*Chapter 6* While Loops

There are three types of loop or iteration in C++ (and in most other languages):-

- the **FOR** loop

- the **DO** loop

- the **WHILE** loop.

The FOR loop is generally used for repeating statements a known number of times - the number of repetitions is known when the loop is executed the first time. We refer to the FOR loop as **unconditional** or a ***count-controlled loop*** or ***closed*** loop.

The other two loops are used when the number of repetitions is not known at the time the loop is executed the first time - some event occurs during the execution of the loop which causes the loop to terminate. No count is needed. Such a loop is known as **conditional** or an ***event-controlled*** ***loop*** or ***open*** loop.

The main difference between these two conditional loops is that a DO loop tests whether it should continue at the end of the loop - and a WHILE loop makes the test at the start of the loop. The importance of this is that the DO loop will always execute at least once - since only the second and subsequent repetitions are controlled by the continuation test. But the WHILE loop may perhaps never execute - since the test for continuation of the loop is performed immediately the loop is entered.

Look at the following example - it's the old chestnut about summing a series of numbers typed in by the user:-

total = 0;

do

{

cout << "Enter a number:- ";

cin >> number;

if (number != -1)

total = total + number;

} while (number != -1);

Now compare it with the following alternative using a WHILE loop:-

total = 0;

number = 99;

while (number != -1)

{

cout << "Enter a number:- ";

cin >> number;

if (number != -1)

total = total + number;

}

The WHILE version compares poorly here because **number** must be initialised to any value other than -1 to ***force*** the program to enter the loop the first time.

Here is a much better version of the previous code fragment. In this version we have repeated the code to get the user’s input – it appears before the loop (to get the user’s first input) and again at the end of the loop (to get subsequent user inputs).

total = 0;

cout << "Enter a number:- ";

cin >> number;

while (number != -1)

{

total = total + number;

cout << "Enter a number:- ";

cin >> number;

}

This looks rather odd but is in fact a very ***robust*** and useful structure. Its strength is that whenever we read in a user input the next thing we do is check with the WHILE to see whether it is our terminating or ***sentinel value***. So we don’t need to use an IF to jump round processing when the terminator value is entered. And we no longer need to initialise **number** to a spurious value to force entry to the loop.

The program segments are functionally equivalent. Are they equally easy to understand? The last one should be easier because it has only one boolean expression - the others have two (in the post-loop test and in the IF statement).

An advantage offered by the last version is that it allows the loop to be executed **zero times**. In other words if the user quits at the first prompt then we will never enter the loop. This is true of the WHILE command in general and this structure in particular.

The price we have paid for these advantages is that we have duplicated lines of code:-

cout << "Enter a number:- ";

**cin >> number;**

Instinctively this duplication feels wrong to a programmer. But in this case it is worthwhile because of the robustness and flexilibility we have gained.

We could also use the situation to our advantage by altering the first version of these two lines (before the WHILE loop):-

cout << "Enter the first number:- ";

**cin >> number;**

And the second version (inside the WHILE loop) might become:-

cout << "Enter the next number:- ";

**cin >> number;**

In the next example we have developed this fragment of code into a full program which receives user numbers until a value of zero is entered. At that point we show the total of all entries and their average (Chap0601.cpp):-

*// Program to receive numbers until zero is entered.*

*// Then show total and average.*

*// Chap0601.cpp*

…

void main()

{

int userentry;

int totalentry = 0;

int entrycount = 0;

float averentry = 0;

cout << "Zero terminated repetition" << endl << endl;

cout << "Enter first number (0 to quit):- ";

cin >> userentry;

while (userentry != 0)

{

totalentry = totalentry + userentry;

entrycount++;

cout << "Enter next number (0 to quit):- ";

cin >> userentry;

}

if (entrycount > 0) *// Checks that data was entered*

averentry = float(totalentry) / entrycount;

cout << endl << "Total of " << entrycount

<< "entries was " << totalentry;

cout << endl << "Average of " << entrycount

<< " entries was " << averentry;

…

}

Get this program working. Then test it repeatedly and very carefully! Notice what happens when you enter zero at the first prompt.

Deciding which loop structure to use

Remember that there are three types of loop in C++:-

- the **FOR** loop

- the **DO** loop (often called the ***DO-WHILE*** loop – or the Repeat .. Until loop)

- the **WHILE** loop.

FOR loops are the odd-one-out because they are good for counted or closed loops where the number of iterations is already known when execution starts.

DOs and WHILEs are good for open or event-controlled loops where the number of iterations is not known in advance – such as programs running in real-time. The difference between them is that a DO loop’s terminating condition is tested at the end – in the WHILE the condition is tested at the outset. This means that a DO must execute at least once – whereas a WHILE can execute zero times.

Sometimes you will find that you need a loop which will always execute at least once (code which presents menu choices is like that) - and sometimes you will find you need a loop which shouldn't execute even the first time if conditions are right (an error-handling routine for example).

It is up to you to decide on the basis of the program requirements which type of loop you need. If you make the wrong choice you'll end up with fiddly inefficient code - though you can probably persuade it to work!

Here are some guidelines for making the decision as to **which loop to use**:-

1 If the loop is to be count-controlled use a FOR loop.

(Array handling is one of the commonest count-controlled applications.)

2 If the loop must cope with executing zero times (and can’t be count-controlled) use a WHILE loop.

(Although it may seem strange that you would bother writing a loop knowing it might not execute this situation is surprisingly common in real-life problems.)

3 If the loop is to be an event-controlled loop which must execute at least once use a DO loop.

(Menu control loops are usually of this type.)

4 Otherwise use a WHILE loop.

**When in doubt - use a WHILE loop**.

Some definitions*:-*

**Count-controlled** or **bounded** or **closed loop**

A loop where the number of repetitions is set **before** entering the loop

**Event-controlled** or **unbounded** or **open loop**

A loop where something happens **during** the body of the loop to cause termination

**Sentinel**

A special value which is used to **terminate** execution of an event-controlled loop.

**C++ exercises using WHILE loops**

These exercises are familiar problems we have already met with other loops – DO or FOR.

A save name for your program is suggested in each case - where ***XX*** is your initials.

**Test each program thoroughly** before moving on.

6.01 Design and write a program to output the four times table:-

**1 times 4 is 4**

**2 times 4 is 8**

**3 times 4 is 12**

**…**

**10 times 4 is 40**

*Hint:– This is confusing because it’s so simple – the program has no input!*

*XXWHILE01.HTML*

6.02 As program 6.01 but give the user the choice of which table to see (instead of assuming the four times table).

*XXWHILE02.HTML*

6.03 Design and write a program to prompt the user for 5 numbers. Output the total.

*XXWHILE03.HTML*

6.04 As program 6.03 but output the average as well as the total.

*XXWHILE06.HTML*

6.05 Design and write a program which asks the user how many numbers will be input and then prompts for that many numbers. Output the total.

*XXWHILE05.HTML*

6.06Design and write a program to accept a number from the user and then prompt for a Y/N answer whether the user wishes to continue. If the answer is Y then prompt for the next number. When the user answers N then output the total.

*XXWHILE06.HTML*

6.07 Design and write a program to keep prompting the user for input numbers until the user types in a value of zero. Output the total.

*XXWHILE06.HTML*

6.08 Design and write a program to convert inputs in £ sterling to € euros. Assume an exchange rate of £1 = €1.42. Keep converting until the user enters zero.

*XXWHILE08.HTML*

6.09 As program 6.08 but prompt the user (once only) for today's exchange rate.

*XXWHILE09.HTML*

6.10 As program 6.09 but output the total value converted at the end.

*XXWHILE10.HTML*

6.11 Design and write a program to ask about the coins in the user’s possession. Prompt for the value of a coin and then ask whether there are any more coins. If the user enters Y prompt for the next coin. When the user enters N (no more coins) output the total value of the user’s coins.

*XXWHILE11.HTML*

*\*\*\* Make these problems your priority – you can convert them easily from DO solutions*

## Further WHILE exercises

6.12 Design and write a program to simulate a shop till. You should ask for the quantity sold and unit cost for each item. Calculate the total cost for each item and add that to a grand total – which should be displayed when the user enters a quantity of zero.

e.g. Enter quantity of item sold 6

Enter unit cost of item 1.50

Cost for this item 9.00

Enter quantity of item sold 1

Enter unit cost of item 5.00

Cost for this item 5.00

Enter quantity of item sold 0

Total cost 14.00

Make sure that after the final quantity of zero is entered you don’t then prompt the user for the unit price. Although it shouldn’t make any difference to the total bill this would be confusing and misleading for the user.

(Chap0602.cpp)

6.13 Design and write a program to receive the scores of golfers in a club competition. For each competitor read in the gross score and the handicap. Calculate each player’s nett score (by subtracting the handicap from the gross). Input is terminated by entering a negative gross entry. Then show the overall winning score.

You could make the program more sophisticated by asking whether each player is Male (M) or Female (F) and showing two winning scores at the end – one for each category.

This program also lends itself to some practice with simple validation in the form of range-checking. You could set appropriate minimum and maximum gross scores (60 to 100?). Handicaps are in the range 0 to 28 for males and 0 to 36 for females.

6.14 Write a program which reads in a sequence of characters (usually called a ***string***) into a character array, and which then writes the string back to the screen in reverse order. Make the character array big enough to take 10 characters.

Since the string can be any length - from 0 characters up to 10, you'll have to maintain a count of the number of characters read. Use a WHILE loop, and specify that the user input will be terminated by a '**$**' character.

(Chap0603.cpp)

Alternatively you could return to some of the programs you have already written using FOR and DO loops. Try rewriting them using WHILE loops. With a bit ingenuity you should be able to get a WHILE to do anything the others can do – although your code might get a bit messy for some programs.