3 For-loops

In C, a group of statements between curly braces `{...}` is equivalent to a single *compound* statement.[1]

3.1 For-loops

For-loops are designed to perform the same group of statements several times. For example

```
#include <stdio.h>

int main ()
{
  int i;
  for ( i=0; i<3; i = i+1 )
  {
    printf("i is %d\n", i);
  }
}
-----------------
% a.out
i is 0
i is 1
i is 2
%
---------------
```

A for loop follows this pattern:

for (preparation; condition; before next step) **{ statement; . . . statement; }**

It works this way. It is called a 'loop' to suggests that it returns to the same action several times.

- It performs the 'preparation.' In the example, the preparation is to set $i = 0$.
- It repeats the following steps:
  - It checks the 'condition.' If the condition is false, then the 'loop' terminates. In this example the condition is that i is less than 3.
  - It the condition is true, then the statements **{ . . . }** are performed, and the 'action before next step' is performed. In this example, the statements, or statement in this case, is to print `i is ....` and the 'action before next step' is to add 1 to i.

[1] The curly brace } should *not* be followed by a semicolon.

3.2 Tracing out the example

```
Preparation: i becomes 0.

Condition: is i < 3? Is 0 < 3? yes.
prints ‘i is 0’
action before next step: increment i,  so i is now 1.

Condition: i is 1; is i<3? Yes
prints ‘i is 1’
action before next step: increment i,  so i is now 2.

Condition: i is 2; is i<3? Yes
prints ‘i is 2’
action before next step: increment i,  so i is now 3.

Condition: i is 3; is i<3? No. The loop, and
the program, ends.
```

3.3 A more interesting example

What does this do?

```
#include <stdio.h>

int main()
{
  int n, x, i;

  n = 10;
  x = 0;

  for (i=0; i<n; ++i) // alternative to i = i+1
  {
    printf("%d\n", x);
    x = x + 2*i + 1;
  }

  printf("final value of x %d\n", x);
}
```

3.4 Conditions in C

Conditions are built using

C	meaning
`<`	$<$
`<=`	$\leq$
`==`	$=$
`>=`	$\geq$
`>`	$>$
`!=`	$\neq$

Warning. '=' always means *assignment* in C, *never* equality. The symbol (or symbols) for equality is '=='.