2 Print a variable

2.1 General structure of a C program

A C program is, at its simplest,

```
Some #include statements

int main () // the int is a convention
{
  ...declarations and other statements ...
}

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

2.2 Integer variables

Example of an integer variable.

```
#include <stdio.h>

int main()
{
  int i;
  i = 13;
  printf("i is %d\n", i);
}
-------------------------
% gcc print1.c
% a.out
i is 13
```

This program has three *statements.*

```
  int i;
        // this is a declaration, declaring that ‘i’
        // is is the name of an integer variable.
  i = 13;
        // this assigns the value 13 to i.
  printf("i is %d\n", i);
        // this prints the value of i, with surrounding
        // text.  The %d shows where value of i will be
        // placed in the output. %d means ‘integer value’
        // within print statements.
```

2.3 Printing integer expressions

```
#include <stdio.h>

int main()
{
  int i;
  i = 13*14;
  printf("i is %d\n", i+15);
}
```

This shows that one can — this is no great surprise — do arithmetic in C. Also, one can print `i+15` which has a definite value but is more than a single variable. Its output is

```
i is 197
```

2.4 Arithmetic operations

For the present, only integer data is considered.[1]

Integer expressions are formed from integer variables and constants (such as -45) with the following binary *operators*

```
+    -    *    /    %
```

whose meanings are mostly obvious. There are no keyboard characters for $\times$ and $\div$, so $*$ and / are used.

The percent sign is *remainder modulo*: `m % n` is the remainder on dividing m by n.

2.4.1 Round towards zero

Technically, integer division, in the usual mathematical convention, satisfies the following property. We write $\div$ for integer division. Assume $n > 0$. (The case $n < 0$ is not worth considering).

It can be shown that there exist *unique* integers q and r such that

$$m = qn + r \quad \text{and} \quad 0 \leq r \leq n - 1$$

We recognise q as the *quotient* $m \div n$, and r as the *remainder* $m \mod n$. The quotient is rounded down.

If m is nonnegative, then exactly the same holds in C:

```
m == (m/n)*n + m%n
```

If m is negative (still assuming that n is positive), then the remainder is *non-positive*, and the quotient is rounded up. So again,

```
m == (m/n)*n + m%n
```

but `m%n` is non-positive and `(m/n)*n` is $\geq$ `m`.

[1] `double` is for non-integer data, to high precision.

2.5 Assignment

```
x = y;
```

is ***not an equation***
it is an *assignment.* The value of y, which could be a very complicated expression, is calculated, and *assigned* to x.

So, for example, after these two statements are executed,

```
x = 14;
x = x+1;
```

`x` has the value 15.