Notes 1.0: Introduction to programming in C

COMP9021 Principles of Programming

School of Computer Science and Engineering
The University of New South Wales

2014 session 1

Turing machines

A Turing machine (TM) is an idealised computer that like a real computer, stores data in memory and performs operations on the data.

At any time, a TM is in some state q_i and its head has access to the contents c_i of one of the memory cells that make up its infinite tape. It then either does nothing, or

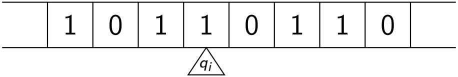
- changes to state q_j (hence remains in the same state if $q_i = q_j$),
- replaces the contents of the cell to c_j (hence leaves the cell's contents unchanged if $c_i = c_j$), and
- moves one cell to the right (R) or one cell to the left (L).

Moreover,

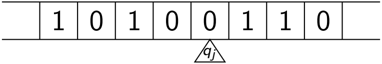
- the contents of any cell is either 0 or 1;
- when it starts, a TM is in a designated initial state q_0;
- a TM is deterministic, in the sense that there is at most one possible action for a given pair (q_i, c_i).

Instructions

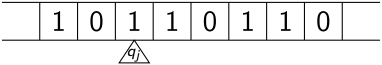
So a basic instruction of a TM has the form $q_i c_i q_j c_j R$ or $q_i c_i q_j c_j L$ with c_i and c_j in $\{0, 1\}$. For instance, if the TM's current configuration is



then executing $(q_i, 1, q_j, 0, R)$ results in the new configuration



whereas executing $(q_i, 1, q_j, 1, L)$ results in the new configuration



Programming a Turing machine (1)

A program for a Turing machine M is a finite set of instructions (so the machine can be in only one of finitely many states).

If M is in state q_i, its head points to a cell that contains c_i, and the set of instructions contains no quintuple that starts with $q_i c_i$, then no instruction can be executed and M stops.

Assume that M is meant to compute the function that maps a nonzero positive number n to $n+2$. We suppose that n is stored in memory in the sense that n consecutive cells contain 1 (all other cells containing 0) and that M's head initially points to the leftmost cell that contains 1. To succeed, M must stop at a point where $n+2$ consecutive cells contain 1 while all others contain 0.

The program consisting of the three instructions $q_0 1 q_0 1 L$, $q_0 0 q_1 1 L$ and $q_1 0 q_1 1 R$ does the job.

Programming a Turing machine (2)

Assume that M is meant to compute the *addition* function that maps two nonzero natural numbers n and m to $n+m$. We suppose that n and m are stored in memory separated by a single cell that contains 0. To succeed, M must stop at a point where $n+m$ consecutive cells contain 1 while all other cells contain 0. The following program does the job.

$$q_0 1 q_1 0 R \quad q_1 1 q_1 1 R \quad q_1 0 q_2 1 R$$

For instance, to compute 2+3, M starts in configuration

	0	0	1	1	0	1	1	1	0	0	0	
			q_0									

and ends in configuration

	0	0	0	1	1	1	1	1	0	0	0	
						q_2						

From TM programs to higher level programs

Turing machines are universal computable devices. So (obviously only) in principle, every computing task could be done using the programming language of TMs, and C or any other programming language can be viewed as only providing "shortcuts".

For instance, the C program add.c provides a convenient shortcut to add 2 and 3, with the extra feature of displaying the result stored in memory. But what does the shortcut actually do? The numbers 2 and 3 are expectedly "still there", they might not be next to each other in memory (applications process huge quantities of data), and we can also expect that the result ends up "anywhere" in memory. The shortcut is likely to perform an operation of the form:

> *Look at the numbers stored at locations x and y, add them up, and store the result at location z.*

The C program detailed_add.c gives evidence that this view is correct.

Programs that don't make sense, incorrect programs

- Some TM programs do not make sense, for instance:

$$q_0 1 0 q_0 R$$

Some C programs do not make sense either, for instance: incorrect.c

- Some TM programs do not do what they are expected to do. For instance,

$$q_0 1 q_1 1 R \quad q_1 1 q_1 1 R \quad q_1 0 q_2 1 R$$

is incorrect if it is meant to compute the sum of 2 nonzero natural numbers.

Some C programs are also incorrect; for instance, wrong_add.c is incorrect if it is meant to get two nonzero natural numbers from the user and compute, store and display the result, even though it seems to be correct when tested by a user who provides 7 and 1 as input.

Programs that never stop, programs that crash

- Some TM programs never stop, for instance:

  $$q_0 0 q_1 0 R \quad q_0 1 q_1 1 R \quad q_1 0 q_0 0 L \quad q_1 1 q_0 1 L$$

  Some C programs never stop, for instance: do_nothing.c
- A TM program cannot crash. This is because a TM has *infinite memory*.
  A real computer has finite memory. Trying to move forever to the next cell on the right of the current cell in memory will eventually make a program crash. The C program crash.c illustrates.

Bye bye TM programming

We have already learnt a lot about programming in general, and C programming in particular.

- A computer performs deterministic, mechanical instructions, that can in principle be reduced to a very small set.
- 0's and 1's stored in contiguous memory cells code the data.
- A program can fail to make sense, fail to be correct, fail to stop, and it can crash; this will be experienced more often than wished...
- We have made our first acquaintance with pointers, claimed to be the most difficult aspect of the C language.
- We have a very simple but powerful enough model of memory that is essential to understanding fundamental programming concepts. This model is necessary, and in many ways sufficient, to develop strong C programming skills.

Programs, inputs, outputs

We will study the activity of:

- writing *programs*, making use of and for use by a computer;
- providing data—the *input*—to the computer;
- letting the computer code the input and the program into sequences of bits, namely, 0s and 1s, and store those sequences into memory;
- letting the computer execute the program and perform various operations on bits;
- letting the computer decode some sequences of bits that represent computation results—the *output*;
- having the output displayed or stored.

	1	0	1	1	0	1	1	0	0	1	
	76134_a	76134_b	76134_c	76134_d	76134_e	76134_f	76134_g	76134_h	76135_a	76135_b	

A simple abstraction of memory

Dealing with input and output (1)

Input can come from:

- the keyboard, in two possible ways:
  - when the user starts the program and provides command line arguments;
  - after the program has been started and stopped execution, usually prompting the user to enter some information;
- a file.

Output can be sent to:

- the screen;
- a file.

Some programs do not need any input, or do not produce any output.

The program input_output.c, together with the interaction described next, gives a flavour of how a program will be compiled and run, getting input and yielding output in all possible ways just mentioned.

Dealing with input and output (2)

```
$ ls input_file.txt
input_file.txt
$  cat input_file.txt
There is only line in this file.
$ ls output_file.txt
ls: output_file.txt: No such file or directory
$ gcc -std=gnu99 -Wall input_output.c
$ ./a.out v
Enter a character please:  X
I have seen the characters v, X and T
$ ls output_file.txt
output_file.txt
$ cat output_file.txt
I have kept track of the characters v, X and T
```

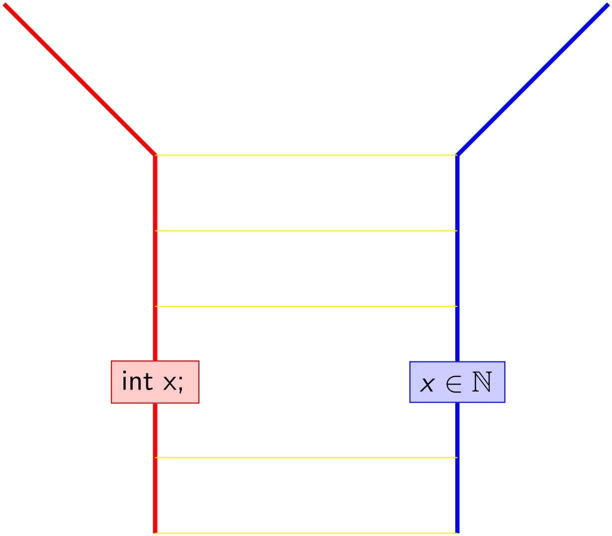
Two worlds, close up to a point where they diverge... (1)

Often programs deal with numbers and operations on numbers. Basic mathematical functions, such as multiplication between integers, are already implemented and ready for use.

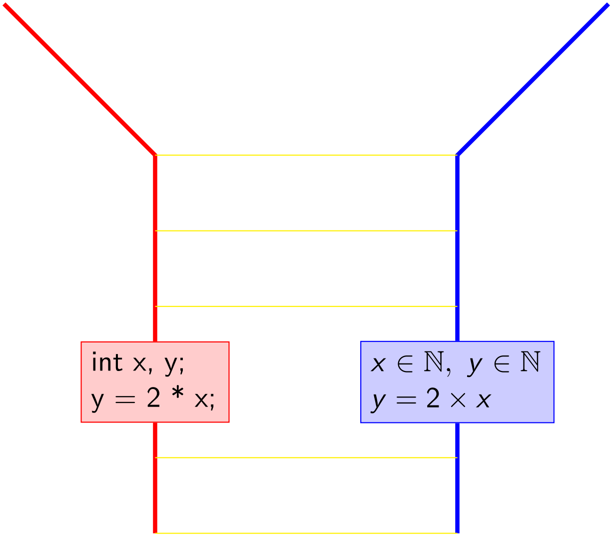
The output of the program two_worlds.c suggests that the relationship between the world of computing and the usual world of numbers and operations on numbers is not as simple as one might expect, as both worlds closely correspond to each other up to a point where they diverge.

Good programmers program with both worlds in mind.

Two worlds, close up to a point where they diverge...(2)



Two worlds, close up to a point where they diverge. . . (3)



The world of numbers and functions

- A palindrome is a sequence of characters that can be read either from left to right or from right to left, for instance:
  - racecar
  - delia saw I was ailed
  - 146641
- A perfect square is a number of the form n^2 for some natural number n, such as 4, 81, or 144.

The program perfect_square_palindromes.c finds all 6-digit palindromes that are perfect squares; designing such a program is a small exercise in problem solving. It yields as output:

```
The solutions are:

698896 (equal to the square of 836)
```

The world of 0s and 1s

The key representation of the data objects of the computing world is

name	*address*	*value*	*type*

with as particular example,

`distance`	`0xbffff1a8`	`00101011001010100010100100101000`	`int`

as illustrated (except for the actual address) by the program

zeros_and_ones.c

The main steps of the programming activity

1. Define the program objectives: determine what the program is meant to achieve.
2. Design the program: determine how to represent data and which algorithms to implement.
3. Write the code: translate the design into the C language.
4. Translate the source code into executable code.
5. Run the program.
6. Test and debug the program.
7. Maintain and modify the program, make it more efficient.

In practice, particularly for larger projects, the process is not linear and the programmer has to go from one step back to earlier steps.

The gear we need to program

- Steps 1 and 2 need pen and paper.
- Step 3 needs an editor.
- Step 4 needs a compiler and a linker.
- Step 5 needs a command interpreter.
- Step 6 needs a debugger.
- A profiler is used to improve program efficiency.
- A control version system is used to keep track of the various modifications to the code.

Compiling programs from the command line

The source code of a C program has to end in `.c`.

Compiling a program whose source code is stored in a unique file *file_name*`.c` can be done in many ways, starting with `gcc` *file_name*`.c` with, between `gcc` and *file_name*`.c`, possibly one or more of

- `-o` *output_file_name*, so that the executable code be stored in *output_file_name* rather than the default `a.out`,
- `-Wall` to get all relevant Warnings, hence maximise the chances of detecting possible mistakes,
- `-ggdb` to generate the information needed to use a debugger,
- `-std=gnu99` to use the features of the C99 standard, as we will in this course,
- not to mention hundreds of other options. . .

Running programs from the command line (1)

A program that has been compiled without the `-o` option can be run by typing `./a.out`, to execute the file `a.out` stored in the current (`.`) directory.

If the current directory is in your path (the sequence of directories that are searched when a command is to be executed), then you can type `a.out` rather than `./a.out`.

To have the current directory as the (preferably last) directory in the list of directories that makes up your path, do the following.

- In your home directory, that you access by typing `cd`, check that you have a file named `.profile` by typing `ls .profile`; if you don't, create one by typing

```
>.profile
```

Running programs from the command line (2)

- if the file `.profile` contains no line starting with `export PATH=` then insert somewhere in `.profile` the line

```
export PATH=$PATH:
```

- If the file `.profile` contains a line that starts with `export PATH=` with on the right hand side of the `=` sign, a sequence of symbols that does not start with a colon possibly preceded with a dot, does not contain two successive colons possibly with a dot in between, and does not end in a colon possibly followed by a dot, then add a colon to the end of that sequence of symbols.

To change the value of the path without having to first log out and log in again (the `.profile` file is read every time you log in, so nothing will have to be done for all future sessions), type

```
. .profile
```

Programming with an IDE

Eventually, you will use an Integrated Development Environment—a single application that integrates most of the programming gear.

An IDE is indispensable to work on large projects.

Still, when working on many small programs, learning and experimenting, an IDE is uselessly powerful and complicated.

In this course, you will have the option to work with a lean but powerful programming environment which should make your learning experience as simple and rewarding as possible.