**Chapter 4 DO Loops**

In a previous approach to adding up a list of numbers, the user was asked how many numbers will be input, so that a FOR loop could be used to input and total them. That's all very well if the user can easily count the number of numbers in the list. But if there were hundreds of numbers, it would be difficult.

The program below takes a different approach to averaging a set of numbers:-

*// Introduces the DO loop*

*// Chap0401*

**…**

**void main()**

**{**

**int mark, total, countMarks;**

**float average;**

**countMarks = 0;**

**total = 0;**

**cout << "Average mark calculator " << endl << endl;**

*// Total the numbers and keep count*

*// until user types a negative number*

**do**

**{**

*// Get next number*

**cout << "Enter a mark (type -1 to finish) :- ";**

**cin >> mark;**

*// Add to total and count if not the end of the list*

**if (mark != -1)**

**{**

**total = total + mark;**

**countMarks++;**

**}**

**} while (mark != -1);**

*// Work out average*

**average = (float) total / countMarks;**

*// Display average and number of marks*

**cout << endl << "The average of the " << countMarks**

**<< " marks was " << average;**

**cout << endl << endl;**

**…**

**}**

Using a DO loop instead of a FOR loop solves the problem of having to know the number of numbers beforehand. In this program the idea is to calculate the average mark in an exam.

What is important here is that the loop does not terminate according to how many times it has executed. It terminates in this instance when the user types in a negative number to indicate the end of the list. There is a slight catch in that we have to make sure that the end-of-list indicator (often known as a ***sentinel***) is not itself added in to the total. So we include the total-and-counting code in an IF statement as shown.

Note that it is easy to choose a sentinel value when working with data which are all within a defined range. In this case marks are assumed to be in the range 0 <= mark <= 100 (between and including 0 and 100). The sentinel value could be any value outside this range (in this case we use the value –1 because presumably nobody will ever get a minus score on this test!). But what happens if we are simply adding up numbers which can take any value? The problem is that any sentinel you might choose could actually be a valid piece of data. In such cases more advanced techniques must be employed - we'll not consider this further just now.

Since the number of inputs is not known in advance, we have to get the program to count the inputs as they are entered, so that we can get the program to work out the average. The variable **no\_of\_marks** is used as a counter for this purpose.

Notice a big difference between the FOR loop and the DO loop is that the condition for determining whether the loop will continue another time is at the end of the loop, rather than at the beginning as for the FOR loop. This practical effect of this is that a **DO LOOP will always execute at least once** ( - unlike other loop structures which might under some circumstances not execute even once). This is very important in many applications - though it is not of particular importance in this particular example.

Finally note that the body of a DO loop is a compound statement.

Try modifying the above program so that any negative number will indicate the end of the list.

Although DO loops are fairly simple to use they are in practice probably the least common of the three iteration structures available in C++. You will meet and use them most often in programs where the user must make a choice from a menu.

For now let's consider another common use of a DO loop – a more advanced example too. We also introduce a new data type - the character type or **char**.

The idea of this next program is to generate, using a built-in function called **random**, two random numbers, display them and ask the user to say what their sum is. The program will itself calculate their sum and compare this with what the user has provided, giving an appropriate message to say whether or not the user is correct (the program will always be correct of course!). We want this program to continue testing until the user indicates he/she doesn't want (or can't stand) any more.

The pseudocode for the program might be:-

**do**

**get two random numbers between 0 and 99**

**add them together**

**display the numbers on the screen for the user to add up.**

**ask user to enter the correct answer**

**check answer and print an appropriate message**

**ask if user wants to continue**

**while user wants to continue.**

*// This program tests the user's ability at addition.*

*// Also includes the random number generator - rand()*

*// Chap0402*

**#include <iostream>**

**#include <time.h>**

**using namespace std;**

**void main()**

**{**

**int first, second;**

**char anothergo;**

**float guess, answer;**

**int highestnumber = 100;** *// Increase for a harder test!*

*// Seed the random-number generator with current time so that*

*// the numbers will be different every time we run the program.*

**srand((unsigned)time(NULL));**

**do**

**{**

*// Allocate random numbers to our two variables*

**first = rand();**

**second = rand();**

*// Now reduce them to the range from 1 to highest*

**first = first % highestnumber;**

**second = second % highestnumber;**

**answer = first + second;**

*// Ask the user to enter an answer*

**cout << endl << "What is " << first**

**<< " + " << second << "? :- ";**

**cin >> guess;**

*// Print an appropriate message*

**if (guess == answer)**

**cout << endl << "Well done";**

**else**

**cout << endl << "Wrong - answer is " << answer;**

*// Ask if user wants to continue*

**cout << endl << endl << "Do you want another go? ";**

**cin >> anothergo;**

**} while (anothergo == 'y');**

**cout << endl;**

**system ("pause");**

**}**

The method shown for generating random numbers is standard, using the '**rand**' function. If you do look up **rand()** in Help then also have a look at '**srand()**'. Where are the prototypes for these functions?

More importantly notice that the DO loop does not repeat a pre-determined number of times. It continues or terminates depending on user input.

Modify the above program so that it reports, at the end of a session, how many correct and incorrect answers were given in the session.

The variable '**anothergo**' is of type '**char**', and will hold a single character. You can guess that variables of type '**char**' traditionally take up one byte of memory.

But it’s no longer necessarily strictly true that a **char** variable occupies a single byte of memory or storage. This could be for two reasons:-

* Broader system architecture (eg 64 bit) mean that it is no longer efficient (or possible to access a single byte – each cycle carries 64 bits ie 8 bytes
* Data is no longer coded in simple ASCII (8 bits - one byte per character) but instead uses Unicode to allow for a much wider range of possible characters from different alphabets – Unicode characters occupy two bytes each – 16 bits

## Try this

Extend the previous Addition Test program so that it offers the choice of testing addition or multiplication. You don't need any new language features, but you will need to be well organised as you amend the previous program.

## Hint

You will probably find it useful to print a copy of the previous program and work on paper - marking where you need to insert new code to offer the multiplication option.

This problem is no longer trivial and most of us find it hard to keep all the details of a complex problem in our heads as we work.

(Chap0403)

## If you have time

See if you can extend the program so that it tests your addition, multiplication, subtraction and division.

(Chap0404)

For further practice with DO loops you could try converting any of the programs you’ve already written using FOR loops. Be sure to save them under new names to avoid losing your originals.

*// This program demonstrates further the use of char variables*

*// It tests a user's ability in addition or multiplication*

*// Also includes the random number generator - rand()*

*// Chap0403*

**…**

**void main()**

**{**

**int first, second;**

**char oper, anothergo;**

**float guess, answer;**

**int highestnumber = 100;** *// Increase for a harder test!*

*// Seed the random-number generator with current time so that*

*// the numbers will be different every time we run.*

**srand((unsigned)time(NULL));**

**cout << "Do you want to test your:-" << endl**

**<< " + Adding " << endl**

**<< " \* Multiplying" << endl**

**<< "Select an operator:- ";**

**cin >> oper;**

**do**

**{**

*// Allocate random numbers to our two variables*

**first = rand();**

**second = rand();**

*// Now reduce them to the range from 1 to highest*

**first = first % highestnumber;**

**second = second % highestnumber;**

**if (oper == '+')**

**answer = first + second;**

**else**

**answer = first \* second;**

**cout << endl << "What is " << first**

**<< " " << oper << " " << second << "? :- ";**

**cin >> guess;**

*// Print an appropriate message*

**if (guess == answer)**

**cout << endl << "Well done";**

**else**

**cout << endl << "Wrong - answer is " << answer;**

**cout << endl << endl << "Do you want another go? ";**

**cin >> anothergo;**

**} while ( anothergo == 'y');**

**cout << endl;**

**}**

*// This program demonstrates a way of handling multiple choice*

*// Also includes the random number generator - rand()*

*// Chap0404*

**…**

**void main()**

**{**

**int first, second, highestnum = 100;**

**char oper, anothergo;**

**float guess, answer;**

**int srand((unsigned)time(NULL));**

**cout << "Do you want to test your:-" << endl**

**<< " + Adding " << endl**

**<< " \* Multiplying" << endl**

**<< " - Subtracting" << endl**

**<< " / Dividing\n\n" << endl**

**<< "Select an operator:- ";**

**cin >> oper;**

**do**

**{**

**first = rand();**

**second = rand();**

*// Adding one prevents zero causing 'divide by zero' error*

**first = first % highestnum + 1;**

**second = second % highestnum + 1;**

**if (oper == '+')**

**answer = first + second;**

**else**

**if (oper == '\*')**

**answer = first \* second;**

**else**

**if (oper == '-')**

**answer = first - second;**

**else**

**answer = (float)first / second;**

**cout << endl << "What is " << first**

**<< " " << oper << " " << second << "? :- ";**

**cin >> guess;**

**if (guess == answer)**

**cout << endl << "Well done";**

**else**

**cout << endl << "Wrong - answer is " << answer;**

**cout << endl << endl << "Do you want another go? ";**

**cin >> anothergo;**

**} while (anothergo == 'y');**

**cout << endl;**

}

**Exercises using DO .. WHILE**

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

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

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

**1 times 4 is 4**

**2 times 4 is 8**

**…**

**10 times 4 is 40**

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

4.02 As program 4.01 but give the user the choice of which table to see (instead of assuming the four times table).

4.03 Design and write a program to prompt the user for 5 scores. Output the total.

4.04 As program 4.03 but output the average score as well as the total.

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

4.06Design and write a program to accept a score 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 score. When the user answers N then output the total score.

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

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

4.09 As program 4.08 but prompt the user (once only) for today's exchange rate.

4.10 As program 4.09 but output the total value converted at the end.

4.11 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.

4.12 As 4.11 above but just prompt for the value of each coin until the user enters a coin value of 0. Then output the total value of the user’s coins.

4.13 **Carpet sales**

Write a program for a **carpet salesperson** who deals with commercial customers.

The salesperson will move through a building  measuring each office and entering the **length and width**. The program should then show the **floor area** (ie amount of carpet needed) for that room.

When all the offices have been measured the user will just enter a length of **0** at the next prompt. The program should then just show the **total floor area** of all the offices measured.

The sequence might look like this for a company with three office rooms – user inputs are shown in **bold**:-

Enter room length **6**

Enter room width **3**

Area of room = *18* sq metres

Enter room length **8**

Enter room width **2**

Area of room = *16* sq metres

Enter room length **7**

Enter room width **3**

Area of room = *21* sq metres

Enter room length **0**

Total area of all rooms = *55* sq metres

You can assume that:-

* all the offices are rectangular
* no carpet is needed in corridors.