Imperial College of Science, Technology and Medicine	University of London
Computer Science (CS) / Software Engineering (SE)	BEng and MEng Examinations Part I
Department of Computing	Integrated Laboratory Course
Laboratory work is a continuously assessed part of the examinations and is a required part of the degree assessment. Laboratory work must be handed in for marking by the due date. Late submissions may not be marked.	

Exercise: 1	Working: Individual
Title: Introduction to Linux and Haskell	
Issue date: 6th Oct 2003	Due date: 13th Oct 2003
System: Linux	Language: Haskell

Aims

- The objectives of this first exercise are to gain some familiarity with the **Linux** operating system, the screen editor **nedit** and the language system **Haskell**. You will be writing your first five programs in the **functional** programming language **Haskell**, and you will be working in the **Linux** programing environment using the **nedit** editor for much of the first year.
- Although this exercise is not assessed, you should still submit it.

The Problem

- In this exercise you should write the following functions in a *script* which should be called **Quadratic.hs**. After you have completed and tested your functions so that you are certain that they work according to their specifications, you should **submit** the script in the way described in the section on Submission below.

  The functions that you should complete are the following:

  - **quad** which evaluates quadratic equations.
  - **quadIsZero** which determines if a quadratic equation evaluates to zero.
  - **quadraticSolver** which returns the two roots of a quadratic equation in a tuple, provided they are real.
  - **realRoots** which determines whether a quadratic equation has real roots.
  - **bigger** which returns the larger of two integers.
  - **smaller** which returns the smaller of two integers.

- **biggestOf3** which returns the largest of three integers.
- **smallestOf3** which returns the smallest of three integers.
- **isADigit** which determines whether a character is a digit.
- **isAlphabetic** which determines whether a character is an alpahabetic character.
- **digitCharToInt** which converts a digit character to an integer.
- **toUpperCase** which given an alphabetic character returns the corresponding upper case character.

Submit by Monday 13th Oct 2003

What To Do

You will find it useful to have your "Lab Info Pack" handy when doing this exercise.

- **Log in** – type in your name and your password which was given on the form you received from CSG. There is a separate document in your Lab Info Pack covering how to log in. **Ask the lab helpers** if you have any difficulty.
- You may want to change your password using the **passwd** command. Your new password should be something easy to remember but not easy to guess. For security reasons you should change your password regularly, ideally at least once every three months.
- Enter the Haskell system, with **Quadratic.hs** as the current script. Type

  hugs Quadratic.hs

- You can now use the **:edit** command inside Haskell. This will enable you to edit your script, using the **nedit** editor. You will find the script is initially empty. Add the following definitions for quad and quadIsZero, so that they use a quadratic equation:

```
quad :: Int -> Int -> Int -> Int -> Int
-- Returns evaluated quadratic expression.
quad a b c  x
  = a * x^2  +  b * x  +  c

quadIsZero :: Int -> Int -> Int -> Int -> Bool
-- Returns True if a quadratic expression evaluates to zero; False otherwise
quadIsZero a b c x
  = quad a b c x == 0
```

  Note that lines beginning with −− are comments. You do not have to enter those exact comments, but **you should get into the habit now of commenting all your work**. Larger programs in particular should be liberally commented to show what they are supposed to do and to help yourself and others understand and maintain them.

  Now save the script by selecting **Exit** from the **File** menu. You do this by moving the cursor to the **File** item on the menu bar and pressing the left mouse button. This will

display a menu containing a number of file operations. Keep the mouse button pressed and move the cursor to the **Exit** item and then release the mouse button. You will then be asked if you want to save your changes. Select **Yes** and your script will be saved. You will then be returned to the Haskell prompt and you can test the functions you have written.

- The Haskell system spots that the script has been edited and checks it for errors; if there are any it reports them and gives the line numbers where the errors have been found. Use the **:edit** command again and correct any errors. You can use the **Go_To** option provided by the **Search** menu to go to a line although for a small script you may find it easier to use the arrow keys to move around in the script.

  The script **Quadratic.hs** now contains the **declarations** and **definitions** of two functions **quad** and **quadIsZero**. **quad** evaluates a quadratic equation, given the values of the coefficients and the variable, and **quadIsZero** tests the value of the quadratic equation and returns True if it is zero. Evaluate the functions **quad** and **quadIsZero** with a number of different values. Type

```
quad 1 0 2 (-1)
quad 0 2 0 4
quad 7 3 6 10
quadIsZero 0 0 0 3
quadIsZero 4 5 1 2
```

- Re-enter the editor and define a functions **quadraticSolver** which will compute the two roots of a given quadratic. This functions should be declared as follows:

```
quadraticSolver :: Float -> Float -> Float -> (Float,Float)
-- Returns the two roots of a quadratic equation with coefficients a, b, c
```

  These functions take arguments of type **Float** which are used in Haskell to represent numbers with fractional parts. For the function **quadraticSolver** you should use the formula:

$$x = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

  for finding the roots.

  Since $\sqrt{b^2 - 4ac}$ is both added and subtracted from $-b$ to give the roots of the equation you will have two expressions for the roots. Return both of these in a tuple of the type **(Float,Float)**.

  You can use the Haskell prelude function (functions defined in the prelude are often called *built-in* functions) **sqrt** to calculate the square roots.
  Notice that the expression $\sqrt{b^2 - 4ac}$ needs to be refered to twice to compute both roots. To save typing it out twice, name the expression using **(a)** a qualified expression (using 'let') and **(b)** using a 'where' clause. You will also notice that the expression $2a$ occurs twice. You could also choose to name this (try it), although you save little or nothing either in typing effort, clarity or program efficiency...

- Now define a funcion **realRoots** which will determine whether or not a given quadratic has real roots:

```
realRoots :: Float -> Float -> Float -> Bool
-- Returns True if the quadratic equation has real roots; False otherwise
realRoots a b c
  = b^2 - 4 * a * c >= 0.0
```

  For the function **realRoots** you can use the fact that if $b^2 - 4ac >= 0.0$ then the two roots are real.

- Exit the editor and test your functions.

- Now re-enter the editor and define the functions **bigger** and **smaller** which return the bigger and smaller of two integers respectively, or one of the integers if the integers are the same. These functions should be declared as follows:

```
bigger, smaller :: Int -> Int -> Int
```

  Using only these functions, define the functions **biggestOf3** and **smallestOf3** that return the biggest and smallest of three integers:

```
biggestOf3, smallestOf3 :: Int -> Int -> Int -> Int
```

  As an exercise try to define these functions using nested conditionals ('if' .. 'then' .. 'else' ..) **(a)** using just the operators >= and && **(b)** using just the operator >=. Notice how much easier it is to solve the problem in terms of higher-level functions like **bigger** and **smaller** rather than >= and &&. this is your first lesson in abstraction - remember this always!

- Exit the editor and test these functions. How many test cases do you need to check that the functions biggestOf3 and smallestOf3 are correctly defined?

- In addition to **Int**, **Float** and **Bool** Haskell provides the built in type **Char** for processing characters. Characters are represented in the computer using **ascii** numerical values, with each ascii value representing a character. Hence there is a distinction between the character '7' and the number 7. '7' + 6 will give a type error. Haskell provides two built in functions **ord** and **chr** for converting characters to numbers and vice versa so that `chr (ord 'a')` gives the result 'a'.

  The ascii numbers have been designed so that the encoding of the letters digits and letters 'a'..'z', 'A'..'Z' and '0'..'9' are contiguous, eg `ord 'a'` = 97, `ord 'b'` = 98 etc. You can use characters in boolean expressions in Haskell eg `'Z' > 'A'` gives the result True

  Write two functions:

```
isADigit :: Char -> Bool
-- Returns True if the character represents a digit '0'..'9'; False
```

```
-- otherwise
isAlphabetic :: Char -> Bool
-- Returns True if the character represents an alphabetic
-- character either in the range 'a'..'z' or in the range 'A'..'Z';
```

- Finally write two functions:

```
digitCharToInt :: Char -> Int
-- Returns the integer [0..9] corresponding to the given character.

toUpperCase :: Char -> Char
-- Returns the upper case character corresponding to the input.
-- Uses guards by way of variety.
```

  To write the above functions you will need to use **ord** and **chr** for type conversions. Hint: `ord '7' - ord  '0' == 7` gives the result True

- Leave the Haskell system using the **:quit** command.

- You can now print a copy of your file *Quadratic.hs* on the self-service line printer in the terminal room (Room 219) by typing:

  lpr Quadratic.hs

  Please make sure you follow the instructions given on the printer when you print a file. If you have any difficulties please ask one of the demonstrators.

Submission

- Before submitting your program which should be in the file **Quadratic.hs**, make sure that your functions are called by the names given above. **Test your program thoroughly before submission as some marks are awarded on the basis of automatic test runs of your submitted labwork**. This first exercise is **unassessed**, but get into the habit of testing your programs now! You should test your program on a wide range of inputs to make sure it works as specified. Your PPT will be entitled to deduct marks from programs with errors, especially if those errors could have been detected easily by simple tests.

- To submit your work type

  submit 1

  while you are in the same directory as the files you are submitting. You should receive confirmation by e-mail that your work was received successfully. This will usually arrive in a few minutes. (You do not have to wait around for mail, it will be saved in your mailbox even if you have logged out.) Details about how to read your e-mail are given in the **Lab Info Pack**, or the on-line documentation provided by CSG. **Note** that you

can submit again if you want (so long as it's still in by the due date), for example if you realise you need to correct something. A later submission will overwrite and supersede any earlier ones.

Your submitted labwork will be forwarded to your PPT who will mark it and return it to you in the next PPT-session.

Note: Your PPT may require that you submit a printout of your program to the Student General Office (Room 345) in addition to the automatic submission. You should find out from your PPT if they want you to hand in a printout of your program.

- Remember to **log out** before leaving the lab. See the **Lab Info Pack** for details. If you have any problems then please **ask the lab helpers** to help you log out of the system.

Assessment

```
Although this first exercise is unassessed
you should still submit it.
```