no difference, proceed to another test case. If the two output files do not match, then you have a bug in your Java program. Find the bug, squash it, and retest.

For exercise 1, you **must get at least 6 test cases correct** to get the 2% assigned for the lab.

Important Note:

Remember that course marker always resume at the level you have last used it. So, remember to double check whether you are in the right exercise/lab in future before you attempt any command!

Hope you find this guide helpful! Happy coding!

Acknowledgement

Mr.Hugh Anderson's guide on CourseMarker is a great help for writing this document.
Dr. Soo Yuen Jien has amended it for CS1102C and I have adopted it for CS1102.