5 Printf and scanf and while-loops

5.1 Printf

Printf is for 'formatted printing.'

```
printf(<format string>,  <item1>, <item2> ...)


printf("hello");
printf("hello\n");
printf("%s", "hello\n");
printf("%s\n", "hello");
printf("%s %s\n", "hello", "there");

printf("hell%c\n", 'o');

int i;
for (i=0; i<4; ++i)
  printf("%d potato\n", i+1);

double x;
x = 0;
for (i=0; i<4; ++i)
  x = x+i;
printf("x is %f\n", x);
```

The items (if there are any) are 'embedded' in the format string and the result is printed. There are different 'format codes'

- `%d` integer (printed as decimal)
- `%f` floating-point (double) printed decimal, default 6 decimal places.
- `%c` a single character
- `%s` a character 'string.'

The computer doesn't check that the format item correctly matches the item to be printed. For example,

```
  printf("%d\n", "hello");
```

will print an *apparently* random set of digits.

5.2 Scanf

Scanf is for *reading* data from the keyboard. It resembles printf deliberately

```
scanf ( <format string> , <item1>, <item2> ...);
```

Items are read from the keyboard and stored.
There are four vital differences between printf and scanf.

- It is best to use scanf only for reading numeric data, no text. Spacing is ignored when reading via scanf.
- In scanf, the items read in are stored at various places in memory. Their *addresses* must be given. This is not true of printf, where only the values matter.

  The address of the variable `n` is

  ```
  &n
  ```

  For example

  ```
  int n;
  scanf ( "%d", & n );
  ```

  will cause n to be read from the keyboard. The *memory address* of the variable n must be used.
- One needs to be much more careful with 'format control items' on input. Given ('output')

  ```
  double x;
  x = 3.14159;
  printf("%f\n", x);
  ```

  will print what you would expect. But

  ```
  double x;
  scanf("%f", &x);
  ```

  Will give spurious answers, because the `%f` means 32-bit: a double-precision number occupies 8 bytes and this scanf would only fill four of them.

  For `scanf()`,

  - `%d` for `int`, an integer (4 bytes),
  - `%h` for a `short` integer (2 bytes),
  - `%f` for a `float` (details later), a 4-byte floating point number ('single precision')
  - `%lf` (that's an ell, not a one) for `double`, an 8-byte floating point number ('double precision').
- `scanf()` *returns a value,* the number of items successfully scanned.

5.3 While-loops

A while-loop is simpler than a for-loop:

```
  while ( <condition holds> )
  {
    ....
  }
```

Example, showing how to read a sequence of numbers from the keyboard. What you do then is a different matter.

```
prompt% cat scan_example.c
#include <stdio.h>
main()
{
  double x;
  while ( scanf("%lf", &x ) == 1 )
  {
  } // do nothing ( this is a comment )

  printf("That's the lot\n");
}
prompt% gcc 2.c
prompt% a.out
1
3 2 5
17 1
3

4
That's the lot
prompt%
```

And what made it stop after that 4? Answer: it is not visible, but after the 4, at the beginning of the next line, a **ctrl-D** character was typed. *Ctrl-D, at the beginning of a line, marks the end of input to the 'scanf' function.*