Programming with C

Terry Marris November 2010

10 Iterations - for

Previously we looked at while loops, dry runs and filters. We continue our consideration of loops by examining the *for* statement.

10.1 for Loops

We start with a simple example that sums the first five integers, starting with zero:

0 + 1 + 2 + 3 + 4 = 10

Using a *while* loop, you might write something like

```
int sum = 0;
int i;

i = 0;
while (i < 5) {
  sum = sum + i;
  i++;
}
```

The equivalent using a *for* loop is

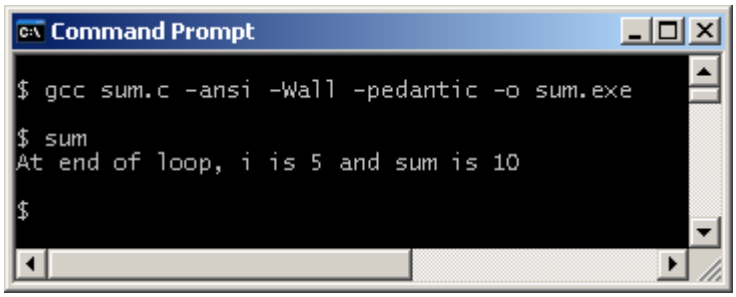
```
int sum = 0;
int i;

for (i = 0; i < 5; i++)
  sum = sum + i;
```

There are three parts within the brackets, each separated by a semi colon. The first part, *i = 0*, is the initialisation. This is done only once, just before the loop is entered. Then comes the condition that controls the loop, *i < 5*: loop while *i < 5* is true. Finally, you have the re-initialisation stage, *i*++, which is executed once each time round the loop.

The nice thing about *for* loops is that all the elements for loop control, initialisation, guard and re-initialisation, occur all together in one place.

Here is the complete program and its run.

```
/* sum.c: introduces the for loop */

#include <stdio.h>

int main()
{
  int sum = 0;
  int i;

  for (i = 0; i < 5; i++)
    sum = sum + i;

  printf("At end of loop, i is %d and sum is %d\n", i, sum);
  return 0;
}
```



If you have more than one statement in the loop body, you would need to use braces to complete a statement block, as shown below.

```
int num = 2;
int sum = 0;
int i;

for (i = 0; i < 5; i++) {
  sum = sum + num;
  num = num + 2;
}
```

So, which one should you use? *while* or *for*? Well, it does not really matter. Whichever one you prefer.

10.2 Endless Loops

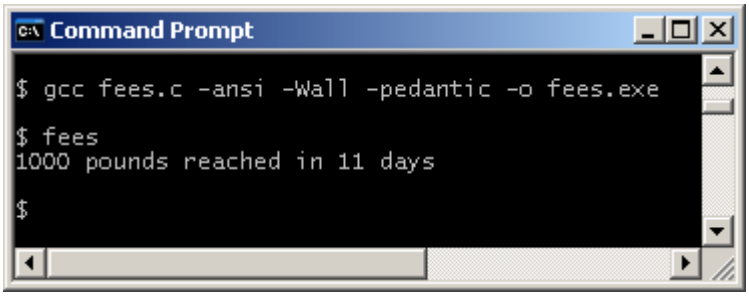
You can have an endless loop by writing

```
for ( ; ; )  { /* loop for ever */
  ...
}
```

But it would be helpful if you can break out of it when you want to. You can break out of any loop with the *break* statement, as shown in the following example.

Your starting fee on a new job is £1, but is doubles each day you are on the job. How many days will it take for your fee to reach £1000?

```
/* fees.c: calculates many days to reach £1000 */

#include <stdio.h>

int main()
{
  double fee = 1;
  int day = 1;

  for ( ; ; ) {
    day++;
    fee = fee * 2;
    if (fee >= 1000.00)
      break;
  }

  printf("1000 pounds reached in %d days\n", day);
  return 0;
}
```



10.3 Square Root

We see how to find the square root of a positive number using the Newton-Raphson method. Basically, we make a guess and see whether guess^2 is close enough to the given number. If it is not, we refine the guess.

If *guess* is an approximation to the square root of *number*, then

betterGuess = (guess + number / guess) / 2

is a better *guess*.

We want to find the square root of 36.

A first guess is *number* / 2 = 18

18 x 18 is nowhere near 36. We need a better guess.

betterGuess = (18 + 36 / 18) / 2 = 10

10 x 10 is nowhere near 36. We need a better guess.

betterGuess = (10 + 36 / 10) / 2 = 6.8

6.8 x 6.8 is nowhere near 36. We need a better guess.

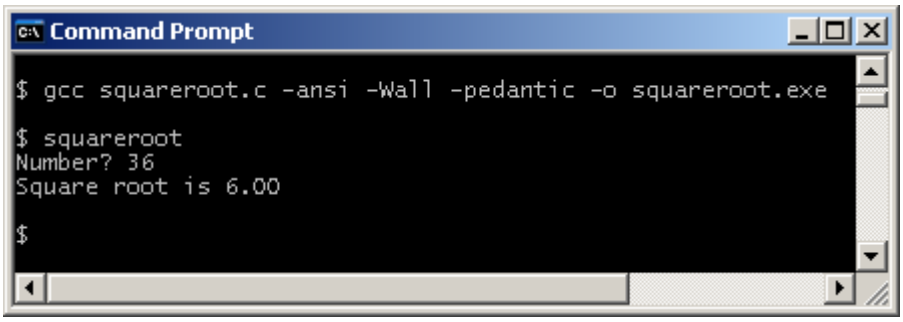
betterGuess = (6.8 + 36 / 6.8) / 2 = 6.047

6.047 x 6.047 = 36.57. Not close enough to 36. We need a better guess.

betterGuess = (6.047 + 36 / 6.047) / 2 = 6.0002, which is close enough.

Here is the program and its run.

```
/* squareroot.c: uses Newton-Raphson method */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>

int main()
{
  double epsilon = 0.0001;
  double num, prev, curr;
  char string[BUFSIZ];

  printf("Number? ");
  gets(string);
  num = atof(string);

  if (num < epsilon) {
    printf("number cannot be less than zero\n");
    exit(EXIT_FAILURE);
  }

  prev = num;
  for (curr = num / 2; abs(prev * prev - num) > epsilon;
                  prev = curr, curr = (prev + num / prev) / 2)
    ;
  printf("Square root is %0.2f\n", curr);
  return 0;
}
```



Notice that the re-initialisation stage comprises two statements separated by a comma:

```
prev = curr, curr = (prev + num / prev) / 2
```

Both of these are executed once each time round the loop.

Also notice that the loop has no body since all the work is done within the loop header. We have placed a semi-colon for the empty loop body under the loop heading.

```
for (curr = num / 2; abs(prev * prev - num) > epsilon;
                  prev = curr, curr = (prev + num / prev) / 2)
      ;
```

Better to put it there than on the same line as the loop heading because doing so in the wrong circumstances is the cause of so many unexpected run-time errors.

Exercise 10.1

1. Write and test a program that will print a times table up to 12 x n, where n is a value chosen by the user

2. Given that there are 1.61 Kilometres in one mile, write and test a program that prints a conversion table showing kilometres for miles in steps of 5 miles, from 5 up to 100. Use appropriate headings and ensure data is right lined under their heading.

3. Given that there are 0.62 miles in 1 Kilometre, write and test a program that prints a conversion table showing miles for Kilometres in steps of 10 Km, from 10 up to 200. Use appropriate headings and ensure data is right lined under their heading.

4. If y is one approximation to the cube root of x, then

$$z = \frac{2y + (x / y^2)}{3}$$

is a better approximation. Design, write and test a program that calculates the cube root of a real positive number input by the user.

We have seen how to implement iterations using the *for* statement.

Next we look at functions.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988
Dromey R *How to Solve it by Computer* Prentice Hall 1982