Systems Programming: Practical 7
Libraries and Makefiles

If you haven't finished Practical 6 yet, do that before attempting this one.

All these exercises need to be performed on a UNIX system or UNIX type system. You can use a MAC, a Linux distribution or remotely login to the university shared Linux system: `mira.dur.ac.uk` (there are instructions for this available on Ultra).

A Scalar product

A vector is an ordered collection of values and its dimension is the number of values it contains. In a file `scalar.c`, put a global variable with file scope `static int dim` that will keep track of what dimension vectors the program is currently using. Add a function `set_dim(int d)` to `scalar.c`, which sets `dim` to the value `d` and a function `get_dim()`, which returns the current value of `dim`.

The scalar product of two vectors is obtained by multiplying the elements entry by entry and then summing them up e.g. `(1,2,3)*(4,5,6)=1*4+2*5+3*6=4+10+18=32`.

Write a function `int scalar_product(int *v1, int *v2)` that finds the scalar product of the vectors `v1` and `v2` and returns the result.

Add a `main()` function, which:

1. Takes one command-line argument and sets the dimension to this value.
2. Asks the user to type in the values of two vectors of this dimension.
3. Uses `scalar_product()` to calculate the scalar product of the two vectors and prints this value out.

i.e. the output when run should look like this:

```
$ gcc -Wall -Wextra -o main scalar.c
$ ./main 3
Input first vector: 1 2 3
Input second vector: 4 5 6
The scalar product is 32
```

B Creating a Library

Take your code from Part A and move the `main()` function to a new file `main.c`. Create a `scalar.h` file and add suitable `#include`s so that `main()` can call the functions in `scalar.c`. If you compile with: `gcc -Wall -Wextra -o main main.c scalar.c`, then running `./main` should work as before.

Next:

1. Compile `scalar.c` as a static library `libscalar.a` and compile your executable by linking to this library statically. Check that the program still works.

2. Compile `scalar.c` as a dynamic library `libscalar.so` and compile your executable by linking to this library dynamically. You should find that your program only runs if `LD_LIBRARY_PATH` has been set appropriately.

C Write a Makefile

Download the files `input1.txt`, `input2.txt` and `input3.txt`. Create a `Makefile`, so that when `make` is run, the following happens:

1. The `tr` command is used to convert all letters in `input1.txt`, `input2.txt` and `input3.txt` to lower case and the output of this is written to `input1.out`, `input2.out` and `input3.out`, respectively.

2. The output of all three `.out` files is concatenated into a file `input.all`.

Apart from the `all` rule, there should be one rule for each file produced and commands should only be run if the corresponding input files have been modified.

D Optional – More on Makefiles

Modify your `Makefile` from Part C, such that it uses pattern rules to create the `.out` files form the `.txt` files in the current directory. Add a `clean` rule, which deletes all the generated files.