CS 101: Computer Programming and Utilization

Jul-Nov 2016

Bernard Menezes
(cs101@cse.iitb.ac.in)

Lecture 6: General Loops

About These Slides

- Based on Chapter 7 of the book

  An Introduction to Programming Through C++

  by Abhiram Ranade (Tata McGraw Hill, 2014)

- Original slides by Abhiram Ranade

  –First update by Varsha Apte

  –Second update by Uday Khedker

The Need of a More General Loop

Read marks of students from the keyboard and print the average

- Number of students not given explicitly
- If a negative number is entered as marks, then it is a signal that all marks have been entered

  Examples
  - Input: 98 96 -1, Output: 97
  - Input: 90 80 70 60 -1, Output: 75

- The repeat statement repeats a fixed number of times. Not useful
- We need a more general statement

  while, do while, or for

Outline

- The while statement
  - Some simple examples
  - Mark averaging
- The break statement
- The continue statement
- The do while statement
- The for statement

The WHILE Statement

```
while (condition)
        body

next_statement
```

1. Evaluate the condition

   If true, execute body. body can be a single statement or a block, in which case all the statements in the block will be executed

2. Go back and execute from step 1

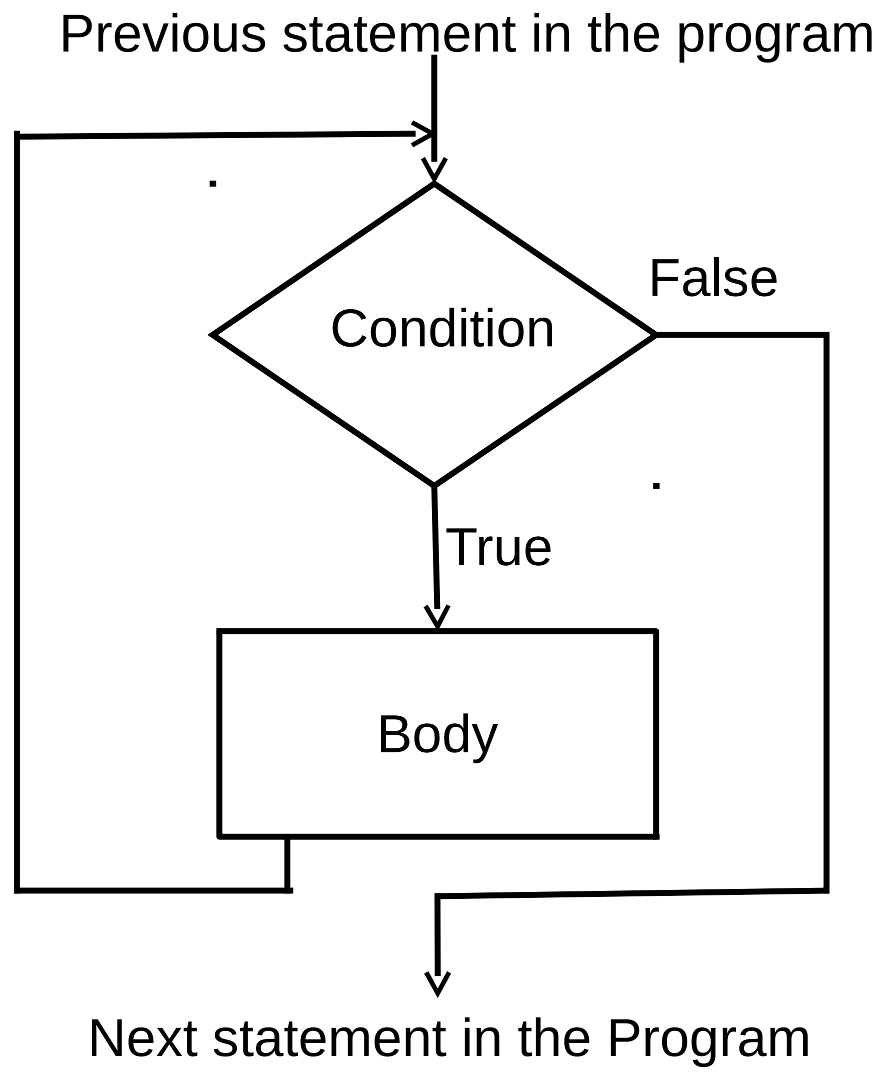
3. If false, execution of while statement ends and control goes to the next statement

The WHILE Statement

```
while (condition)
        body
```

- The condition must eventually become false, otherwise the program will never halt. Not halting is not acceptable
- If the condition is true originally, then the value of some variable used in condition must change in the execution of body, so that eventually condition becomes false
- Each execution of the body = iteration

WHILE Statement Flowchart


Previous statement in the program
Condition
False
True
Body
Next statement in the Program

A Program That Does Not Halt

```
main_program{
    int x=10;
    while(x > 0){
        cout << “Iterating” << endl;
    }
}
// Will endlessly keep printing
// Not a good program
```

A Program That Does Halt

```
main_program{
     int x=3;
     while(x > 0){
          cout << “Iterating” << endl;
          x--; // Same as x = x – 1;
     }
}
// Will print “Iterating.” 3 times
// Good program (if that is what
// you want)!
```

Explanation

```
main_program{
    int x=3;
    while(x > 0){
    cout << "Iterating" <<
endl;
        x--;
    }
}
```

- First x is assigned the value 3
- Condition x > 0 is TRUE
- So body is executed (prints Iterating)
- AFTER x-- is executed, the value of x is 2

Explanation

```
main_program{
    int x=3;
    while(x > 0){
        cout << “Iterating” <<
endl;
          x--;
    }
}
```

- Again the condition is evaluated. For x with value 2, condition is still TRUE
- So execute this
  - print iterating
- Decrement x
- Value now is 1

Explanation

```
main_program{
     int x=3;
     while(x > 0){
          cout << Iterating <<
endl;
          x--;
     }
}
```

- Again the condition is evaluated. For x with value 1, condition is still TRUE
- So execute this
  - print iterating
- Decrement x
- Value now is 0

Explanation

```
main_program{
    int x=3;
    while(x > 0){
        cout << Iterating <<
endl;
        x--;
    }
}
```

- Again the condition is evaluated. For x with value 0, condition is still FALSE
- So control goes outside the body of the loop
- Program exits

WHILE vs. REPEAT

Anything you can do using repeat can be done using while (but not vice-versa)

```
    repeat(n){ any code }
```

Equivalent to

```
    int i=n;
    while(i>0){i--; any code}
```

This is a simplistic explanation

See file include/simplecpp for a more precise explanation

Mark Averaging

Natural strategy

1.Read the next value

2.If it is negative, then go to step 5, if it is >= 0, continue to step 3

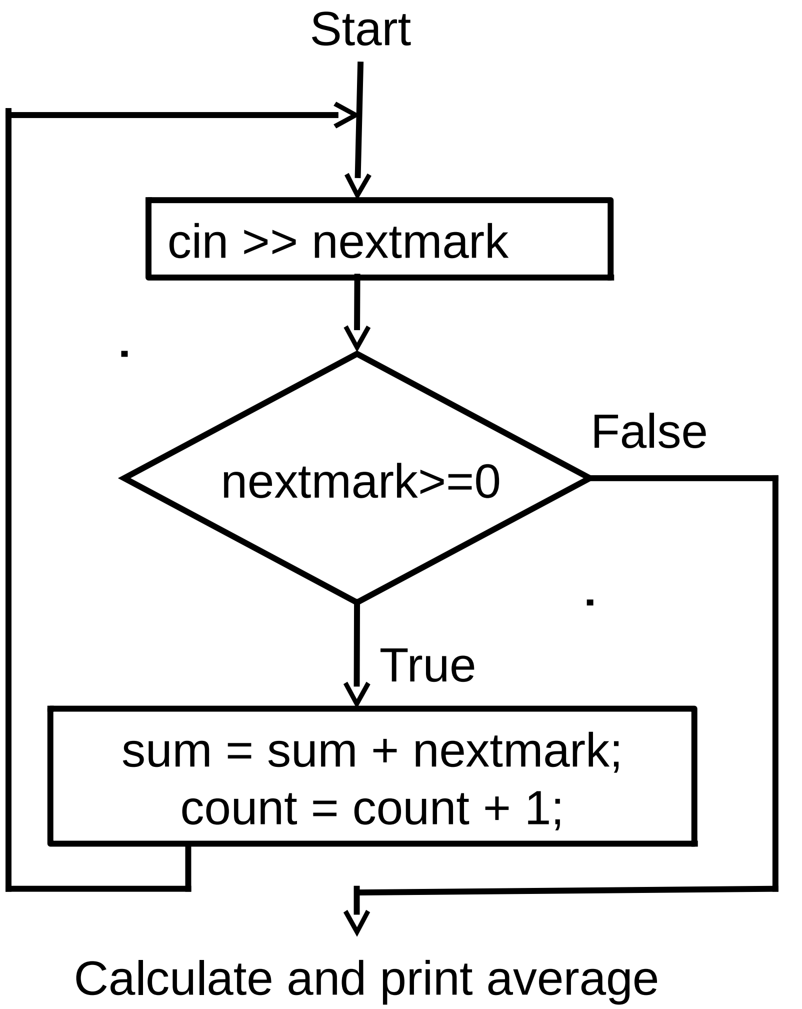
3.Add the value read to the sum of values read so far, Add 1 to the count of values read so far.

4.Go to step 1

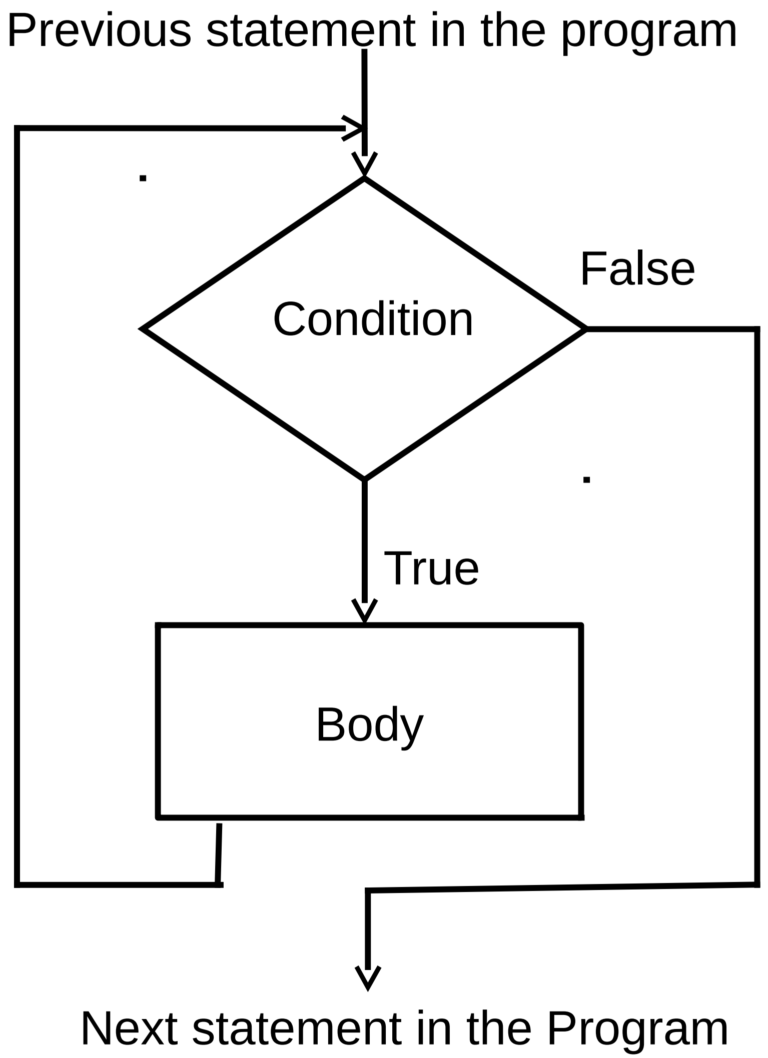
5.Print sum/count

A bit tricky to implement using while

Flowchart Of Mark Averaging vs. Flowchart Of While



Flowchart of mark averaging

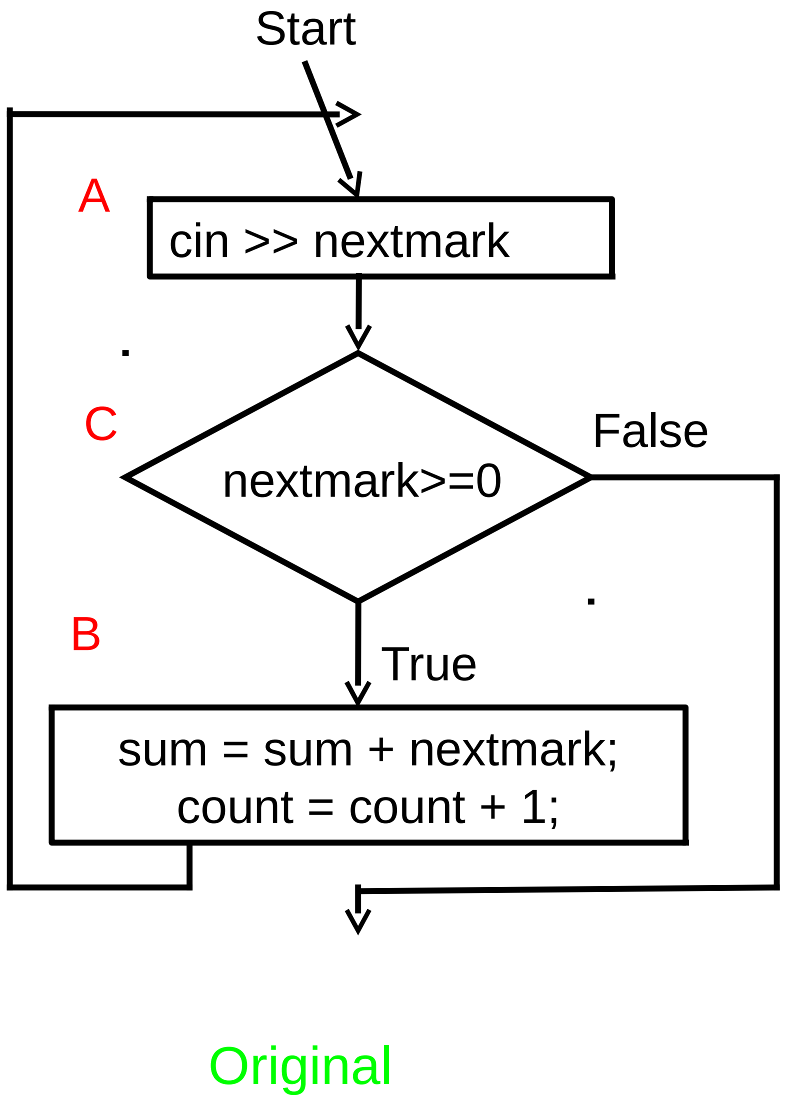

Previous statement in the program
Condition
False
True
Body
Next statement in the Program

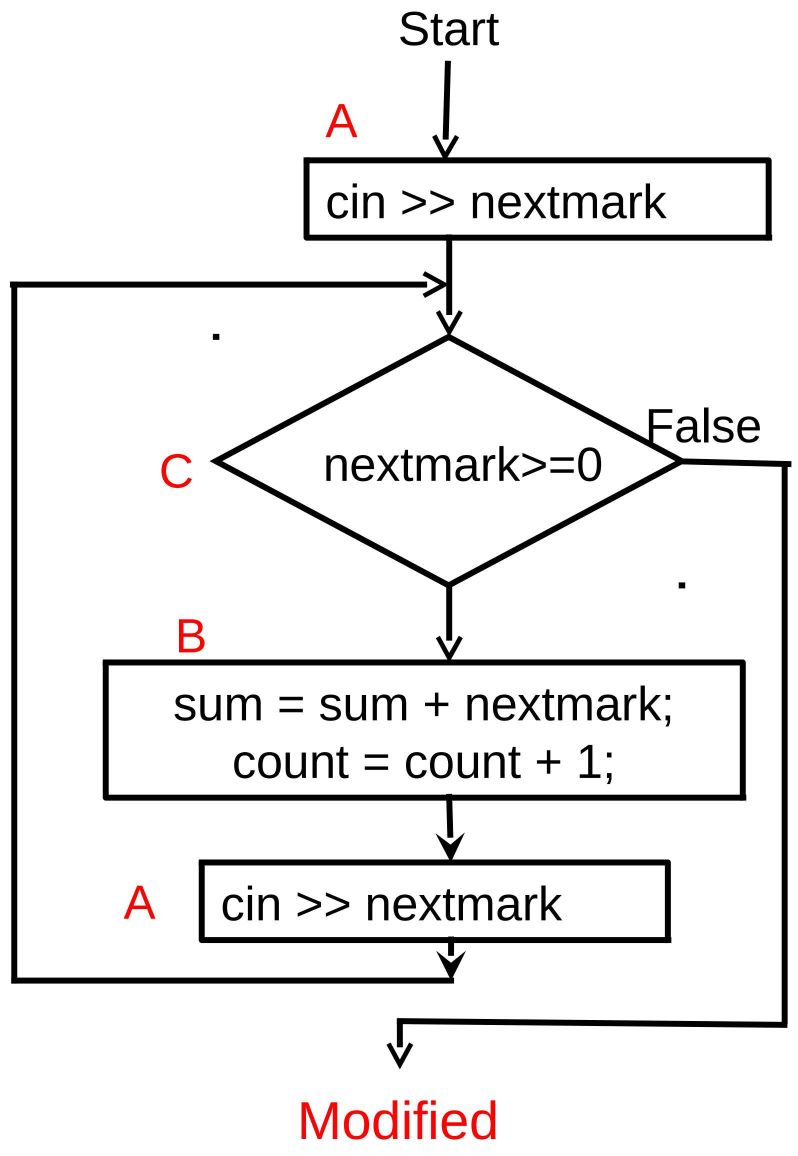
Flowchart of WHILE

Flowchart Of Mark Averaging vs. Flowchart Of WHILE

- In the flowchart of mark averaging, the first statement to be repeated is not the condition check
- In the flowchart of while, the first statement to be repeated, is the condition check
- So we cannot easily express mark averaging using while

Flowchart Of Mark Averaging vs. Flowchart of WHILE





A Different Flowchart For Mark Averaging

- Let's label the statements as A (input), C (condition), and B (accumulation)

- The desired sequence of computation is

  A-C-B   A-C-B   A-C-B ... A-C

- We just rewrite it is

  A   C-B-A   C-B-A   C-B-A ... C

- Thus we take input outside of the loop once and then at the bottom of the loop body

Program

```
main_program{
  float nextmark, sum = 0;
  int count = 0;
  cin >> nextmark;                    // A
  while(nextmark >= 0){
      sum += nextmark; count++;
      cin >> nextmark;                // copy of A!!
  }
  cout << sum/count << endl;
}
```

Remarks

- Often, we naturally think of flowcharts in which the repetition does not begin with a condition check. In such cases we must make a copy of the code, as we did in our example
- Also remember that the condition at the beginning of the while must say under what conditions we should enter the loop, not when we should get out of the loop. Write the condition accordingly
- Note that the condition can be specified as true, which is always true. This may seem puzzling, since it appears that the loop will never terminate. But this will be useful soon..

Nested WHILE Statements

We can put one while statement inside another The execution is as you might expect. Example:

```
int i=3;
while(i > 0) {
    i--;
    int j=5;
    while(j > 0){
        j--;
        cout << “A”;
   }
    cout << endl;
}
```

What do you think this will print?

The BREAK Statement

- The break keyword is a statement by itself
- When it is encountered in execution, the execution of the innermost while statement which contains it is terminated, and the execution continues from the next statement following the while statement

Example of BREAK

```
main_program{
  float nextmark, sum = 0;
  int count = 0;
  while(true){
        cin >> nextmark;
        if(nextmark < 0)
              break;
        sum += nextmark;
        count++;
        }
 cout << sum/count << endl;
}
```

If break is executed, control goes here, out of the loop

Explanation

- In our mark averaging program, we did not want to check the condition at the beginning of the repeated portion
- The break statement allows us just that!
- So we have specified the loop condition as true, but have put a break inside
- The statements in the loop will repeatedly execute; however when a negative number is read, the loop will be exited immediately, without even finishing the current iteration
- The break statement is of course useful in general

The CONTINUE Statement

- continue is another single word statement
- If it is encountered in execution, the control directly goes to the beginning of the loop for the next iteration, skipping the statements from the continue statement to the end of the loop body

Example

Mark averaging with an additional condition :

- if a number > 100 is read, discard it (say because marks can only be at most 100) and continue with the next number. As before stop and print the average only when a negative number is read

Code For New Mark Averaging

```
main_program{
  float nextmark, sum = 0;
  int count = 0;
  while (true){
     cin >> nextmark;
     if(nextmark > 100)   continue;
     if(nextmark < 0)
     break;
     sum += nextmark;
     count++;
  }
  cout << sum/count << endl;
}
```

If executed, the control goes back to condition evaluation

The DO-WHILE Statement

Not very common

Discussed in the book

The FOR Statement: Motivation

- Example: Write a program to print a table of cubes of numbers from 1 to 100

```
nt i = 1;
repeat(100){
 cout << i <<‘ ‘<< i*i*i << endl;
 i++;
}
```

- This idiom: do something for every number between x and y occurs very commonly
- The for statement makes it easy to express this idiom, as follows:

```
for(int i=1; i<= 100; i++)
   cout << i <<‘ ‘<< i*i*i << endl;
```

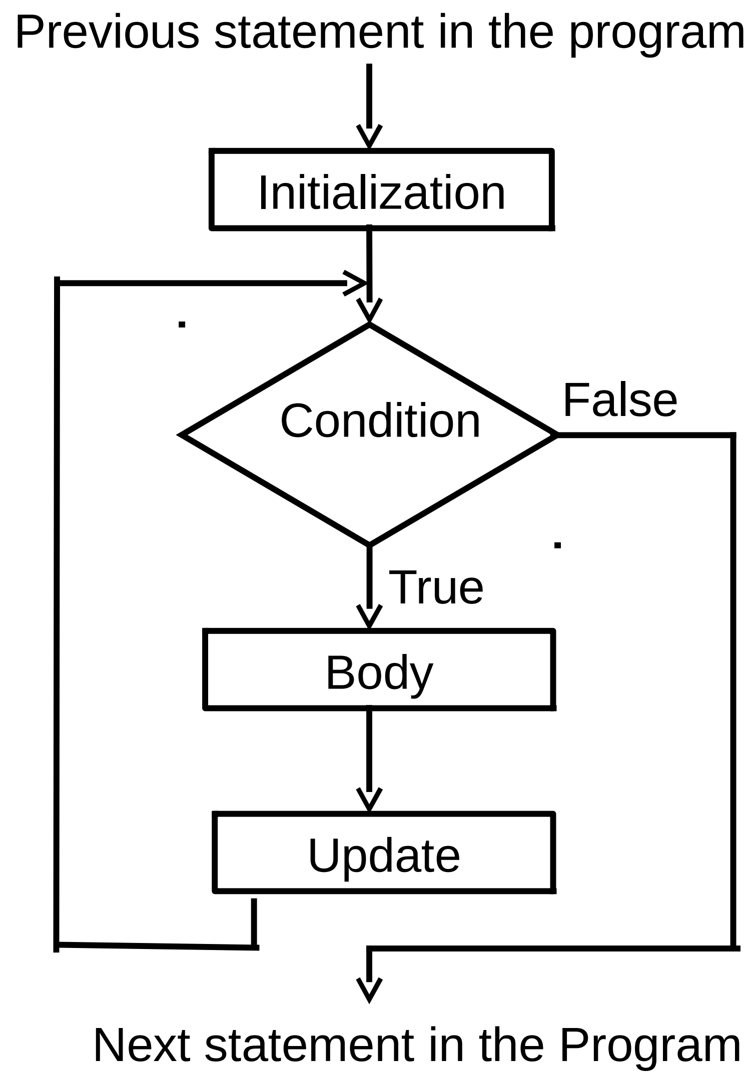
The FOR Statement

for(initialization; condition; update)

  body

- initialization, update : Typically assignments (without semi-colon)
- condition : boolean expression
- Before the first iteration of the loop the initialization is executed
- Within each iteration the condition is first tested. If it fails, the loop execution ends. If the condition succeeds, then the body is executed. After that the update is executed. Then the next iteration begins

Flowchart for FOR Statement


Previous statement in the program
Initialization
Condition
False
True
Body
Update
Next statement in the Program

Definition of Repeat

```
repeat(n)
```

is same as

```
for (int _iterator_i = 0, _iterator_limit = n;
      _iterator_i < _iterator_limit;
      _iterator_i ++)
```

Hence changing n in the loop will have no effect in the number of iterations

Remarks

- New variables can be declared in initialization. These variables are accessible inside the loop body, including condition and update, but not outside
- Variables declared outside can be used inside, unless shadowed by new variables
- Break and continue can be used, with natural interpretation
- Typical use of for: a single variable is initialized and updated, and the condition tests whether it has reached a certain value. Such a variable is called the control variable of the for statement

Remarks

- while, do while, for are the C++ statements that allow you to write loops
- repeat allows you to write a loop, but it is not a part of C++ It is a part of simplecpp; it was introduced because it is very easy to understand.
- Now that you know while, do while, for, you should stop using repeat

Remarks

An important issues in writing a loop is how to break out of the loop. You may not necessarily wish to break at the beginning of the repeated portion. In which case you can either duplicate code, or use `break`