CS 101: Computer Programming and Utilization

Jan-Apr 2017

Sharat
(piazza.com/iitb.ac.in/summer2017/cs101iitb/home)

Lecture 5: Loops

About These Slides

- Based on Chapter 3 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
  - Third update by Sunita Sarawagi

The Need of a More General Loop

Read marks of students from the keyboard and print the average

- Number of students not given explicitly
- Two cases
  1. 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

  2. No such artificial signal

b. The repeat statement repeats a fixed number of times. Not useful

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
```

- 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

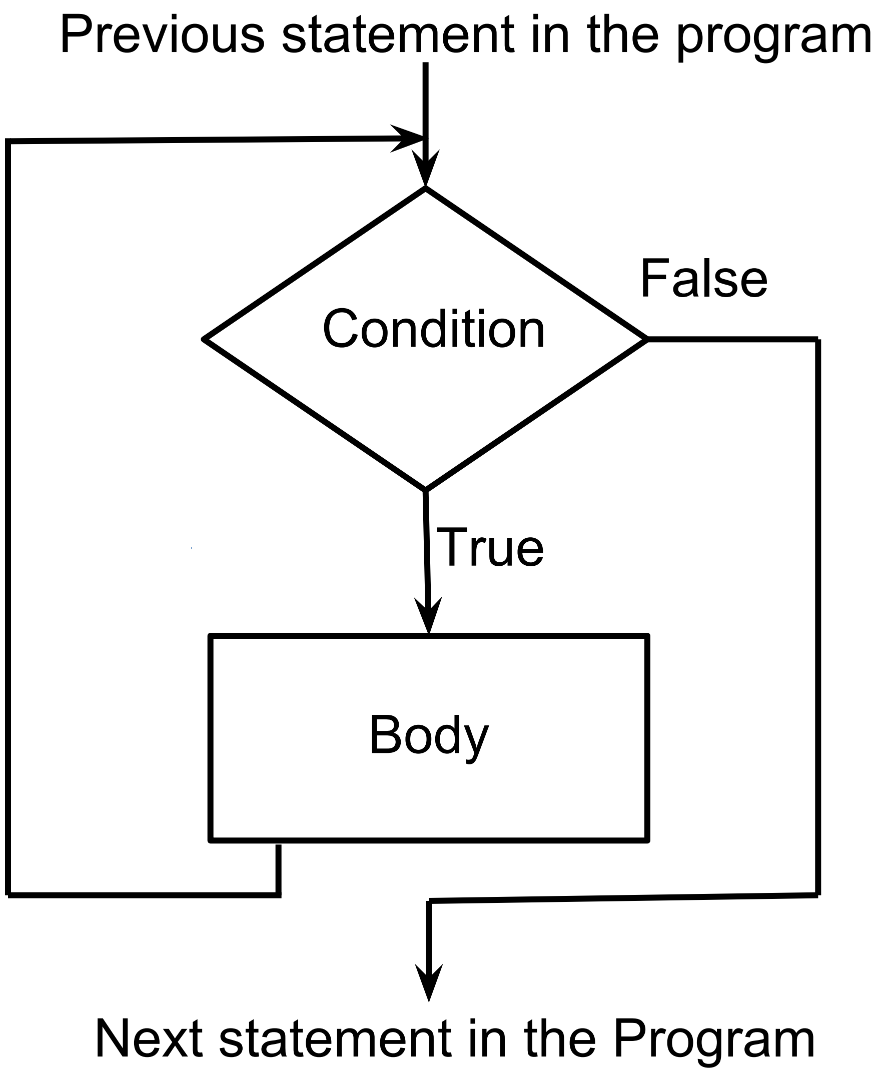
1. Go back and execute from step 1
2. If false, execution of while statement ends and control goes to the next statement

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

Time To Demo

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

Marks Average

Read marks of students from the keyboard and print the average

- Number of students not given explicitly
- Two cases
  1. 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

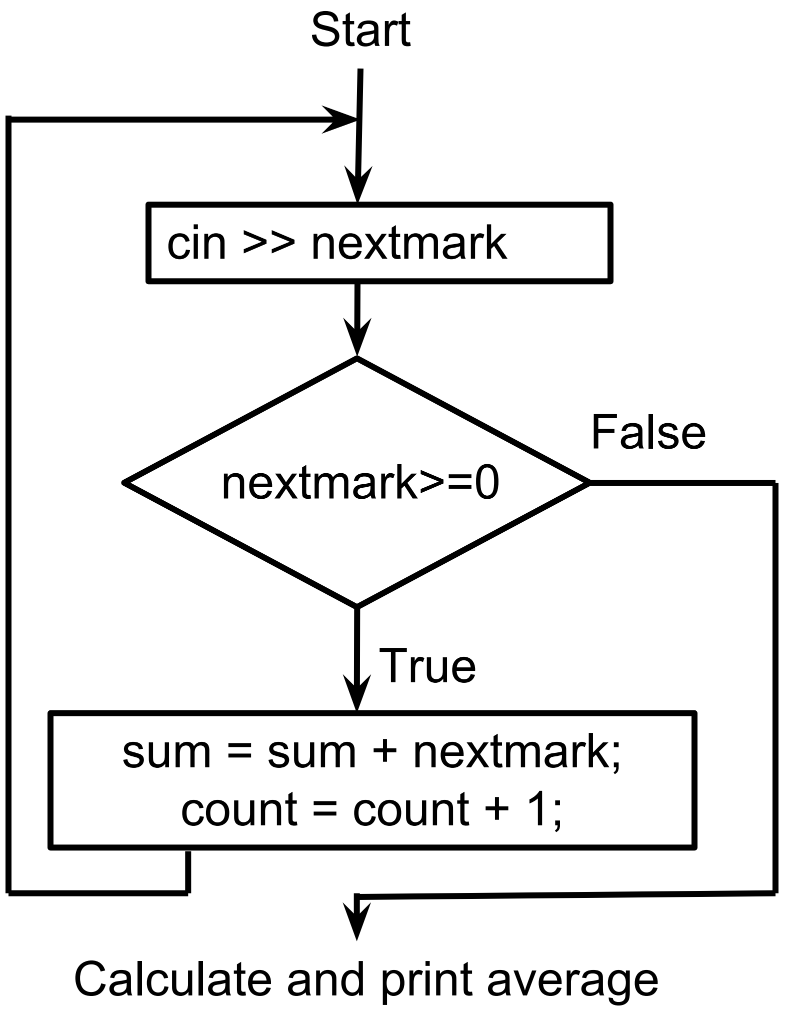
Mark Averaging

Natural strategy

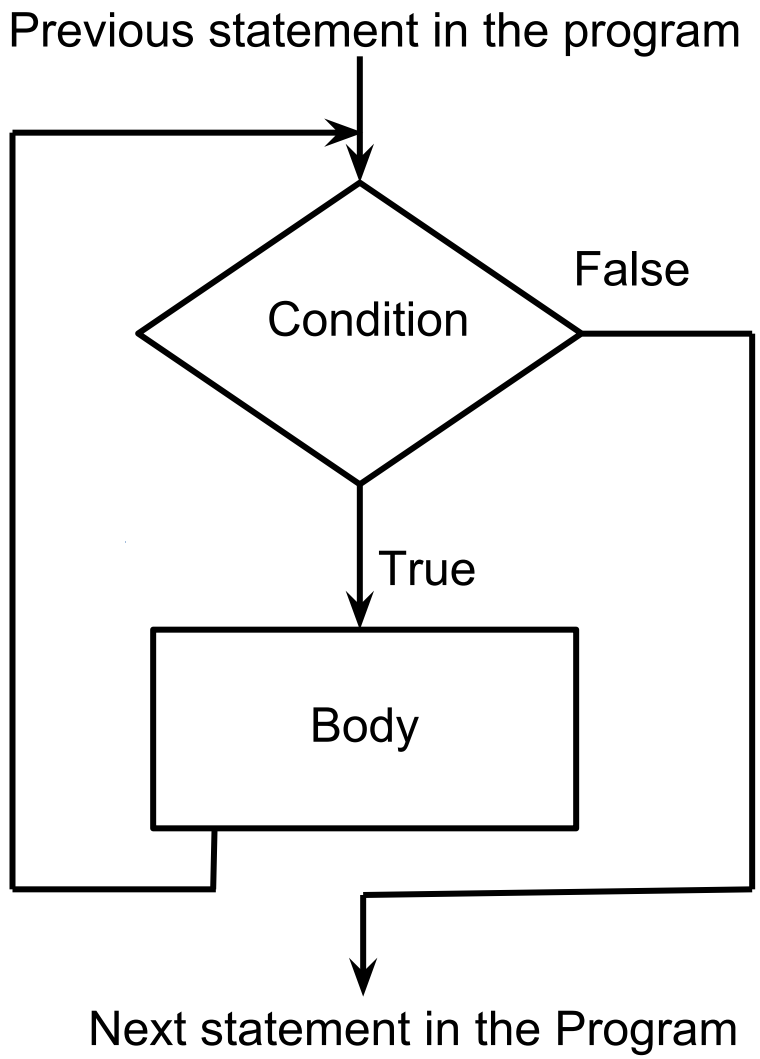
1. Read the next value
2. If it is negative, then go to step 5, if it is >= 0, continue to step 3
3. If it is not negative, add the value read to the sum of values read so far, Add 1 to the count of values
4. Go to step 1
5. Print sum/count

A bit tricky to implement using while

Flowchart Of Mark Averaging vs. Flowchart Of While



Flowchart of mark averaging



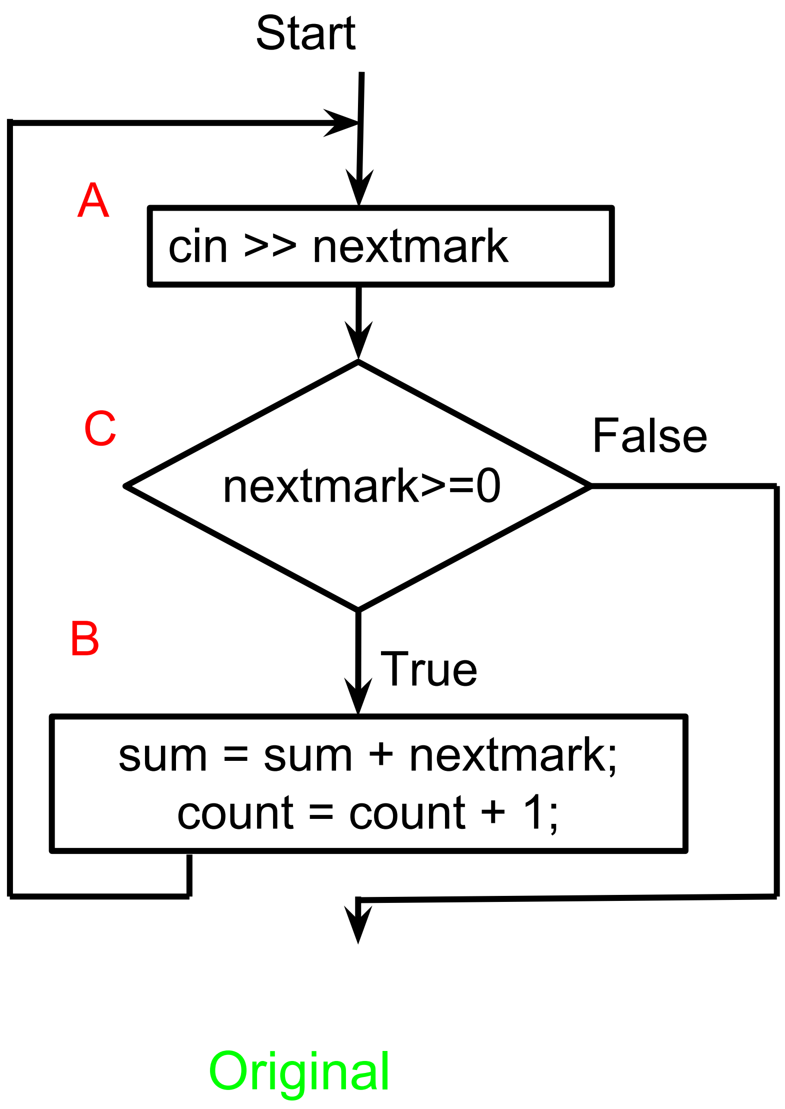
Previous statement in the program
Condition
False
True
Body
Next statement in the Program

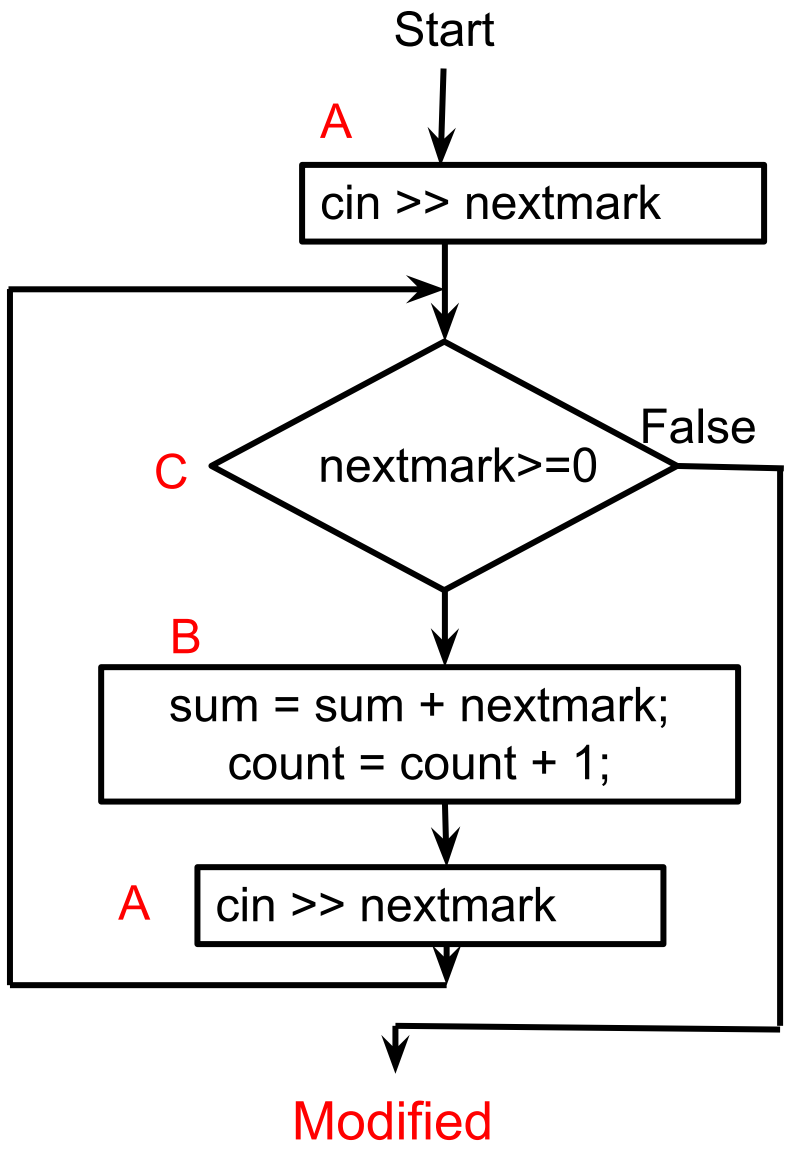
Flowchart of WHILE

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;
}
```

Time To Demo

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.

Marks Averaging

Read marks of students from the keyboard and print the average

- Number of students not given explicitly
- Two cases

  ~~1. If a negative number is entered as marks, then it is a signal that all marks have been entered~~

     – ~~Input: 98 96 -1, Output: 97~~

  2. No such artificial signal. In JEE we have several students who get negative marks :) followed by some who have positive marks

Time To Demo

Key Idea:

```
  while (cin >> nextNumber) {
}
```

The act of taking the next number will tell us whether we are going to have some data or not

Algorithm For GCD (aka HCF)

- Greatest Common Divisor (GCD) of +ve integers m, n:
  largest positive integer p that divides both m, n
- Standard method: factorize m,n and multiply common factors
- Euclid’s algorithm (2300 years old!) is different and much faster
- A program based on Euclid’s method will be much faster than program based on factoring

Euclid's Algorithm

Basic Observation: If d divides both m, n, then d divides m-n also, assuming m > n

Proof: m=ad, n=bd, so m-n=(a-b)d

Converse is also true: If d divides m-n and n, then it divides m too

m, n, m-n have the same common divisors

The largest divisor of m,n is also the largest divisor of m-n,n

Observation: Instead of finding GCD(m,n), we might as well find GCD(n, m-n)

Example

GCD(3977, 943)
=GCD(3977-943,943) = GCD(3034,943)
=GCD(3034-943,943) = GCD(2091,943)
=GCD(2091-943,943) = GCD(1148,943)
=GCD(1148-943,943) = GCD(205, 943)

We should realize at this point that 205 is just 3977 % 943 (repeated subtraction is division)

So we could have got to this point just in one shot by writing GCD(3977,943) = GCD(3977 % 943, 943)

Example

Should we guess that GCD(m,n) = GCD(m%n, n)?

This is not true if m%n = 0.

But we can save the situation, as Euclid did

Euclid's theorem: If m>n>0 are positive integers, then if n divides m then GCD(m,n) = n. Otherwise GCD(m,n) = GCD(m%n, n)

Example Continued

GCD(3977,943)

= GCD(3977 % 943, 943)

= GCD(205, 943) = **GCD (943,205) =** GCD(205, 943%205)

= GCD(205,123) = GCD(205%123,123)

= GCD(82, 123) = GCD(82, 123%82)

= GCD(82, 41)

= 41 because 41 divides 82

Algorithm Our GCD Program

input: values M, N which are stored in variables m, n.

iteration : Either discover the GCD of M, N, or find smaller numbers whose GCD is same as GCD of M, N

Details of an iteration:

- At the beginning we have numbers stored in m, n, whose GCD is the same as GCD(M,N).
- If n divides m, then we declare n to be the GCD.
- If n does not divide m, then we know that GCD(M,N) = GCD(n, m%n)
- So we have smaller numbers n, m%n, whose GCD is same as GCD(M,N)

Program For GCD

```
main_program{
    int m, n; cin >> m >> n;
    while(m % n != 0){
        int nextm = n;
        int nextn = m % n;
        m = nextm;
        n = nextn;
    }
    cout << n << endl;
}
// To store n, m%n into m,n, we cannot
// just write m=n; n=m%n;
// Can you say why?  Hint: take an example!
```

Remark

We have defined variables nextm, nextn for clarity

We could have done the assignment with just one variable as follows

- int r = m%n; m = n; n = r;

It should be intuitively clear that in writing the program, we have followed the idea from Euclid’s theorem. However, having written the program, we should check this again

Time for Demo