U11601 C programming (intro)

Rec. lecture Ø

* How to compile a C program, login to maths machines, transfer files, submit programming assignments.

* C programming handbook (print it)
* Class notes
* The *nix command line

* lectures Tu 2 recorded, Fri zoom
* 50% coursework (programs, quizzes) and 50% final exam.
* Zoom tutorials for programs + quizzes.
* identify clashes.

Topics how to write + read
simple programs, comments, for (—),
if(), while()

Data types short, int, float, double, char, char*
→ internal rep of given no.

arrays
→ address of array elements

Subroutines, functions, recursion
→ simulating routines
→ " recursion
→ correctness + loop invariants

2-dim arrays

Conversions, casts

pointers, allocation

structured types.

U11601 intro C programming

Prerecorded lectures + 1 zoom/week

50% coursework 50% final

5 `quizzes' and weekly programming assignments

Lectures accompany web notes which are
on my web page, not Blackboard
www.maths.tcd.ie/~odunlain/

The smallest C program does nothing

```
main()
{}
```

The second smallest:

```
#include <stdio.h>
main()    // Prints a message
{
    printf("Hurrah for programming\n");
}
```

#include: stdio.h is essential to allow the use of printf

// double slash introduces a comment

printf: prints to terminal

\n: newline or carriage return

A C program is just text, like an email. It must be translated into`executable code' using a program called a compiler.

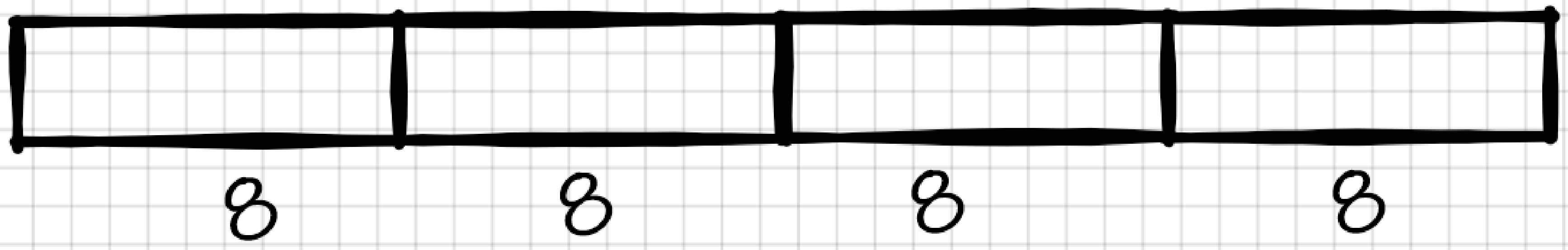
The machine on which these notes are being written has no C compiler. I need to remote login to get the use of the C compiler. Of course you can get free C compilers, but there is another reason you need to remote login to the maths machines: in order to submit programs and quizzes. This will be discussed later.

All the maths computers are Unix machines, but two kinds of Unix, and the C compilers have different names.

The Linux machines have gcc (gnu cc) and are named hamilton, gstokes, and aturing

The FreeBSD machines have clang and are named jbell, synge, walton, and salmon

Print a variable



Integers are usually encoded as binary numbers consisting of 32 bits. A BIT is a binary digit, a BYTE is a group of 8 bits; so an integers occupies 4 bytes. 32-bit binary numbers are between 0 and 4 billion (or: 2^32 - 1) but the encoding allows for negative numbers, so integers vary between -2 billion and 2 billion roughly.

4 gigabytes

A basic C program is:
#include statements
int main () // the int is a convention
{ .... declarations and statements...}

Every statement ends in a semicolon
A declaration introduces one or more variables

Example: print1.c

```
#include <stdio.h>
int main()
{ int i; // i is an integer variable
   i = 13;
   printf("i is %d\n", i);
 }
```

1 declaration + comment

2 statements: assign 13 to i
print i

```
 % gcc print1.c
 % a.out
 i is 13
 %
```

clang on FreeBSD machines

The compiler converts the C text
to a.out, an `executable program,'
which is executed by the command
a.out

Pay attention to the prompt %

Integers m, n; $n > 0$

Maths convention $m \div n$ is the largest integer $\leq \frac{m}{n}$ rounded down and $m \bmod n$ is the remainder so

$$m = (m \div n) * n + m \bmod n$$

$$99 = (99 \div 8) * 8 + 99 \bmod 8$$

$$99 = 12 * 8 + 3$$

$(-99) \div 8 = -13$

$(-99) \bmod 8 = 5$

In C, round towards zero. $/$: division

$\%$ remainder

$99/8 = 12$ $\quad$ $99 \% 8 = 3$

$(-99)/8 = -12$ $\quad$ $(-99) \% 8 = -3$

Print statements can print more than just variables They can print `expressions':

```
#include <stdio.h>
int  main()
{ int i;
  i = 13*14;
  printf("i is %d\n", i+15);
}
```

running gcc and a.out you get
i is 197
which is correct.

Arithmetic expressions. Programming languages allow for addition, subtraction, multiplication, division, and many other operations, including the remainder on division by a number. The keyboard has no special sign for multiplication or division, so * and / are used.

The signs are + - * / %

Rules for evaluation: The `BODMAS' rules apply, brackets, division, multiplication, addition, subtraction. More detailed rules will be given later.

At present, we focus on integer variables. Highly accurate calculations are possible with another kind of variable, called `double.'

in dividing m/n, if m and n are positive the answer is as expected: it is rounded DOWN to the NEAREST integer. Thus 7/4 = 1.

Important technicality

In algebra there is the so-called `division algorithm' which says that if m is an integer and n a positive integer, there exist unique integers q and r such that

m = qn+r   AND 0 <= r <= n-1

q is called the QUOTIENT and r the REMAINDER.

In C, this is different when m < 0 (and n>0): the quotient is `rounded towards zero' so -(n-1) <= r <= 0.

The remainder r is also called `m modulo n' and in C, where it is surprisingly important, itwould be m%n. The round towards zero rule carries over to m%n when m is negative (and n positive), so in C, if m < 0 and n > 0, m%n < 0.

See the web notes for an example.

Assignments.

The statement

```
x= y;
```

is an ASSIGNMENT, Not an equation. It evaluates y (maybe a complex expression) and assigns the value to x (assuming x is an integer andy an integer expression).

```
Example.   x = 14;  // now x is 14
           x = x+1;  // now x is 15.
```