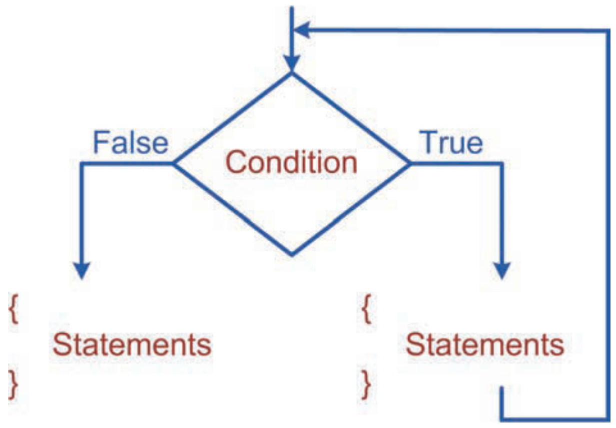
The "while" Loop

The **"while loop"** is the easiest repetition structure to understand for beginners. Its structure suits the most to the description stated above.

Be careful with **infinitive loops**. An infinitive loop executes forever without terminating. Usually incorrectly setting the termination condition or incorrectly increasing or decreasing the loop-control variables (counters) causes an infinitive loop.

The structure is made up of a logical condition and the block of code to be repeated. First, the condition is evaluated, if it is true the statements in the block of the code are executed, and the condition is evaluated one more time. This process continues until the condition becomes false. The condition must become false at the end; otherwise we can fall into an infinitive loop. The **while loop** first checks the condition and then decides whether executes the statements therefore, it is called a **pre-conditional** repetition structure.



Let's make a basic program to understand the **while loop**. The following program prints the numbers from 1 to 10. It prints the value of a variable (that is a counter) one at a time, and repeats this process ten times. To print the numbers from 1 to 10, the variable counter is initialized to one, and its value is increased by 1 in each iteration. On the other hand, the condition "`counter <= 10`" checks if the counter exceeds its final value.

The key concepts of looping are, the **counter**, the **initial value** of the counter, the **final value** of the counter, **increment or decrement factor** of the counter, the **condition** and the **statements** to be executed.

The variable "`counter`" is our counter in this program. Since it gets only the whole numbers from 1 to 10, its type is integer. The initial value of counter is 1, the final value of counter is 10, and the increment factor of the counter is 1.

"`counter <=10`" is the condition of the loop. The **while loop** checks the condition at the header of the structure. This condition states that the statement will be executed while the counter is not bigger than 10.

"`cout <<counter<<" ";`" is the statement that is executed each time in the looping process. The purpose of the program is to print the consecutive numbers from 1 to 10 thus the value of the counter must get those values one by one in each turn.

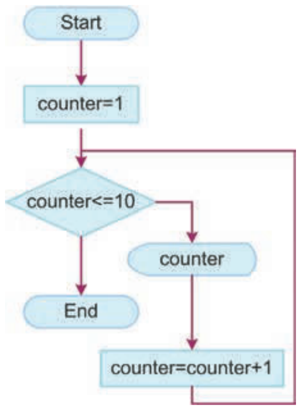
The following three statements are all the same:

```
c++;

c = c+1;

c += 1;
```

"**counter++**" increases the value of counter by one. This statement can be written as "**counter = counter + 1**". C++ used this syntax at the first time. Usually programmers use only 'c' instead of "counter" as a variable name, so "**c++**" means increase the value of a counter "c" by one. I think, this is where the name of this programming language is comes from.

```
/*
PROG: C3_01while.cpp
Printing the numbers from 1 to 10 increasing by 1.
*/
#include <iostream>
using namespace std;
int main()
{
    int counter = 1;                //initialize the counter to its
                                    //starting value
    while (counter <= 10)           //continue looping until counter
                                    //is less than its final value
    {
        cout <<counter<<" ";        //print the current value of the
                                    //counter
        counter++;                  //counter gets its next value
    }
    system("PAUSE"); return 0;
}
```

```
1 2 3 4 5 6 7 8 9 10 Press any key to continue . . .
```



Flowchart of the Program "while"

Increment and Decrement Operators

C++ provides unary **increment** (++) and **decrement** (--) operators. Both operators can be used after (a++, post-increment) or before (++a, pre-increment) their operands. If the variable 'a' needs to be incremented by 1, "a++", "a=a+1", or "a+=1" can be used. The following program demonstrates the use of **increment** and **decrement** operators.

```
/*
PROG: C3_02inc_dec.cpp
Demonstrating increment and decrement operators
*/
#include <iostream>
using namespace std;

int main()
{
  int a=5;
  cout <<"initial a is "<<a<<endl;
  cout <<"after a++ is "<<a++<<endl;
  cout <<"current value of a is "<<a<<endl;
  cout <<"after ++a is "<<++a<<endl;
  cout <<"current value of a is "<<a<<endl;
  cout <<"after --a is "<<--a<<endl;

  system("pause");  return 0;
}
```

```
initial a is 5
after a++ is 5
current value of a is 6
after ++a is 7
current value of a is 7
after --a is 6
Press any key to continue . . .
```

Example: Molecules

Calculate the mass of a molecule. A molecule is a chemical structure that is made up of more then one atom. H_2O is the molecule of water. It contains two hydrogen and one oxygen atoms. The weight of hydrogen is 1, and the weight of oxygen is 16. Thus the weight of water molecule is $2*1 + 16*1 = 18$.

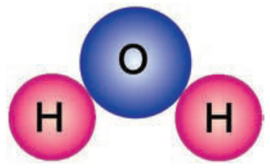
The first line of the input contains an integer (N) that denotes the number of the atoms in the molecule. Each of the following N lines denotes an atom of the molecule with two integers. The first integer is the mass of the atom, and the second integer is the quantity of the atom.

chem.in

```
2
1 2
16 1
```

chem.out

```
18
```



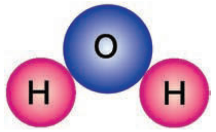
A Water (H_2O) Molecule

```
/*
PROG: c3_03molecules.cpp
*/
#include <fstream>
using namespace std;

int main()
{
  int n, weight, quantity;
  int sum=0, counter=1;

  ifstream fin("chem.in");
  ofstream fout("chem.out");
```

```
fin >> n;         //How many kinds of atoms?

while (counter <= n)
{
  fin >> weight >> quantity;
  sum += weight*quantity;      //add the weight for the new
                               //kind of atom.
  counter++;
}

fout << sum <<endl;

fin.close();
fout.close();

system("pause");
return 0;
```

Exercise: Molecules

Modify the program “Molecules” so that it calculates the weight of a molecule without using any variable as counter.



Exercise: Train

A train leaves the first station with N passengers and visits K stations before it arrives at the last station. In each station some passengers got off the train and some passengers got on the train. Everybody in the train got off at the last station. Make a program that calculates how many passengers got off the train at the last station.

The first line of the input has two integers N and K. Each of the following K lines contains two integers, the first one denotes the number of passengers who got off the train at that station, and the second one denotes the number of passengers got on the train at that station. The output should hava a single integer that is number of passengers who got off the train at the last station.

Input file: train.in

output file: train.out

Sample input

```
5   20
6   15
5   30
20  12
15  8
6   7
```

Sample output

```
40
```

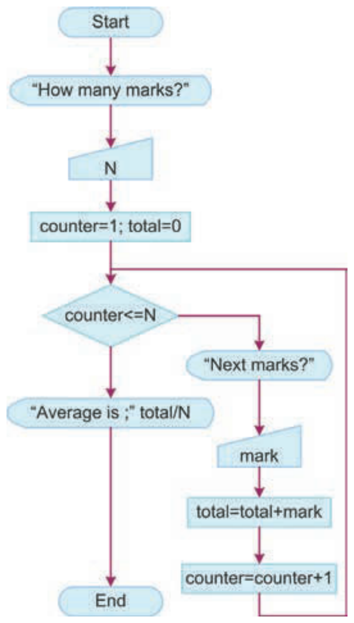
Counter-Controlled and Sentinel-Controlled Repetitions

Counter-Controlled Repetition

Counter-controlled repetition uses a variable called a "counter" to control the number of times the statements will be executed. The variable counter must be initialized before the repetition and must be incremented or decremented during the repetition process. Counter-controlled repetition is called "definite repetition" because the number of repetitions is known before the loop begins executing.

The following program calculates a class average with a counter-controlled loop.

```
/*
PROG: c3_04countercont.cpp
Class average with a counter-controlled loop. Make a program to
calculate average of a class after an exam. There are N students in
the classroom and the marks are between 1 and 5.
*/
#include <iostream>
using namespace std;

int main()
{
  int N, total, mark, counter;

  //Get number of the marks.
  cout<<"How many marks?";
  cin >> N;
  //Initialize the counter and total.
  counter = 1;
  total = 0;

  //Get the mark one by one and add to total.
  while (counter <= N)
  {
    //Get the next mark.
    cout <<"Enter the next mark [1..5]: ";
    cin >> mark;
    total+=mark;           //Add the new mark to total.
```



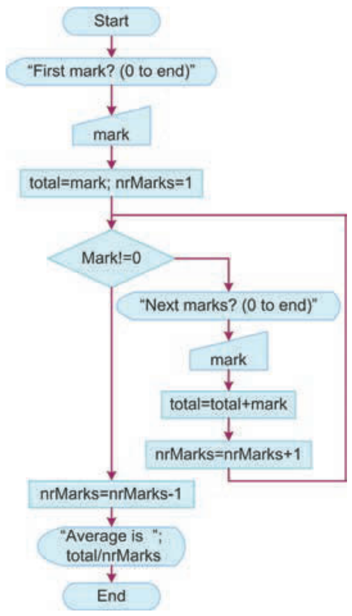
Flowchart of the Program "countercont"

```
    counter++;                      //Increment the counter.
  }

  float average = (float)total/N;  //Calculate the average.

  cout<<"The average is "<<average<<endl;

  system ("pause");
  return 0;
}
```

```
How many marks?5
Enter the next mark [1..5]: 4
Enter the next mark [1..5]: 5
Enter the next mark [1..5]: 3
Enter the next mark [1..5]: 4
Enter the next mark [1..5]: 5
The average is 4.2
Press any key to continue . . .
```

Sentinel-Controlled Repetition

In the case when users don't know the number of the repetitions in advance, a **sentinel-controlled repetition** can be used instead of a counter-controlled repetition. The idea of a **sentinel controlled repetition** is that there is a special value (the "**sentinel**") that is used to say when the loop is done. The sentinel value must be chosen carefully so that it cannot be confused with a legitimate value. **Sentinel-controlled repetition** is called "**indefinite repetition**" because the number of repetitions is not known in advance.



Flowchart of the Program "sentinelcont"

```
/*
PROG: c3_05sentinelcont.cpp
Class average with a sentinel-controlled loop. Make a program to
calculate average of a class after an exam. Your program will
process an arbitrary number of the marks. The marks are between 1
and 5. Use 0 as sentinel value to end the program execution.
*/
#include <iostream>
using namespace std;

int main()
{
  int total, mark, nrMarks;
  //Get the first mark and set the nrMarks to 1.
  cout<<"Enter the first mark: ";
  cin >> mark;
  total = mark;
  nrMarks = 1;      //We have got one mark so far.

  //Get the rest of the marks one by one and add to total.
  while (mark != 0) //Continue if mark is not sentinel
  {
    //Get the next mark.
    cout <<"Enter the next mark [1..5] to finish enter 0: ";
    cin >> mark;
    total+=mark;           //Add the new mark to total.
    nrMarks++;             //Increment the counter.
  }

  nrMarks--;               //Decrease for the sentinel.
  float average = (float)total/nrMarks; //Calculate the average.
  cout<<"The average is "<<average<<endl;
  system ("pause");
  return 0;
}
```

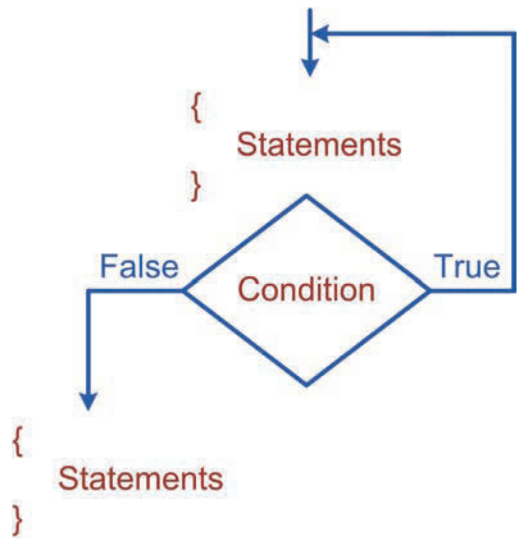
```
Enter the first mark [1..5] to finish enter 0: 5
Enter the next mark [1..5] to finish enter 0: 3
Enter the next mark [1..5] to finish enter 0: 4
Enter the next mark [1..5] to finish enter 0: 5
Enter the next mark [1..5] to finish enter 0: 3
Enter the next mark [1..5] to finish enter 0: 5
Enter the next mark [1..5] to finish enter 0: 0
The average is 4.16667
Press any key to continue . . .
```

Use **setprecision** function to set the precision for floating-point values. Setprecision function is defined in the iomanip library.

```
#include <iomanip>
.
.
.
cout<<"The average is"
<<setprecision(3)
<<average<<endl;
.
.
.
```

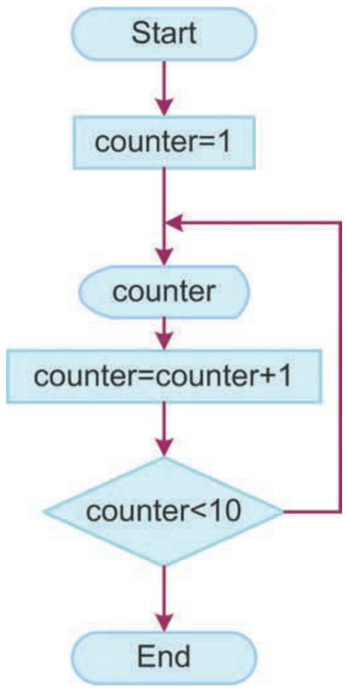
The "do/while" Loop

The "do/while" statement is similar to the "while" statement with an important difference: the "do/while" statement performs a test after each execution of the loop body. Therefore, the loop body is executed at least once. The "do/while" statement is usually preferred to design menu-driven programs.



A **menu-driven program** has an interface that offers the user a simple menu from which to choose an option. The opposite of menu-driven is the **command-driven.** There usually exists one option to exit in the menu-driven programs.

```
Make your choice:
1. New record
2. Modify
3. Delete
4. Print
5. Exit
```

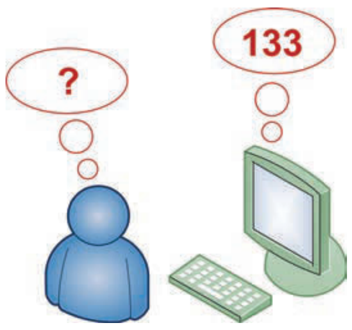


Flowchart of the Program “dowhile”

```
/*
PROG: c3_06dowhile.cpp
Printing the numbers from 1 to 10 increasing by 1.
*/

#include <iostream>
using namespace std;
int main()
{
    int counter = 1;        //initialize the counter to its
                            //starting value

  do
    {
          cout <<counter<<" ";  //print the current value of the
                                //counter
          counter++;            //counter gets its next value
    }
    while (counter <= 10);      //continue looping until counter
                                //is bigger than the final
                                //value.

    system("PAUSE");
    return 0;
}
```

```
1 2 3 4 5 6 7 8 9 10 Press any key to continue . . .
```

Exercise: Guess the number



The **rand** function returns a pseudorandom integer in the range 0 to RAND_MAX (32767).

The game "guess the number" is played between two players. The first player selects an integer at random in the range 1 to 1000. The second player makes his first guess. The first player responds "Too low", "Too high", or "You guessed the number.". The game goes on until the second player guesses the number correctly.

Make a program that plays the game "guess the number" as the first player. Add the following code to your program to generate a random integer between 1 and 1000.

```
#include <time.h>
int main()
{
    srand( (unsigned)time( NULL ) );
    int randomNumber = rand()%1001 + 1;
.
.
.
```