CMSC 11 Lab Exercises, Weeks 0, 1 (Getting Started)

Hi! Welcome to the ICS Computer Labs. This lab will be your home for the next few months so please take good care of the room and its facilities. Your lab instructor should provide detailed lab policies but here are some guidelines. This will allow you and other students using this lab to work and learn more effectively.

1) Please do not eat, drink, litter, nor smoke in any of the labs.
2) Please do not change the hardware, software or network configuration of any of the lab computers without your instructor's permission. Should there be any malfunctioning units, please inform your instructor immediately so this can be reported to the technical support staff as soon as possible.

This week's goals

1) Introduce the paired programming scheme for lab exercises.
2) Start using the computer and be familiar with the Linux programming environment.
3) Use any easy text editor (e.g., kwrite or gedit) to create or edit sample C programs.
4) Compile sample programs using the C compiler (gcc) and run them on a command line terminal. Note and try to explain possible errors that may arise during program compilation or execution.

The Linux environment

Linux is an open-source operating system and most distributions come with a user-friendly graphical user interface so even absolute beginners will find it fairly straightforward to use. There are lots of application programs you can run in Linux and most applications that you may have previously used in Windows have counterparts that run under Linux. Some popular applications are shown below:

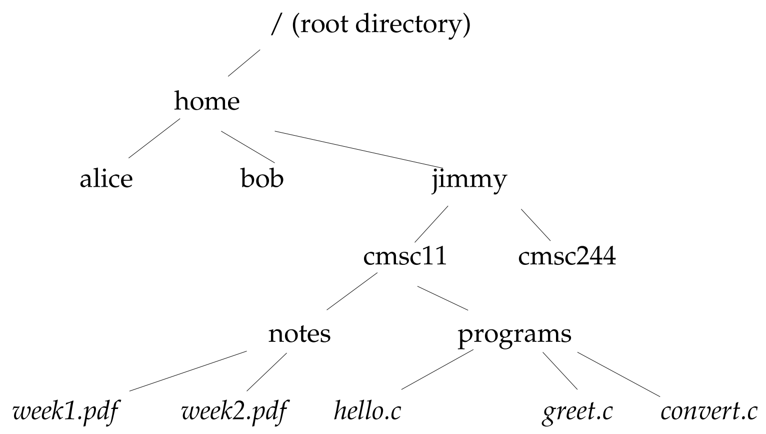
Function	*Sample applications in Linux*	*Windows equivalent*
Word processing	Open-Office Writer	Word
Spreadsheet calculations	Open-Office Calc	Excel
Presentation tools	Open-Office Impress	PowerPoint
Web browsing	Konqueror	Internet explorer

Your instructors may briefly demonstrate some of these basic and widely popular applications, although these are discussed in more depth in other introductory courses offered by the institute such as CMSC 1, CMSC 2, or IT 1.

File system basics

All documents, programs and data in your computer are stored as files. Because there are so many files involved, most operating systems group related files together logically into directories (also known as folders). This is a hierarchical structure where directories can have sub-directories, and sub-subdirectories, etc.

The example below shows multiple directories at several levels as well as several files.



A particular file is identified by its name and extension (e.g., hello.c) or a path of directories (e.g., /home/jimmy/cmsc11/programs/hello.c)

You can use a graphical user interface (GUI) to view the directory structure by clicking on the Home icon. This activates a browser such as konqueror, and you can use the arrows to navigate though the directories. You may also need to know a few basic commands given at a command line interface.

pwd	*- print the current/working directory*
ls *directory*	*- list the contents of a directory*
cat *filename*	*- view the contents of a file*
more *filename*	*- a filter to view a large file one screenful at a time*
cd *directory*	*- change to a directory* (**cd** *.. = move up to parent directory)*
man *command*	*- to know more about a specific command*
mkdir *directory*	*- create a new directory*
rm *filename*	*- remove/delete a file*

Editing, compiling and running a program
Most of the activities in CMSC 11 lab sessions will involve programming so it is important to learn the basics of this process.

1. *Create your own directory so as to avoid messing up other user's directories.* Then edit your program under your directory using any text editor. Full-screen editors are easier to use, such as kwrite or gedit. Save your program in a file (C programs have the extension .c) using a meaningful filename.

```
$ kwrite hello.c&          (invoke the text editor)
```

hello.c

```
/* Example 1.1 a traditional first program */

#include <stdio.h>

main()
{
  printf("hello world\n");
}
```

2. Compile the program using the gcc compiler. If there are no errors, an executable file named "hello" will be created and will be ready to execute.

```
$ gcc -o hello hello.c     - compile the program named
                             hello.c (this creates the executable file named hello)
$ ls                       - list the contents of the directory
hello  hello.c             - other files may be present

$ ./hello                  - run the executable program; the prefix
hello world                       ./ indicates the current directory
```

3. Everyone makes mistakes so you should not panic if something goes wrong and the program does not compile (or execute) properly. Below is a typical error message.

```
$ gcc -o hello hello.c
hello.c: In function `main':
hello.c:8: error: syntax error before '}' token
```

This one is a simple typing error committed by omitting the semicolon at the end of the print() statement in the program. Many other types of errors are of course possible. You should try to deliberately introduce a few errors in your program and see what happens when you compile an erroneous program. When such errors occur, edit the program and re-compile.

```
$ ./hello                         (run the program)
hello world
```

4. Note that some editors (e.g., kwrite) provide color-coding of keywords, program comments, string messages and matching braces to guide you in your editing.

   Error messages may also provide the line number where the error(s) occurred. In the kwrite editor, line numbers can be displayed by pressing/toggling the F11 key.

```
$ gcc -o hello hello.c
hello.c: In function `main':
hello.c:8: error: syntax error before '}' token
```

Some error messages, however, can be misleading so check everything before re-compiling.

5. A few more sample programs to edit, compile and run are given below. We will discuss programming details in the coming weeks but try to understand what these programs are doing by running them with different test data. Keep practicing -- the more you get familiar with the tools of programming now, the more comfortable you will be in using them for serious work later. Programming can be *fun* as well, but you need to invest significant amounts of time, effort, and patience.

greet.c

```
/* Example 1.2 another greeting with user input */

#include <stdio.h>
main()
{
  char username[20];
  int  age;
  printf("Hi. What is your name? ");
  scanf("%s", username);
  printf("Hi %s, how old are you? ", username);
  scanf("%d", &age);
  printf("Bye %s, see you next year when you are %d.\n",
  username, age+1);
  printf("Have fun being %d now!\n", age);
}
```

convert.c

```
/* Example 1.3 a conversion program with calculations */
#include <stdio.h>
main()
{
  float inches, cm;
  printf("Please enter your height in inches: ");
  scanf("%f", &inches);
  cm = 2.54 * inches;
  printf("%6.2f inches is equivalent to %8.1f cm.\n",
  inches, cm);
}
```