Lab Exercise 2: Compiling & running C; Input/Output

Duration: 1 session

Aims

To encourage you to find out about some of the basic features of the C language and the facilities that support their use.

Useful C on-line course themes etc.:

- Java to C
- Types, Operators, and Expressions (in Information Representation)
- Control Flow
- Input and Output

Learning outcomes

On successful completion of this exercise, a student will:

- Know how to compile and link C programs (using `make`) and then run them;
- Know how to use `man` to find out about C library functions, and the use of linker (`-l`) flags and `#include`s;
- Know a lot of features in C and Java that are similar in both languages;
- Know how to use simple input and output facilities in C.

Summary

- Compile and run two C programs whose code you are given, and write notes about it in a file `howto`
- Write a C program, `part2.c`, that uses:
  language features common to both Java and C, and
  simple C input and output facilities.

The unextended deadline is the end of your scheduled lab.
If you need it, if you attend the lab you will get an automatic extension to the begining of your next scheduled COMP26120 lab session (you must use submit to prove you finished in time and get it marked at the start of your next scheduled lab).
You can also get an extension for good reason e.g. medical problems.

Description

For this lab exercise you should do all your work in your `COMP26120/ex2` directory.

Part 1: Compiling and Running C Programs

For this part, you don't have to do much programming, but you do need to write notes about what you do for future reference in the file `howto`.

Copy the starting files into your `COMP26120/ex2` directory from `/opt/info/courses/COMP26120/problems/ex2` (use this path) These files are: `HelloWorld.c`, `SalaryAnalysis.c`, `SalaryAnalysis.java`, `salary-data.txt`, `makefile`, and `howto`

Step 1A: Hello World

Compile `HelloWorld.c` using the command (the > is the command-line prompt - you don't need to type it):

```
 > make HelloWorld
```

You should see a single line of output:

```
 gcc -g -std=c99 -Wall HelloWorld.c -o HelloWorld
```

Now run the resulting program using the command:

```
 > HelloWorld
```

(or, depending on your $PATH variable, you may need to use `./HelloWorld`)
Again, you should see a single line of output:

```
 Hello world!
```

You just used the `make` command, which gets information from your `makefile`, to compile a C program. `make` used `gcc` (the Gnu C Compiler) to create a binary program `HelloWorld`, which can then be run directly without the use of an interpreter (i.e. unlike Java programs, you don't have to use e.g. `java HelloWorld` to run your compiled program).

Use the `man` command (e.g. type `man gcc`) to find out about the command parameters `-g`, `-std=c99`, `-o` and `-Wall` that `make` gave to `gcc`. Write notes about what you discover in `howto`.

Edit `HelloWorld.c` to comment out the `#include` line and recompile it. Write down the error-messages you get, to help you identify this sort of problem in the future.

Use the `man` command to find out about `printf` - if you type `man printf` (try it!) you will get output starting: `PRINTF(1) User Commands PRINTF(1)` which tells you that this is not what you were looking for - instead, you need to type `man 3 printf` to get information about the function in the C library:
`PRINTF(3) Linux Programmer's Manual PRINTF(3)`
Before you proceed, write notes about what you have done in `howto` so that you will be able to cope with similar things in later exercises.

Step 1B: Salary Analysis

Compile `SalaryAnalysis.java` using `javac` and run it with the command:

```
java SalaryAnalysis salary-data.txt
```

to see the output you should expect from the C version of this program.

Compile `SalaryAnalysis.c` using `make` as in step 1A (i.e. `make SalaryAnalysis`). You should see an error message from `ld` saying:

```
"undefined reference to `lround'".
```

`gcc` has compiled the program without problems, but then called on the linker, `ld`, to add pre-compiled code from the libraries to your compiled program, and `ld` cannot find the library code for `lround`. Look at the synopsis section from:

```
man lround
```

The first line says what `#include` to use (check this in the C code), and the last line says what compiler and linker flags to use - you need to have a linker flag of `-lm`
To get this, edit the `makefile` to remove the # (the comment symbol) from the start of the line:

```
LDFLAGS=-lm
```

and use `make` again to recompile the program. Now run it using:

```
SalaryAnalysis salary-data.txt
```

and compare the output with that from the Java version. What differences are there?
Write notes about all of this in `howto`, to help you identify and deal with these sorts of problem in the future.

Part 2: C Input/Output

You should write a single program, `part2.c`, for all steps of this part. You should probably keep a working back-up each time you progress to a new step.

The steps become harder, with less hints, as you progress. If you are running out of time, don't try to complete all steps - instead get everything marked that you can by the deadline, and try to prepare better for the next lab exercise. (Ensure that you use submit to prove that you finished in time.)

Step 2A

Write a C program to read characters one-by-one from standard input (you can use ctrl-D to terminate the input), convert all upper-case characters to lower-case and all lower-case characters to upper-case, and write the result to standard output e.g.:

```
Hello World!
```

would become:

```
hELLO wORLD!
```

You should also count how many characters you have read, and how many of those you have converted in each direction, and output the totals at the end e.g.:

```
Read 13 characters in total, 8 converted to upper-case, 2 to lower-case
```

Hints:
You will need to use getchar and putchar for the individual characters, as well as printf for the final character counts.
You may want to use some of the functions in `ctype.h`, such as `tolower` and `isupper` (e.g. use `man ctype.h`).

Step 2B

Edit your program so that the input is read from a file opened from within your program. For the time being, you can use a fixed file-name such as `"input"`.
Hint: Stream Manipulation

Remember to check that the file is correctly opened. Test this by running the program when there is no input file available.
Hint: refer to `man fopen` and `SalaryAnalysis.c`

Step 2C

Edit your program so that the output is written to a file, using a fixed file-name such as `output`.

Remember to check that the file is correctly opened. Test this by running the program when a file of that name exists, but is write protected e.g. `chmod u-w output`

Step 2D

Edit your program so that, instead of using fixed filenames such as "input" and "output", both file-names are read from standard input when the program is run (i.e. use scanf or similar; I don't want you to use command-line parameters).

Marking Process

You must use `labprint` and `submit` as normal. They will look for `howto` and `part2.c`
The marks are awarded as follows:

Part 1:
2 marks - sensible notes in `howto` (including using gcc, make, man, ld etc., and using C libraries)
Part 2:
2 marks - Step 2A: upper-case and lower-case, and counts
2 marks - Step 2B: input from a file
2 marks - Step 2C: output to a file

2 marks - Step 2D: reading the file-names from standard input
Total 10