Notes 1.0: Introduction to programming in C

COMP9021 Principles of Programming

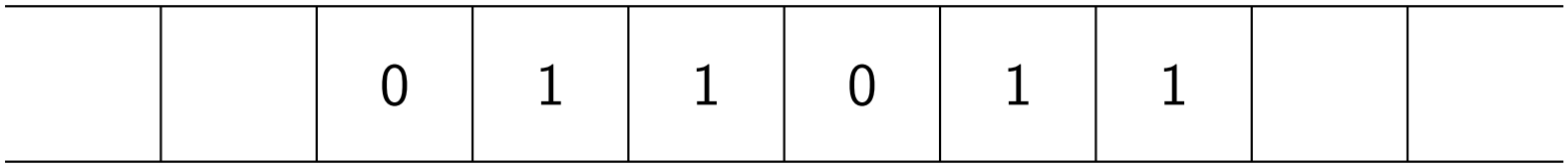
School of Computer Science and Engineering
The University of New South Wales

2010 session 2

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.



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
```

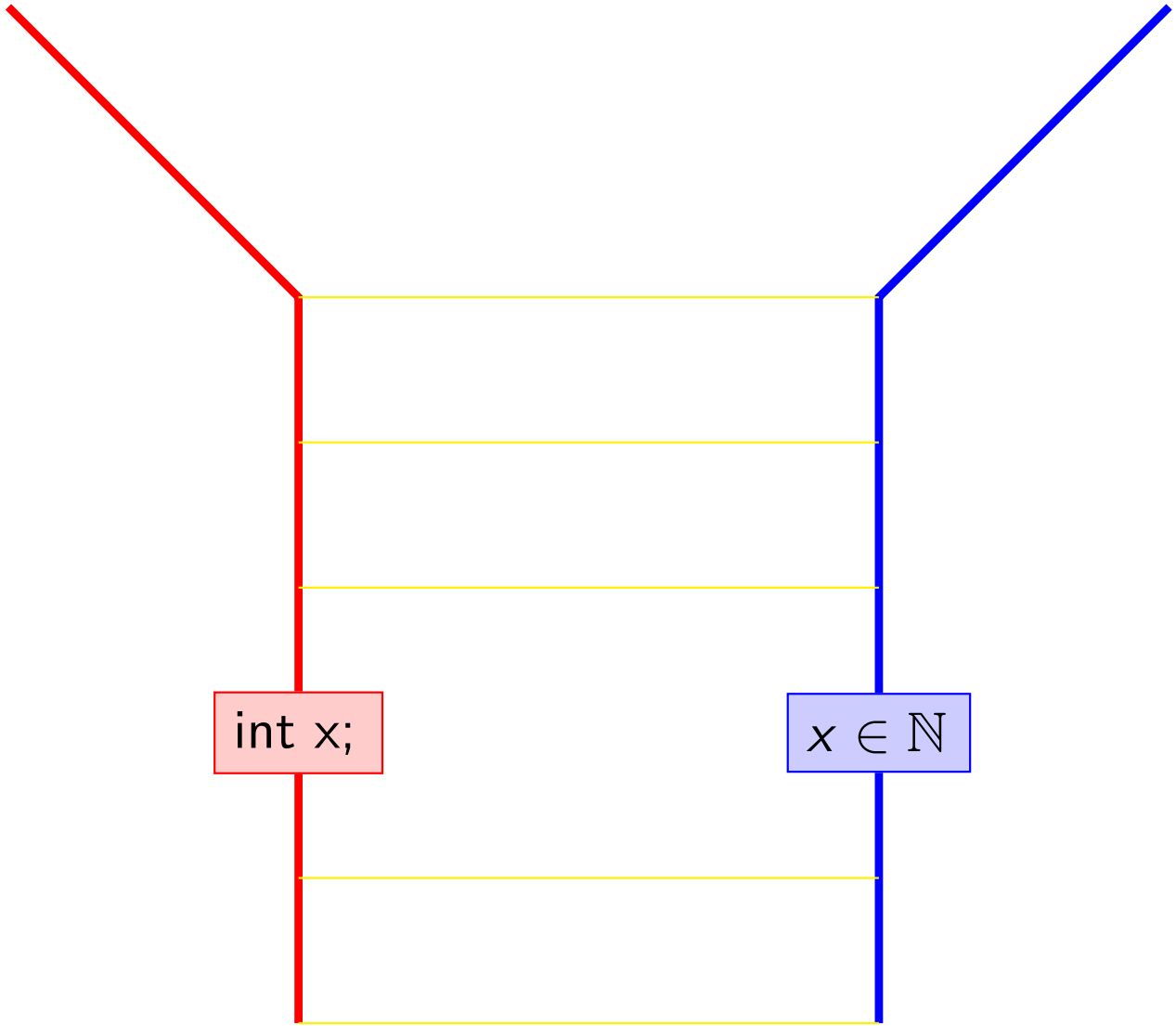
Two worlds, close up to a point where they diverge. . . (1)

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

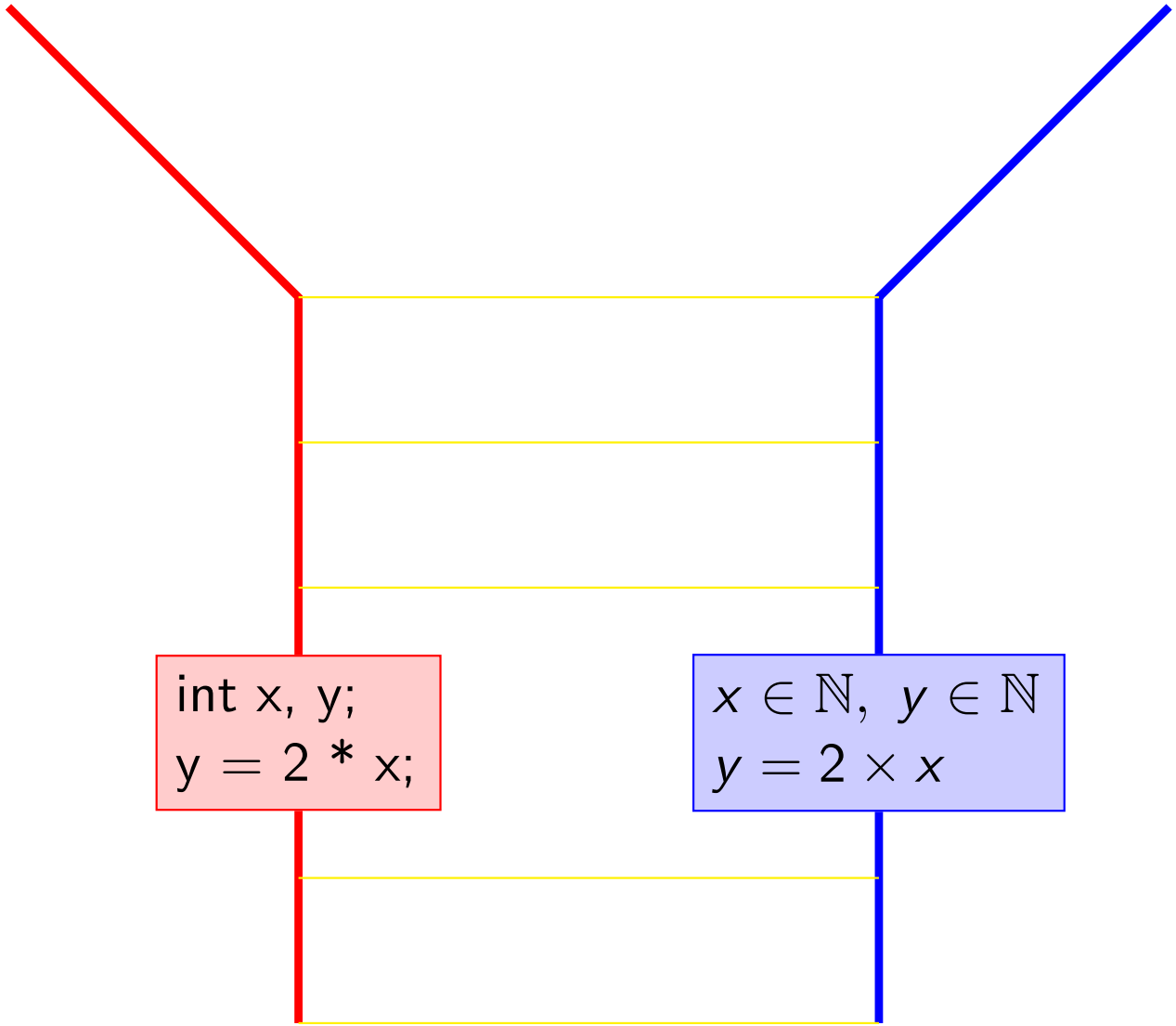
The output of the program two_worlds.c suggests that the relationship between the world of computing and the usual world of numbers and operations on numbers is not as simple as one might expect, as both worlds closely correspond to each other up to a point where they diverge.

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

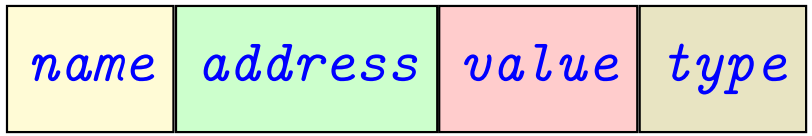
- A palindrome is a sequence of characters that can be read either from left to right or from right to left, for instance:
  - racecar
  - delia saw I was ailed
  - 146641
- A perfect square is a number of the form n^2 for some natural number n, such as 4, 81, or 144.

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:

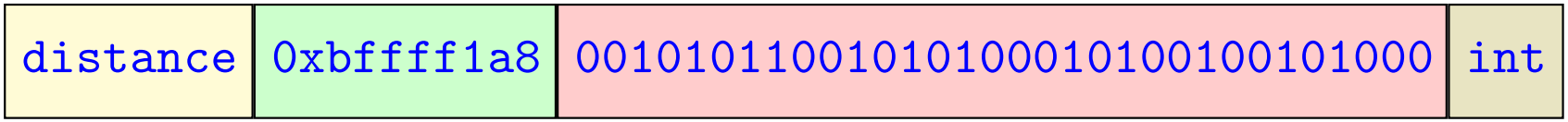
```
The solutions are:

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



with as particular example,



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

zeros_and_ones.c

The main steps of the programming activity: the 7 steps (1)

Working on a program is a process that can ideally be broken down into 7 Steps:

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.

The 7 Steps(2)

- In practice, particularly for larger projects, the process is not linear and the programmer has to go from one step back to earlier steps:
  - 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.
- An IDE(Integrated Development Environment) is a single application that integrates all the tools to perform those steps.

A Short History of C: Early Developments

- C was initially developed at AT&T Bell Labs between 1969 and 1973 on the UNIX operating system.
- Many of C's features were derived from an earlier language called B, hence the name C.
- By 1973 most of the UNIX kernel–the heart of the operating system–, originally written in assembly language, was rewritten in C.

K&R C

- In 1978 Dennis Ritchie and Brian Kernighan published the first edition of The C Programming Language, known to C programmers as K&R.
- K&R describes a version of C that is commonly referred to as K&R C.
- In the years following the publication of K&R, various additional features were added to the language and supported by compilers (programs that translate C code into assembly code) from various companies.

ANSI C and ISO C(1)

- In 1983 the American National Standards Institute formed a committee to provide an unambiguous and machine-independent definition of the language C.
- The standard was completed in 1989 and ratified as ANSI X3.159-1989 Programming Language C, often referred to as ANSI C.
- ANSI C brought a few changes to K&R C, officialized some of the additional features previously introduced, and specified a standard library.

ANSI C and ISO C(2)

- ANSI C is the version of C described in K&R's second edition.
- ANSI C is now supported by most compilers, and most of the C code being written nowadays is based on ANSI C.
- In 1990, the ANSI C standard was adopted (with a few minor modifications) by the International Standards Organization as ISO/IEC 9899:1990.

C99

- ANSI C underwent revision in the late 1990s to support international programming and codify existing practice to address evident deficiencies.
- This lead to the publication in 1999 of ISO 9899:1999
- It was adopted as an ANSI standard in March 2000, and is commonly referred to as C99.
- The interest in the new features introduced in C99 appears to be mixed.
- Some compilers including GCC support most of the new features of C99, but the compiler maintained by Microsoft doesn't.

C's Strengths

- Though C was originally designed and implemented on the UNIX operating system, C is not tied to any particular system.
- C is portable: programs written on one system can be run on other systems with little or no modification.
- Though C was originally designed and implemented to fulfill the needs of programers writing compilers and operating systems, C is not tied to any particular application.
- C is powerful and flexible: major programs in many different areas are written in C.
- C is efficient: programs can run fast and memory can be used sparingly.

C's Shortcomings

With C more than with many other programming languages:

- it is possible to make programming errors that are very difficult to spot;
- it is possible to make programming errors that result in programs with unpredictable behavior;
- it is possible to write code that is extremely difficult to read and understand.

Also C is a relatively low in the family of high level languages: it is close to the hardware and more similar to machine language than many other languages. This gives more control to the programmer, but the downside is that higher level mechanisms must be coded explicitly.

Editing a program

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

Example:
File name: HelloWorld.c

```
/* prints the greeting "Hello World!" on the screen */

#include <stdio.h>
int main(void) {
printf("Hello World!\n");
return 0;
}
```

You can edit the program using a text editor, such as, emacs or pico in UNIX.

Compiling programs from the command line

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` (you probably do); 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.

In this course, you will have the *option* to work with a lean but powerful programming environment.
However, in class examples, emacs/pico and UNIX commandline will be used for editing and compiling/running the programs.

Studying in comfort (1)

Learning programming always involves practice, it is never a purely theoretical exercise. What could be easier than clicking on a link to have a program opened in an editor, where it can be compiled, run, modified?

Acrobat Reader makes this possible (though the behaviour is more or less satisfactory depending on the operating system); when prompted to take some action after clicking on such a link, it must be told that the file is meant to be opened in Emacs (`/usr/bin/emacs22` on a School machine), and possibly that this should be the default action for all files of this type.

Since links in a pdf file are absolute, it is necessary to fix the links in the pdf file of the lecture notes you save together with all files provided—which includes the files for the source code of the example programs—, in order to be able to click on the links with an effect.

Studying in comfort (2)

I provide you with a perl script, called `fixlinks`, to easily fix the links. It is convenient to store it in a directory such as `~/comp9021/scripts` (the subdirectory `scripts` of the subdirectory `comp9021` of your home directory, that first has to be created using the `mkdir` (make directory) command in case it does not exist.

Whatever that directory is, it is convenient to store it in your path. With `~/comp9021/scripts` as example, and assuming that you have already edited your `.profile` file as previously explained, you would edit `.profile` again and add `$HOME/comp9021/scripts:` to the end of the line that starts with `export PATH=` (and run `. .profile` if you want the change to take effect without having to log out first).

You can fix the links in a pdf file by invoking `fixlinks` with a path to that file provided as argument. For instance, to fix the links in the pdf file for this set of lecture notes, one can, in the directory where that file is kept, type `fixlinks notes1.pdf`.