CP Lab 1: Introduction to DICE and the C compiler

Instructions

The main purpose of this Lab is to teach you to become familiar with the DICE environment – this is a Linux-based system hosted by the School of Informatics. All Labs for CP will take place on the DICE machines in AT5.05 (Thursday and Tuesday) or AT6.06 (Monday). For working alone, you may use any of the DICE labs in AT, as long as they are not booked out by a class. You have 24-hour access to AT levels 3,4,5: if your card/pin does not allow you in, use the ITO contact form on our course webpage to contact the ITO.

For this first Lab, *programming is not our main goal*. Our main goal is to get you familiar with the system.

In the course of working through this sheet, you will create and modify some files.

This lab takes place on the DICE machines in AT5.05/6.06 on Thu 21 Sep, Mon 25 Sep and Tue 26 Sep.

If you want to continue the work in your own time, please use the machines in any free AT lab.

Aims

This Lab is intended to introduce you to the computing environment (a Linux-based environment called DICE) provided for this course. You will need to understand *everything* in this Lab sheet in order to complete the future labs in weeks 2–10, and to do a good job with your Coursework assignment.

You will not find any material on DICE, or much on Linux, in A Book on C. *It is auxiliary material.*

Prerequisites

- Make sure you have your password for the DICE system. You should have been sent an email to your University account about this, if you were enrolled on the course in good time.
- If you did not get an email about setting your password for your School of Informatics account, talk to Computing Support who can be found in AT4.11 between 14:00–16:00 during semester-time (and also during our initial lab slots).

Notes

1. You need to understand all the material in this practical to complete the rest of the course. If you don't understand something *please ask the course lecturer and/or demonstrator*.
2. For your own personal practice at programming, or for finishing Lab sheets, use the workstations in the labs in Appleton Tower, any time except when these are being used for a course.
3. Don't be afraid to try experimenting to see if you can work out what is happening.

Keys and characters

Punctuation soup

The C language and the Unix command line use many punctuation characters that are not much used in everyday computing – you will need to find them on the keyboard! Also, we will want to talk about them. Here is a list of all the punctuation characters, and some names we might use for them, moving along the rows of a standard UK PC keyboard:

`!`	pling, bang, shriek	`;`	semi-colon
`"`	double-quote	`:`	colon
`$`	dollar	`'`	(single-)quote
`%`	percent	`@`	at-sign
`^`	caret, uparrow	`#`	hash
`&`	ampersand	`~`	tilde, twiddle
`*`	star	`\`	backslash
`()`	left/right round bracket, paren, bracket	`|`	pipe, bar
`-`	minus	`,`	comma
`_`	underscore	`<`	less-than
`=`	equals	`.`	full stop
`+`	plus	`>`	greater-than
`[]`	left/right square bracket	`/`	slash
`{}`	left/right curly bracket, brace	`?`	query, question-mark

You may have noticed that we missed out the `£` key. That's because the `£` sign is not in the original set of characters used by the computers for which C was developed. You will find two or three other characters on your keyboard which we also haven't mentioned, for the same reason.

The Return key, and Control keys

The Enter or ↲ key is traditionally called the Return key in Unix. As well as starting a new line when you are typing text, it has the important function of telling the *command line* to actually do the command you have just typed. We will remind you to press this key by writing, for example,

```
cd↲
```

If we write Ctrl-X, we mean 'press the X key while holding down the Ctrl key'.

Confusable characters

Some letters and numbers are very similar, but of course you need to use the right one. You will get used to the way they look in different fonts – here are examples of lower-case l (L), upper-case I, digit one, upper-case O and digit zero in the fonts you will see most often.

In these labsheets and on the slides, they are:

```
lower-case l, upper-case I, digit 1; upper-case O, digit 0
```

In the standard font on DICE computers, they look like:

```
lower-case l, upper-case I, digit 1; upper-case O, digit 0
```

Stage A Logging in and out

Preparation for logging in (username/password)

If this is your first time logging in, you must be sure you have already set an initial password for your School of Informatics account. This initial password would have been set by you, using the `pp.inf.ed.ac.uk` weblink, while logged in to another system. Note that your School of Informatics account is not the same as your overall University account, even though the *username* will be your student id (this is the same username as for the University account).

By now, you have probably already been sent an email which instructed you to set your DICE (School of Informatics) password, and which pointed you to the weblink above.

If you have already set this initial password, you are ready to log in now; if not, you need to first get your initial password sorted out, using your smartphone or with computing support in AT4.11, before you can log in. Note that if you have not installed the University's root certificate in your phone, you may get a security warning when you connect to `pp.inf.ed.ac.uk`. It is safe to add an exception, although if you want to use University secure sites, it's a good idea to install the University's certificate – see instructions pointed to from the course homepage. *(We believe this last no longer applies, but are leaving the sentence here just in case!)*

In choosing a password (whether your initial password, or choosing a different password at a later date) always choose a password *easy for yourself to remember, but hard for others to guess*. Passwords must be at least 8 characters long, and must contain at least three out of upper-case letters, lower-case letters, numbers, and punctuation.

Logging in

Find yourself a free computer (if it is asleep, ↲ will bring up the login window).

The login screen contains a small window with two boxes. Type your username in the top box, and your password in the bottom box. *Be careful not to type a space before or after your username or password.* Your username is your student id (the letter `s` followed by your matriculation number). You have already set an initial password as explained above. After typing your username and password, press ↲ or click the 'Login' button.

Assuming your username/password is correct, a simple popup will appear asking you whether you want to register this session as a lab attendance. If you click 'Yes', we will record the machine and time of login for attendance records. If you click 'No', nothing will be recorded. If you click 'Always record', you won't be asked again.

Then after a short wait the system will display the default Desktop for DICE – you will see some *icons* and possibly some open *windows*. The windows you see will depend on what programs were running when you logged out previously – so you won't see any on your first login.

The desktop will always include the following:

- the **Panel** along the top of the screen with menus for Applications, Places, and mayb with individual icons for the Firefox Browser and some other applications. Exactly what you see depends on the release of the software current when you first log in.
- a bar along the bottom of the screen (showing tabs for any open applications).

This environment (which arranges the windows and toolbars on the Desktop) is a Linux-based environment called Gnome™. Linux is a Unix-like operating system, and that is why you have been given a sheet of Unix commands.

Changing your password

You might not want to change your password yet – after all, if you chose the initial password yourself, it will be the one you want. However, to change your password *at a later time*, you can use the `passwd` command from the "command line'of a terminal window. Just type `passwd` in the terminal window (of any open terminal) and reply to the prompts.

You can change your password as often as you wish. Don't write it down! (The password is not stored on our network. Only an *encrypted* form is stored. When you login, the password you type is encrypted, and the result is compared with the stored version.

Logging out

The **Logout** option can be found by clicking the 'Power' symbol at the top right of the top panel, and clicking your name.

What you should now know

- ☐ How to set your initial password;
- ☐ How to log in to the Informatics system;
- ☐ How to change your password later (if necessary);
- ☐ How to log out from the Informatics system.

Make sure that you know how to do the above tasks (you can tick the boxes when you are sure that you can). Ask someone if you have any problems.

The most likely problem you would have in these early weeks is that your password might not work. That may be due to accidentally having the `Caps Lock` or `Num Lock` switched on, or mistaking the letter 'O' for the digit '0', or something similar. In rarer cases, it may be that there is a problem with your password. In the latter case, contact Computing Support.

Always log out before leaving the lab.

Stage B Web Resources

In this stage you will check out and bookmark some key web resources which will be helpful for CP.

The default browser for DICE is Mozilla Firefox, available on the top panel of the Desktop or via Applications→Favourites. (You should accept conditions of use if asked.) DICE also supports the Google Chrome browser, if you prefer that.

Preparation

The same regulations and policies apply to Informatics systems as to other University systems. In short: people wanting to do work have priority, and don't use our systems to access illegal or offensive material.

Resources

Now look for the School of Informatics "Information for new taught students' webpage. You can find this page by directly typing the webpage URL into the Location window that appears under the top toolbar of your browser, this address being:

`http://computing.help.inf.ed.ac.uk/new-taught-students`

There are many useful links on this webpage, relating to issues such as *details of computing facilities*, *getting help from computing support*, *details of printers and the printing system*, and information on our in-house system DICE (this subpage even includes an *alternative tutorial on using DICE*).

There are a collection of other webpages which will be useful at various times during CP. You should take a look at each of the following pages, and bookmark them:

- The CP teaching webpage **(the most important webpage of all)** will contain all lecture notes, tutorial sheets and practicals as they are released. It also often has important announcements.

  `http://www.inf.ed.ac.uk/teaching/courses/cp`

- The Informatics Teaching Organisation (ITO), which has offices on Level 1 of Forrest Hill, performs the administration of all Informatics courses. Their webpage is at the following address:

  `http://web.inf.ed.ac.uk/infweb/student-services/ito`

  The links from that page that you are most likely to use at some time are:

  - The `Contact us` button, which leads to a page including a form for making Internal Enquiries (best way to contact the ITO).
  - The `FAQ for students` link near the bottom of the page.
  - The `Computing Support form` link near the bottom of the page.

- Computing Support are the group of people who maintain and support the DICE network. Their webpage is at the following address:

  `http://computing.help.inf.ed.ac.uk/`

  The link on that page which will be of most importance is the `Frequently Asked Questions`, which contains answers to some common questions asked by computing users. In the case of faulty equipment, you may use the `Computing Support Form` to contact support. Remember that support do not help with any C Programming questions, as that is not their role.

What you should now know

☐ Where the various key webpages important for CP can be found.

Make sure that you have all the aforementioned webpages and links bookmarked: they will be very useful during the coming semester.

Stage C Terminals

In this stage we will discuss the use of windows on DICE, with special reference to Terminal windows.

The default way to bring up a terminal window is to follow

`Applications -> Favourites -> MATE Terminal`

from the top left-hand menu.

Clicking on `MATE Terminal` will launch a terminal emulator, also known as a *shell window* or a *console*. You will see that the 'prompt' of the terminal window is the name of the machine you are using, in brackets, followed by your own username and a colon.

In CP, we will be running our C programs via the command-line of a terminal window. We will need to use Unix commands to interact with the system via the command-line. This is why we have distributed a copy of a sheet with basic Unix commands.

Unix commands

The first thing to know is that the word 'directory' means the same as 'folder' in Windows and other visual operating platforms. Now we discuss Unix/Linux commands.

Go to the the newly-opened Terminal window and try out some (Unix) commands in the Terminal window:

(i) Type `ls` ↲ to see which files are in the current directory (by default, this will be your personal *home directory*). This will show just a few standard directories (since you have not created any files yet).

`ls` *can be remembered as short-for-'list'.*

(ii) Type `ls -a -l` ↲ to see all files, both visible and *hidden* configuration files. This time you will definitely see some filenames (together with the date of most recent edit), though it is unlikely these are files you will ever want to edit directly.

(iii) Type `mkdir lab1` ↲ to create a directory for this initial practical. To make sure this has succeeded, use `ls` ↲ again (to get extra information, use `ls -l` ↲).

(iv) Note that you are *still* located in your original home directory. Type `cd lab1` ↲ to move into the newly-created directory. From this location, you can add new files or sub-directories.

`cd` *can be remembered as 'change directory'.*

(v) To ever return to your personal home directory, just type `cd ~` ↲ at the prompt. (You can also just type `cd` ↲)

`~` *denotes your home directory always*

(vi) More generally (than just returning 'home') whenever you have entered a sub-directory, you can go 'back up a level' by typing `cd ..` ↲

(vii) To check which directory you are in, type `pwd` ↲ ('print working directory' – 'working directory' is another name for 'current directory').

Apart from the basic commands mentioned above, please test some of the other commands on the Unix sheet.

What you should now know

☐ how to use simple Unix commands for files/directories in a Terminal window.

Stage D Using the `emacs` editor

In this stage you will find out

- how to use the editor to view and modify files;
- how to print files;
- how to create files.

Starting the editor

To start the editor, use the terminal window. Make sure you that you are in the `lab1` directory (so then your files will end up in the right place). Think of a filename *filename* for the file you want to edit (it might be a file that is already in your current directory, or alternatively the name for a new file you want to create from scratch). (As a general rule, things in `typewriter` font are things you should type exactly as written, whereas things in *italic* are placeholders for something you choose.)

Note that while Unix filenames *can* contain any character (except `/`), the command line uses many punctuation characters for its own purposes. To avoid confusion, you should only use letters, digits, and the three punctuation characters `.-_` in filenames. In particular, don't use spaces. Also, Unix filenames are *case-sensitive* (unlike Windows): `Question` is a different file from `question`.

Then type

`emacs` *filename* `&`↲

(with your chosen name for *filename*) in the terminal window. The system will load up an `emacs` window onto the Desktop. The `&` after `emacs` is an instruction to the shell that the `emacs` command should be run in the background, leaving the command line free for *further commands*.

In some cases (depending on the exact method you use to start `emacs`*), you will be presented with a 'splash screen' with some text when the* `emacs` *window loads. If this happens, hitting the* 'q' *key will remove this.* At a later stage, try using `emacs` without a *filename* argument and you will see the difference in what happens.

As an example, suppose you already had a text file called `runresult` sitting in your current directory. Then typing the following at the command line will open that file for editing:

```
emacs runresult & ↲
```


If you did not have a file called `runresult` yet, then a new file will be created with that name (assuming you go on to actually save some text in that proposed file).

In opening files this way, you must be in the directory which contains (or is going to contain, if the file is new) that file. To check you are in the right directory, type `ls`.

There is an alternative way to start up the `emacs` editor. You can use the bar at the top of the desktop and click on the sequence `Applications` → `Accessories` → `Emacs`. This will load up the same editor.

If you use this alternative option, please take care in saving your files – you will need to make sure you select the correct folder under `File -> Save As ...` to ensure it is saved in the correct sub-directory of your filespace.

Using the editor

The creation and editing of files is an activity which occupies a sizeable fraction of a computer user's time. This lab gives you an opportunity to learn your way around the `emacs` editor provided on DICE. It is worth taking some time over this: proficiency in the use of the editor will pay dividends later on in the course.

In this part of the lab you will use the mouse and menus to issue commands to the editor. The editor menus appear just underneath the title bar at the top of the `emacs` window. The menu entries often have keyboard equivalents (key combinations or key sequences that achieve the same effect). If a keyboard equivalent exists for a menu entry, it is given in the right-hand column of the individual menus.

Saving a file

At the start of this Lab, you have no text files. So first thing to do is to save some text into a file.

A good name for your first text file is `questions` – you could use this to make a list of notes of things you have noticed during the Lab, which you need to further explore or query with the Lecturer/demonstrator.

After choosing a filename, and starting up `emacs` as described above, next type a couple of stray characters into the `emacs` window and then choose Save ... from the File menu to save this initial file. Alternatively, you could use the keystroke sequence Ctrl-X Ctrl-S to save your input.

Now type Ctrl-X Ctrl-C to shut down `emacs`.

Opening files

To open a different file from inside `emacs`, pull down the `File` menu near the top-left corner of the editor window, slide the pointer to `Open file` and release the mouse button. This will provide a windowed display of your user filespace (starting from your home directory), to allow you to select directories and files.

Note that the starting directory for loading `emacs` is important. :

- If you open `emacs` from the Terminal window while *inside* the sub-directory `lab1`, then you will see an icon for `questions` (and any other files in `lab1`) immediately;
- If you load `emacs` via the specified sequence from the `Applications` menu, the default directory will be your home directory. Hence you would need to enter the folder icon for `lab1` first, and only then you would see `questions`.

When a file is opened, it becomes a *buffer* in `emacs`. You can open several files at a time (they become several buffers) and use the Buffers menu to switch between them. If you find that you have too many buffers, you can delete buffers (i.e., 'close' a file) by selecting Close Buffer from the File menu.

Moving and searching in files

Now suppose you have `emacs` open and displaying the file `questions`. Start off by writing a list of the questions that you have at this point.

After entering a certain amount of text, save the file again. You can use File followed by Save, or alternatively use the Ctrl-X Ctrl-S shortcut.

After that, try moving around the file using the four *cursor* keys at the bottom right of the keyboard

Move back to the beginning of the file using the keys mentioned above and use Ctrl-S to find the first occurrence of the word 'lab' (this might, or might not, appear in your file). Try again with a different word – and make sure to try words in your file, not in your file, and also with duplicates.

Go to the beginning of the file again, and experiment to find out if the Ctrl-S command is affected by the *case* of the string you tell it to find. In other words, would it have made a difference if you had searched for `lAB` instead of `lab`?

The Edit menu also offers options for searching under its Search option. Try to explore these to see what is available.

Modifying a file

Take a look at the Edit menu. The Cut and Paste commands are there, as well as two related commands:
Copy

- lets you duplicate the selected region by making it available for pasting without cutting it.

Clear

- lets you remove the selected region by cutting it without making it available for pasting.

Notice how the menu items are originally greyed out: these commands are not available unless you have already highlighted a region.

Test these commands by putting the answer to one question in the wrong position initially, and then use these commands to move your answer below the appropriate question.

There's also a very useful Undo command in the Edit menu that will undo your last editor command (be careful with this one – but it can be very useful if you Clear something by mistake).

Printing your file

Often you will want to print out the files you are editing in `emacs`. Our student labs now use the University's Cloud printing service, and your default service is cloud-mono. There are mono printers at the back of the labs we use.

Note that these are charged printers on the standard university print-charging system, so it will cost you 5p per page.

To print your file from `emacs` choose Print from the File menu.

What you should now know

- ☐ How to open a file for editing
- ☐ How to save a file
- ☐ How to move about and search in a file
- ☐ How to cut and paste regions of a file
- ☐ How to print a file from `emacs`

Make sure that you know how to do the above tasks (you can tick the boxes when you are sure that you can): you will have difficulty with the rest of the course otherwise.

Stage E Poem program

This is the fun part!

The goal is for you to write a small variation on the 'Hello World' program which we discussed in Lectures 1 and 2 of this course. Your mission is as follows:

> You must write a short 4 or 5 line poem, somewhere containing your own name. Then you must create a small C-program named `mypoem.c` to print out the poem line-by-line.

Some initial comments:

- It will be helpful for you to have the lecture notes from lecture 2 at your side while you are doing this task.
- The poem does not have to be a masterpiece, just in your own words (and keep it appropriate).
- The program must print the poem out on a line-by-line basis.
- The program will not need to have any variables – it shares that feature with `hello.c`.
- You must check that your program compiles (using the `gcc` compiler), and also you must run the program to check the output is correct.

Step by step, here are your instructions.

(i) Create your poem before you make an attempt to write the C-program.

(ii) Open a new file `mypoem.c` in the `emacs` editor. To do this you just need to open `emacs` as described in Stage D, and then type a couple of stray characters, then do an initial saving (when saving, make sure to save it as `mypoem.c` within the `lab1` directory; ie, you should see `/lab1/mypoem.c` at the bottom of the `emacs` window before pressing ↵ to save).

(iii) The first stage of writing any C-program is getting the basic structure (header files, `main` etc) of the program correct. Therefore as your first step, consider the simplest ANSI C program imaginable, and edit `mypoem.c` to contain this program:

(iv) The header files of this very simple program (which does no processing and prints no output) are very limited. We will be using `printf`, so we will need to also include the `<stdio.h>` header file. Add the extra line for this and save again.

```
#include <stdlib.h>

int main(void)
{
   return EXIT_SUCCESS;
}
```

Figure 1: The simplest C program

(v) Design a series of `printf` commands which will print out your poem line-by-line, and add these inside the `main` function in the appropriate order. Once you think you have a correct program, save again and exit `emacs`.

(vi) Type `gcc -Wall mypoem.c` ↲ while you are in the directory which contains `mypoem.c`. This may give some warnings and/or errors to be corrected. If so, be proud that you have reached the DEBUGGING stage. If there are no errors in your program, you will see nothing! `gcc`, like most Unix commands, works on the principle that 'no news is good news' – if everything worked fine, there's no need to say anything.

(vii) Once you have debugged your code and created an `a.out` file in your directory, type `./a.out` ↲ to *run* your program!!! When you see the output, you will know whether your program needs extra modification.

(viii) When you are happy with your program, congratulations! Finally, in the directory with your program, type the command

`submit cp lab1 mypoem.c`

which will file a copy of your program with us, so that you (and we) can keep an eye on your progress through the labs.

What you should now know

- ☐ How to use the `printf` command to print formatted lines of text
- ☐ How to to use `gcc` to compile a C-program
- ☐ (maybe) How to interpret errors/warnings from the `gcc` output to help debug a program
- ☐ How to run the output of `gcc`
- ☐ Submit your program, *then* tick this final box.

Stage F Squaring program (Mon, Tues labs) **(optional)**

In Lecture 3 we finished the details of how to design the `square.c` program, to read an integer input from the user, square this number, and print the the squared value out to the user.

Can you create a file called `square.c` inside the `lab1` directory, and type in our 'squaring' program into it, taking care of the little syntactic details. Then carry out the steps (vi), (vii) and (viii) described in the discussion on `mypoem.c`, to debug the program with `gcc` and finally run and test (and further debug) the program.