

Practical 1

Theme

- Introduction to functional programming in Lisp and Haskell
- Dive-in taster for a variety of programming languages
- Running programs in different programming languages
- Making small changes to the programs


Key concepts: PL syntax, functional programming, compiled vs interpreted PL, types and type checking


1.1. Start up and essential configuration

a) Login to Ubuntu Linux

It is assumed you use the supplied virtual machine (VM) image or equivalent. The MB370/372 labs have a compatible Ubuntu installation.

b) Open this sheet and the provided Linux lab cheatsheet via Blackboard.

 **Hint.** You can *cut and paste* some commands and program extracts from the PDF document. Remember that in most Unix applications, you can cut and paste very conveniently by selecting the text using the mouse, moving the mouse over the target location and clicking the mouse middle button (or wheel). Unfortunately, some characters from this PDF document cannot be pasted correctly, most notably & and the single quote. These need to be overwritten by the correct characters after pasting.

 **Danger.** If you scroll using the mouse scroll wheel, take care not to press the wheel as this will paste the most recently selected text. If unnoticed, such pasting has the potential to significantly corrupt your program files. If it happens and you notice it immediately, the best thing to do is to undo the last change in the text editor using `Ctrl-Z`.

c) Start a terminal window.

You can start a terminal by pressing the terminal icon on the left of the screen.

d) Get familiar with the Unix terminal shell (if not yet).

The terminal starts up running *bash*, a command-driven interface (so-called “shell”), which enables you to launch programs by typing their names followed by parameters (traditionally called “arguments”) separated by spaces.

The cheatsheet includes a summary of some basic common *bash* commands.

It is recommended that you practise using *bash*, repeatedly executing commands such as the following:

```
> mkdir <folder>
> nautilus . &
> gedit <folder>/<file> &
```

e) *Create a folder for this module.*


From now on, it will be assumed that you use a folder called `PLC2130` inside your home folder for your PLC work.

(On the provided VM image this folder already exists.)

f) *Download from Blackboard and extract the archive `lab1-circle.zip`.*

Open the archive with the default application and extract its complete contents, preserving included folders. The archive contains a folder `lab1-circle` with 6 sub-folders, one per programming language.

In this practical you will be working mainly with Lisp and Haskell files.

 **Hint.** The cheatsheet contains instructions how to execute the given programs.


1.2. Basics of Lisp

All code extracts in this section come from the file `lisp/tasks.lisp`.

a) *Fix a common mistake in a simple Lisp expression.*

```
2 (print (1 + 2))
```


The fixed code has to contain all the characters of the original code, ie you should not replace `(1 + 2)` with `3`.

 **Quiz.** Answer **quiz question 1** related to this task.

b) *Write a Lisp if-then-else.*

```
6 (format t "type something: ")
7 (setf input (read-line))
8 (format t "you typed: ~a" input)
```


Modify the expression in on line 8. After modification, the program should print “nothing typed” if the input is empty, otherwise behave in the same way as before the modification. (Use function `string=` to compare two strings.)

 **Quiz.** Answer **quiz question 2** related to this task.

c) *Execute a Lisp program which has been stored in a variable.*

```
11 (let ((prg ' (+ 1 n))) (print prg))
```

This code currently prints the program stored in variable `prg`. Add further code within the `let`-expression so that the program `prg` is also executed with `n = 1`.

 **Quiz.** Answer **quiz question 3** related to this task.

1.3. Basics of Haskell

All code extracts in this section come from the file `haskell/Tasks.hs`.

a) *Identify and fix a mistake in the following simple Haskell program:*

```
4 sayHello =  
5   do  
6   let name = "Alice"  
7     putStrLn "hello"  
8   putStrLn name
```

 **Hint.** The mistake is purely of syntactical nature. It should be very easy to fix.

b) *Play with a recursive loop.*

```
12 ask :: String -> IO ()  
13 ask prompt =  
14   do  
15   putStrLn prompt  
16   line <- getLine  
17   if line == ""  
18     then ask prompt  
19     else putStrLn ("you said: " ++ reverse line)
```


(i) First, compile and execute the program using the command:

`runhaskell Tasks.hs`

(Make sure you `cd` to the `haskell` folder first.)

You should be prompted to “say something” and if you do, it is echoed. If you press `Enter` without typing anything, the prompt will repeat.


(ii) Alter the above code so that when someone presses enter without typing anything, the repeated prompt will say “please try again”.

 **Quiz.** Answer **quiz question 4** related to this task.


(iii) Undo the changes made in task (ii) and then alter the code so that when someone presses enter without typing anything, the repeated prompt will have an extra “!” at the end. With each further repeat, there should be an additional “!”.

 **Quiz.** Answer **quiz question 5** related to this task.

(iv) On line 18 replace `ask prompt` with `ask (putStrLn prompt)`. This will lead to a compiler error. Read the error carefully.

 **Quiz.** Answer **quiz question 6** related to this task.


(v) Answer the question: What would happen if an analogous change as in task (iv) was made in a Lisp translation of the code?

 **Quiz.** Answer **quiz question 7** related to this task.

1.4. (Recommended) Play with Java, Python, Ada, Lisp, Haskell, Elm programs

Execute and read the “circle” programs, then make small changes, such as:

- a) Make the programs draw not just one arc, but 2 arcs, the new arc being half-size of the original arc.
- b) Make the program draw many arcs, each approximately $3/4$ the size of the previous one. Beware that you do not attempt to draw infinitely many arcs!
- c) Give a nice error message when the program is called without the required numerical parameter.

 **Hint.** In all cases, the program is executed with one numerical parameter, eg:
`clisp circle.lisp 20.`