Programming with C

Terry Marris November 2010

9 Iterations - while

Previously we looked at the switch-with-case statement and the conditional operator. Now we leave the consideration of selections and move onto loops.

9.1 while Loops

Iterations, repetitions, loops all mean the same thing: the coding that causes a sequence of statements to be repeatedly executed. We start with a small example: sum the first 5 even integers 2, 4, 6, 8 and 10. The answer should be 30.

What is happening here?

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

i = 0 is the initialisation stage. *while()* introduces the loop. *(i < 5)* is the Boolean expression that guards entry to the loop. *{* and *}* mark the body of the loop. **. . .** represents the statements to be repeatedly executed (which we ignore for the moment). *i*++ is the re-initialisation stage.

Initially, ***i*** is zero. We come to the *while()*. ***i*** is less than 5. So we enter the loop body and add 1 to ***i***. ***i*** is now **1**. We are at the bottom of the loop. We return to the top of the loop, to the *while()*.

The Boolean expression is still true, ***i*** is less than 5. So we enter the loop body and add 1 to ***i***. ***i*** is now **2**. We come to the bottom of the loop. We return to the top of the loop.

The Boolean expression is still true, ***i*** is less than 5. So we enter the loop body and add 1 to ***i***. ***i*** is now **3**. We come to the bottom of the loop. We return to the top of the loop.

The Boolean expression is still true, ***i*** is less than 5. So we enter the loop body and add 1 to ***i***. ***i*** is now **4**. We come to the bottom of the loop. We return to the top of the loop.

The Boolean expression is still true, ***i*** is less than 5. So we enter the loop body and add 1 to ***i***. ***i*** is now **5**. We come to the bottom of the loop. We return to the top of the loop.

This time, the Boolean expression is false, ***i*** is not less than 5. So we exit the loop. Finished.

We loop for as long as, i.e. while, the Boolean expression remains true. The Boolean expression is tested only at the top of the loop.

Initially, the *sum* is zero and the even number, *num*, is 2. Every time round the loop we want to add the even *num* to *sum,* and add 2 to *num* get the next even number.

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

	sum	***num***	***i***	***i < 5?***
Initially	0	2	0	true

i is less than 5 so we enter body of the loop, add *num* to *sum*, add 2 to *num*, and add 1 to *i*.

	sum	***num***	***i***	***i < 5?***
Initially	0	2	0	true
in to the loop	2	4	1	true

i is less than 5 so we enter body of the loop, add *num* to *sum*, add 2 to *num*, and add 1 to *i*.

	sum	***num***	***i***	***i < 5?***
Initially	0	2	0	true
in to the loop	2	4	1	true
in to the loop	6	6	2	true

i is less than 5 so we enter body of the loop, add *num* to *sum*, add 2 to *num*, and add 1 to *i*.

	sum	***num***	***i***	***i < 5?***
Initially	0	2	0	true
in to the loop	2	4	1	true
in to the loop	6	6	2	true
in to the loop	12	8	3	true

i is less than 5 so we enter body of the loop, add *num* to *sum*, add 2 to *num*, and add 1 to *i*.

	sum	***num***	***i***	***i < 5?***
Initially	0	2	0	true
in to the loop	2	4	1	true
in to the loop	6	6	2	true
in to the loop	12	8	3	true
in to the loop	20	10	4	true

i is less than 5 so we enter body of the loop, add *num* to *sum*, add 2 to *num*, and add 1 to *i*.

	sum	***num***	***i***	***i < 5?***
Initially	0	2	0	true
in to the loop	2	4	1	true
in to the loop	6	6	2	true
in to the loop	12	8	3	true
in to the loop	20	10	4	true
in to the loop	30	12	5	false

i is not less than 5 and so looping terminates.

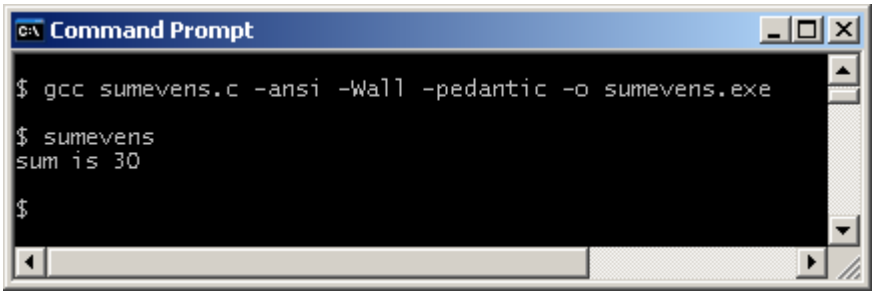
A table such as the last one shown above is an example of a *dry run*. You have variables and Boolean expressions as headings. As you work through the logic line-by-line you write down the values of the variables as they change. Helps you to understand, and get right, the logic for your loops.

Here is the complete program, and its run..

```
/* sumevens.c: sums the first five even numbers */

#include <stdio.h>

int main()
{
  int sum = 0;
  int num = 2;
  int i = 0;
  while (i < 5) {
    sum = sum + num;
    num = num + 2;
    i++;
  }
  printf("sum is %d\n", sum);
  return 0;
}
```



Notice the layout. An opening brace is on the same line as the while. The closing brace is in line immediately below the while. The statements between the opening and closing braces are indented by two spaces and vertically aligned.

9.2 Filters

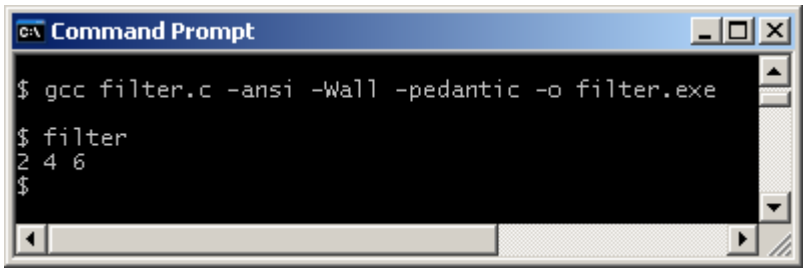
A filter is a bit like a sieve, it lets through a chosen sequence of data values.

A filter is implemented with a selection inside a loop.

```
/* filter.c: selects even integers */

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

int main()
{
  int num = 1;
  int i = 0;
  while (i < 7) {
    if (num % 2 == 0)
      printf("%d ", num);
    num++;
    i++;
  }
  return 0;
}
```



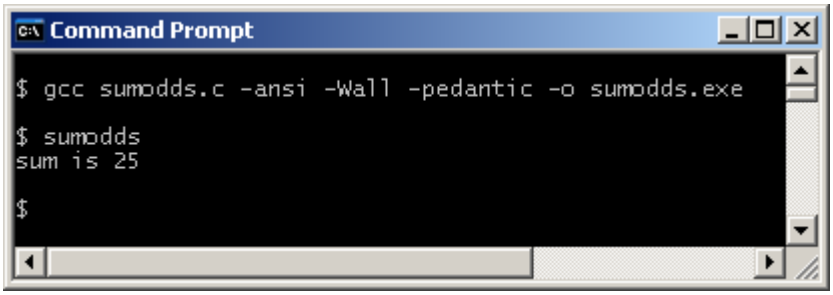
9.3 do ... while Loops

do ... while loops are used when a loop must execute at least once. The next program sums the first five odd numbers. The expected result is 1 + 3 + 5 + 7 + 9 = 25.

```
/* sumodds: uses a do...while loop to sum the first five
   odd integers */

#include <stdio.h>

int main()
{
  int num = 1;
  int sum = 0;
  int i = 0;
  do {
    sum = sum + num;
    num = num + 2;
    i++;
  } while (i < 5);
  printf("sum is %d\n", sum);
  return 0;
}
```



do... while loops are not usually used because the loop control, the Boolean expression, is some way from the top of the loop. And the *do..while* loop must execute at least once - not always the safest thing to do.

Exercise 9.1

1. Dry run each of the program segments shown below:

a.
```
int num = 1;
int i = 0;
while (i < 4) {
  if (num % 2 == 0)
    printf("%d ", num);
  num++;
  i++;
}
```

b.
```
int n = 1;
int sum = 0;
int i = 0;
while (i < 3) {
  sum = sum + n * n;
  n++;
  i++;
}
printf("sum first 3 squares: %d\n", sum);
```

2. Write and test a program that finds the average (arithmetic mean) of the first six positive integers, 1..6. Remember to carefully check that your program delivers the correct answer.

3. Write and test a program that prints the first five odd integers 1, 3, 5, 7, 9.

4. Write and test a program that enters the marks of 5 students, and outputs how many of them scored more than 50.

5. Write and test a program that prints the result of factorial(n) where n is an integer > 0 and input by the user. For example, factorial(4) = 4 x 3 x 2 x 1 = 24.

We have seen how to implement iterations using the *while* statement, how to complete dry runs, and how to implement filters.

Next we see how to program loops using the *for* statement.

Bibliography

Kernighan B and Ritchie D *The C Programming Language* Prentice Hall 1988
Mark Williams Company *ANSI C A Lexical Guide* Prentice Hall 1988