Week 09 Laboratory Exercises

Objectives

- Developing Perl skills
- Understand make and Makefiles

Preparation

Before the lab you should re-read the relevant lecture slides and their accompanying examples.

Getting Started

Create a new directory for this lab called `lab09`, change to this directory, and fetch the provided code for this week by running these commands:

```
$ mkdir lab09
$ cd lab09
$ 2041 fetch lab09
```

Or, if you're not working on CSE, you can download the provided code as a zip file or a tar file.

EXERCISE:

Make, Make, Make, Make, Make ...

A `Makefile` is a useful tool for documenting dependencies among components, as well as ensuring that executable components are up-to-date relative to their source code.

While in general automatically extract all dependencies for complex software systems, a tool that generate most of a Makefile can be very useful.

Your task in this exercise is to write a Perl script `makemake.pl` which generates an appropriate Makefile for a multi-file C program. This will require making a number of assumptions.

Your task

Write a Perl script `makemake.pl` to generate a `Makefile` that will build the C program whose source code is in the current directory.

The script takes no arguments, and writes the `Makefile` text to its standard output.

Assumptions

- The current directory contains all code files from only one C program.
- The C program is composed only of `.c` and `.h` files.
- Exactly one `.c` file will contain a `main()` function.
- The `main()` function appears on a line that matches the regexp

  ```
  /^\s*(int|void)\s*main\s*\(/
  ```

  No other line in the files matches this regex.
- The executable is named after the file containing the `main()` function (without the **`.c`**)
- Any line starting with `#include` is a genuine include line
  (ignore multi-line comments and other preprocessor commands like `#if`)
- All the dependencies that need to be included in the `Makefile` involve files that exist in the current directory

Example Output

Three example C programs are available as a zip file

Here is the output your program should produce for each of these examples:

```
$ unzip /web/cs2041/20T2/activities/makemake/examples.zip
Archive:  examples.zip
   creating: easy/
 extracting: easy/graphics.h
  inflating: easy/graphics.c
....
$ cd easy
$ ls
easymain.c  graphics.c  graphics.h  world.c  world.h
$ ../makemake.pl >Makefile
$ cat Makefile
# Makefile generated at Mon 27 Jul 17:24:51 AEST 2020

CC = gcc
CFLAGS = -Wall -g

easymain:   easymain.o graphics.o world.o
    $(CC) $(CFLAGS) -o $@ easymain.o graphics.o world.o

graphics.o: graphics.h world.h graphics.c
easymain.o: world.h graphics.h easymain.c
world.o:    world.h world.c
$ make
gcc -Wall -g   -c -o easymain.o easymain.c
gcc -Wall -g   -c -o graphics.o graphics.c
gcc -Wall -g   -c -o world.o world.c
gcc  -o easymain easymain.o graphics.o world.o
$ cd ../medium
$ ls
a.c  a.h  aaa.c  aaa.h  b.c  b.h  bb.c  bb.h  c.c  c.h  common.h  main.c
$ ../makemake.pl >Makefile
$ cat Makefile
# Makefile generated at Mon 27 Jul 17:26:43 AEST 2020

CC = gcc
CFLAGS = -Wall -g

main:   a.o aaa.o b.o bb.o c.o main.o
    $(CC) $(CFLAGS) -o $@ a.o aaa.o b.o bb.o c.o main.o

b.o:    b.h bb.h aaa.h b.c
c.o:    c.h c.c
a.o:    common.h a.h a.c
bb.o:   bb.h aaa.h bb.c
main.o: a.h c.h b.h bb.h aaa.h main.c
$ make
gcc -Wall -g   -c -o a.o a.c
gcc -Wall -g   -c -o aaa.o aaa.c
gcc -Wall -g   -c -o b.o b.c
gcc -Wall -g   -c -o bb.o bb.c
gcc -Wall -g   -c -o c.o c.c
gcc -Wall -g   -c -o main.o main.c
gcc  -o main a.o aaa.o b.o bb.o c.o main.o
$ cd ../hard
$ ls
circ1.c  circ1.h  circ2.c  circ2.h  circmain.c
$ ../makemake.pl >Makefile
$ cat Makefile
# Makefile generated at Mon 27 Jul 17:29:14 AEST 2020

CC = gcc
CFLAGS = -Wall -g

circmain:   circ1.o circ2.o circmain.o
    $(CC) $(CFLAGS) -o $@ circ1.o circ2.o circmain.o

circ1.o:    circ1.h circ2.h circ1.c
circmain.o: circ1.h circ2.h circmain.c
circ2.o:    circ2.h circ1.h circ2.c
$ make
gcc -Wall -g   -c -o circ1.o circ1.c
```

```
gcc -Wall -g   -c -o circ2.o circ2.c
gcc -Wall -g   -c -o circmain.o circmain.c
gcc  -o circmain circ1.o circ2.o circmain.o
```

Further Requirements

- You should include four standard lines at the front of every `Makefile`

```
# Makefile generated at timestamp in format given above

CC = gcc
CFLAGS = -Wall -g
```

- There should be a blank line immediately before each `Make` rule

Hints

- You could use a hash to represent all dependencies
- The Perl expression `` `date` `` will generate a suitable timestamp
- The Perl builtin `glob` is an easy way to get a list of the `.c` or `.h` files in the current directory
- You do not need to produce rules for dependencies on standard `.h` files (e.g. `stdio.h`)
- You can assumes includes of standard libraries use <...> notation, e.g:

```
#include $lt;stdio.h>
```

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest makemake
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab09_makemake makemake.pl
```

before **Tuesday 04 August 21:00** to obtain the marks for this lab exercise.

CHALLENGE EXERCISE:

What Can't Regexes Do?

Write a Perl regex which matches unary number iff it is composite (not prime).

In other words write a regex that matches a string of *n* ones iff *n* is composite.

A test program is provide to assist you in doing this:

```
#!/usr/bin/perl -w

die "Usage: $0 <min> <max> <regex>\n" if @ARGV != 3;

($min, $max, $regex) = @ARGV;

die "regex too large" if length($regex) > 80;

foreach $n ($min..$max) {
    $unary = 1 x $n;
    print "$n = $unary unary - ";
    if ($unary =~ $regex) {
        print "composite\n"
    } else {
        print "prime\n";
    }
}
```

Download test_regex_prime.pl, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/20T2/activities/regex_prime/test_regex_prime.pl .
```

For example to test the regex **`^1(11)+1$`** against the integers 2 to 12, you can run

```
$ chmod 755 test_regex_prime.pl
$ test_regex_prime.pl 2 12 '^1(11)+1$'
2 = 11 unary - prime
3 = 111 unary - prime
4 = 1111 unary - composite
5 = 11111 unary - prime
6 = 111111 unary - composite
7 = 1111111 unary - prime
8 = 11111111 unary - composite
9 = 111111111 unary - prime
10 = 1111111111 unary - composite
11 = 11111111111 unary - prime
12 = 111111111111 unary - composite
```

Put your solution in `regex_prime.txt`, for example:

```
$ test_regex_prime.pl 40 50 "$(cat regex_prime.txt)"
40 = 1111111111111111111111111111111111111111 unary - composite
41 = 11111111111111111111111111111111111111111 unary - prime
42 = 111111111111111111111111111111111111111111 unary - composite
43 = 1111111111111111111111111111111111111111111 unary - prime
44 = 11111111111111111111111111111111111111111111 unary - composite
45 = 111111111111111111111111111111111111111111111 unary - composite
46 = 1111111111111111111111111111111111111111111111 unary - composite
47 = 11111111111111111111111111111111111111111111111 unary - prime
48 = 111111111111111111111111111111111111111111111111 unary - composite
49 = 1111111111111111111111111111111111111111111111111 unary - composite
50 = 11111111111111111111111111111111111111111111111111 unary - composite
```

Your regex must be less than 80 characters.

Hint: you can't do this with a true regular expression, i.e using `|*()` alone, you need to use features that Perl adds.

Don't google for other people solutions - see if you can come up with your own.

When you think your program is working, you can use `autotest` to run some simple automated tests:

```
$ 2041 autotest regex_prime
```

When you are finished working on this exercise, you must submit your work by running `give`:

```
$ give cs2041 lab09_regex_prime regex_prime.txt
```

before **Tuesday 04 August 21:00** to obtain the marks for this lab exercise.

Submission

When you are finished each exercises make sure you submit your work by running `give`.

You can run `give` multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Tuesday 04 August 21:00** to submit your work.

You cannot obtain marks by e-mailing your code to tutors or lecturers.

You check the files you have submitted here.

Automarking will be run by the lecturer several days after the submission deadline, using test cases different to those `autotest` runs for you. (Hint: do your own testing as well as running `autotest`.)

After automarking is run by the lecturer you can view your results here. The resulting mark will also be available via give's web interface.

Lab Marks

When all components of a lab are automarked you should be able to view the the marks via give's web interface or by running this command on a CSE machine:

```
$ 2041 classrun -sturec
```

COMP(2041|9044) 20T2: Software Construction is brought to you by
the School of Computer Science and Engineering
at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au
CRICOS Provider 00098G