1 A first C program

1.1 In defence of C

- The C programming language has been around for almost 50 years. There are more powerful and sophisticated languages which are extensions of C, such as C++ and Java.
- Many powerful 'libraries' come with these languages, and actually it is rather easy, if you know C, to learn enough C++ to use these libraries. But C++ is complicated and it is a much harder job to actually create such libraries.
- C is a small language and one can learn most of it in a single term. It is unsuitable for projects written by teams of programmers, but can be useful for fairly small jobs, and is the right place to start if you want to learn C++ or Java.
- Modern software is often very greedy of computer resources, occupying disc space measured in gigabytes and requiring powerful machinery even to start running in less than half an hour. Often one can get the results in a few seconds using C code.
- 'Moore's law' predicted in 1965 that computer chips would double in speed every two years, and that prediction was fulfilled. But that rate of improvement has ended and efficiency cannot be ignored forever. C was meant to be efficient (important for the processors of the 1970s).
- (*This point is not favourable.*) Programming in C is hard. Mistakes in C programs can be very hard to find. C was meant to be efficient but not to be easy.

1.2 The minimal C program

The simplest C program is

```
main()
{}
```

Imagine that the above code is written in a file `nothing.c`, stored in a subdirectory called `cprogs`. You need a compiler, software to translate this file into machine code; for example `gcc` or `clang`. This machine (on which these notes are being written) runs `gcc`.

```
% cd cprogs/
% ls
nothing.c
% gcc nothing.c
% ls
a.out  nothing.c
%
```

Before `gcc` there is one file in `cprogs`. After `gcc`, there are two. The second file `a.out` is code which the computer can run directly. To run it, type its name.

```
%a.out
%
```

and the program executes successfully. What does it do? Nothing.

1.3 The smallest program which does something

Every program except the silliest must produce some output. We use `printf` to produce output — to the terminal (monitor). This is not officially a C feature. It is essential to tell the C compiler what it does, hence the `include` statement below. Suppose the C code is stored in `hurrah.c`

```
% cat hurrah.c
#include <stdio.h> // essential for input and output
main ()            // these parentheses are essential
{
  printf("Hurrah for programming\n");
}

// Two slashes introduce a comment.
% gcc hurrah.c
% a.out
Hurrah for programming
%
```

What about the percent sign in `gcc hurrah.c`, and so on? It is a *command prompt* asking you to enter a command — in this case the command is to compile `hurrah.c`

```
% gcc hurrah.c
```

The `gcc` compiler is run on `hurrah.c` It produces no visible output, but it creates the file `a.out` Because there is no visible output, the command prompt returns. Next

```
% a.out
```

Now the computer executes `a.out` which *does* produce visible output.

```
Hurrah for programming
```

And next we get a command prompt. The `\n` indicates *newline*. If omitted then the % would appear to be stuck to the end of the line. If omitted,

```
  printf("Hurrah for programming");
```

and we compile and run the program, we get

```
Hurrah for programming%
```

What about the semicolon? See the next section of the notes.