ECM2433: The C family

C Worksheet 1: getting started

Programming environment

The programs will be written, compiled/linked and run from the Linux command line. We will be using two servers that are reserved for undergraduate Computer Scientists, named:

```
emps-ugcs1
emps-ugcs2
```

On a Linux machine (or a MacOS machine), log into one of these servers and start up the terminal window (from the menu: Applications; Accessories; Terminal). From a Windows machine use the Putty application to establish a connection.

When using these machines, please bear in mind that they are shared machines.

Everything you will need to do for the first workshops will execute very quickly so there is plenty of CPU resource. Please don't hog resources: if you write a program for the workshops that runs for more than a minute (actually a couple of seconds) then it has a bug; kill it by pressing ^C.

You cannot run the sudo program or install new system-wide software (you can install programs in your userspace). Each attempt sends annoying messages to the systems administrator, so please don't try. If there is software you think you need, please ask.

Make a working directory

At the command line, enter the following commands to create a new directory to hold your C programs, and move to it:

```
mkdir ecm2433
cd ecm2433
```

Exercises

1. Hello World!

Write a C program called hello.c that prints out “Hello World!”. Compile and link it using the instructions in the lecture slides. If this is successful then you should find that your directory contains two new files: hello.o and hello. Run the program by typing

```
./hello
```

What happens if you remove the #include directive and then compile and link the program?

Always explicitly naming the executable to be run as ./hello can rapidly become annoying, so it may be useful to edit your ~/.bashrc file to always put the current directory in your path. In your ~/.bashrc file put the following line:

```
PATH=.:$PATH
```

(If you use a different shell, it's a different file, but you probably know where it is if you've changed shell. Ask if not.)
This puts the current directory (./) at the front of the path that is searched for executable programs. After you have sourced the .bashrc (type source ~/.bashrc at the prompt or logout and login again) you should be able to run the program just by typing its name:

```
hello
```

Purists regard including the current directory in the path as dangerous (because someone might put a malicious program in a directory and persuade you to execute it), but I think the convenience far outweighs the risk.

2. Variable declarations and printf

(a) In a new program, define three variables, as follows:

```
unsigned short int   x = 1;
short int            y = 65535;
unsigned char        c = 'A';
```

Use the printf function to print out these values, like this:

```
x has the value 1
y has the value 65535
c has the value A
```

(b) Set the value of the *y* variable to 65540. What happens when you compile the program?

(c) Set the value of the *y* variable back to 65535. Then add a new program line, as follows:

```
y = y + 5;
```

before the printf statement(s). Do you get what you expect?

(d) Set the values of the *x* and *c* variables, like this:

```
x = -1;
c = 66;
```

before the printf statement(s). Do you get what you expect?

(e) Create a new program which defines three separate variables, assigns them the integer values 0, 100 and –123456 and prints out their values in the following format:

```
variable 1 is          0
variable 2 is        100
variable 3 is    -123456
```

3. Loops and ifs

(a) In a new program, use a for loop to print out the even numbers between 1 and 20.

(b) Now do the same thing, but using a while loop instead of a for loop.

4. Compiling, linking and executing

Edit two new files circle.c and radius.c that have the contents defined in the lecture slides. Compile and link them in different ways as follows (remove any object files and circle before starting each one):

First, compile all the source code and link it in one go:

```
gcc circle.c radius.c -o circle -lm
```

Now compile each source file separately to get object files; then link them together with the maths library:

```
gcc -c circle.c
gcc -c radius.c
gcc circle.o radius.o -o circle -lm
```

Compile one (circle.c)and link with other object code :

```
gcc –c radius.c
gcc circle.c radius.o -o circle -lm
```

The following will all fail. Try each and make sure you understand why.

```
gcc circle.c -o circle -lm
gcc radius.o -o circle –lm
gcc circle.o radius.o -o circle
```