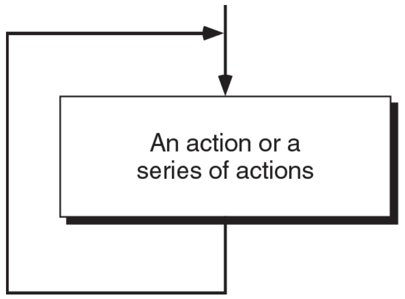
Loop / Repetition

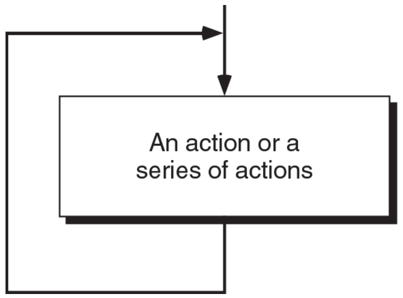
- The main idea of a loop is to repeat an action or a series of actions.



The concept of a loop

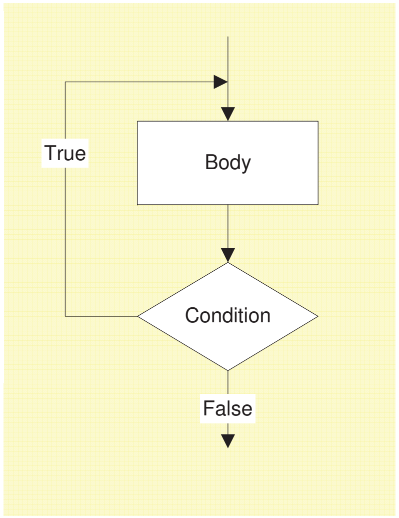
Loops

- But, when to stop looping?
- In the following flowchart, the action is executed over and over again. It never stop - This is called an infinite loop
- Solution - put a condition to tell the loop either continue looping or stop.



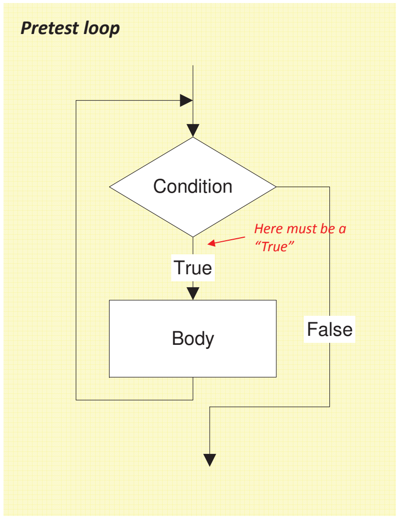
Loops

- A loop has two parts - body and condition
- Body - a statement or a block of statements that will be repeated.
- Condition - is used to control the iteration - either to continue or stop iterating.



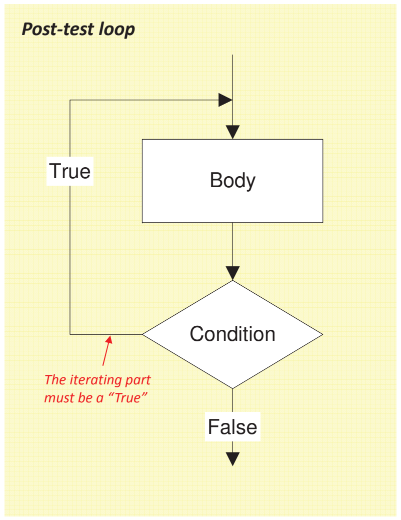
Types of loop

- Two forms of loop - pretest loop and post-test loop.
- Pretest loop
  - the condition is tested first, before we start executing the body.
  - The body is executed if the condition is true.
  - After executing the body, the loop repeats



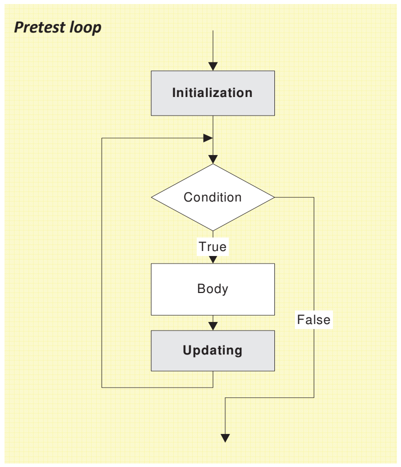
Types of loop

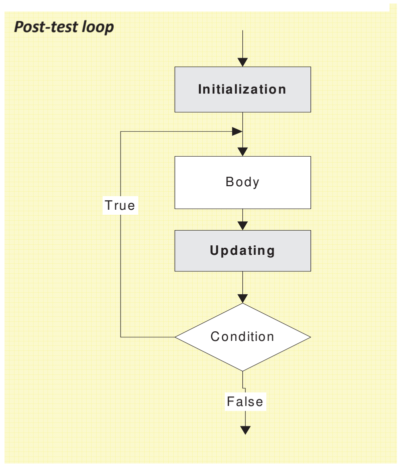
- Post-test loop
  - the condition is tested later, after executing the body.
  - If the condition is true, the loop repeats, otherwise it terminates.
  - The body is always executed at least once.



Parts of a loop

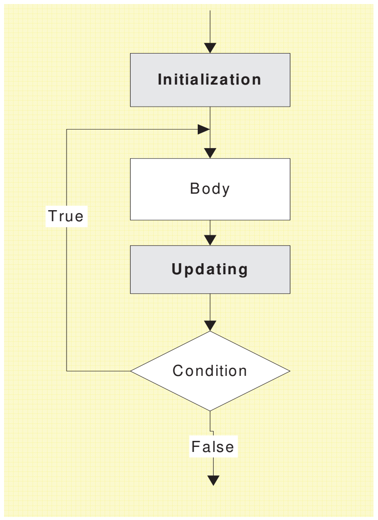
- Beside the body and condition, a loop may have two other parts - Initialization and Updating





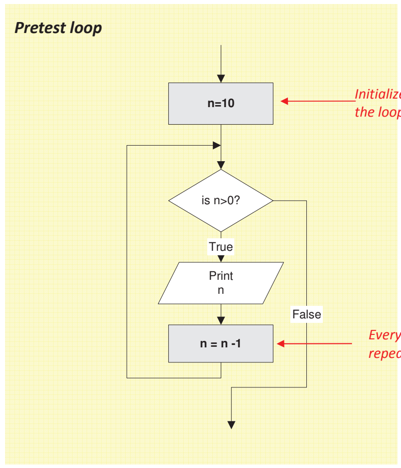
Parts of a loop

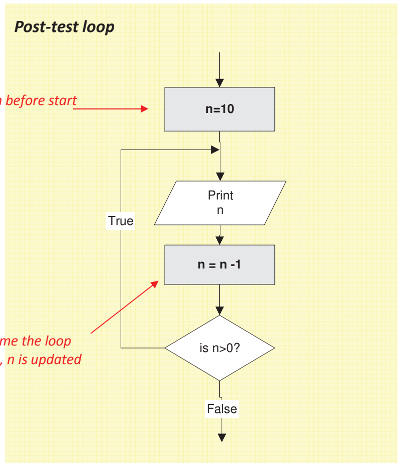
- Initialization
  - is used to prepare a loop before it can start -usually, here we initialize the condition
  - The initialization must be written outside of the loop - before the first execution of the body.
- Updating
  - is used to update the condition
  - If the condition is not updated, it always true => the loop always repeats - an infinite loop
  - The updating part is written inside the loop - it is actually a part of the body.



Parts of a loop

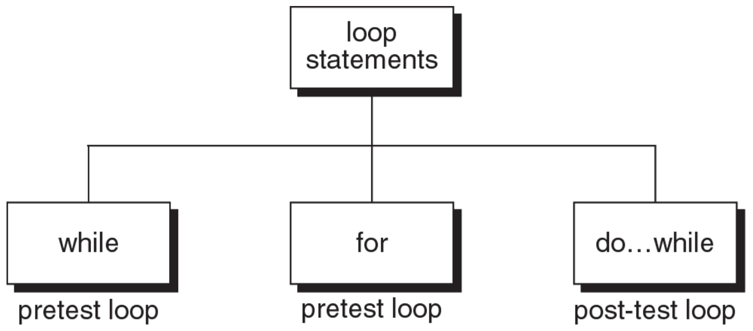
Example: *These flowcharts print numbers 10 down to 1*




Post-test loop
n=10
Print n
True
n = n -1
is n>0?
False
Initialize n before start the loop
Every time the loop repeats, n is updated

Loop statements

- C++ provides three loop statements:

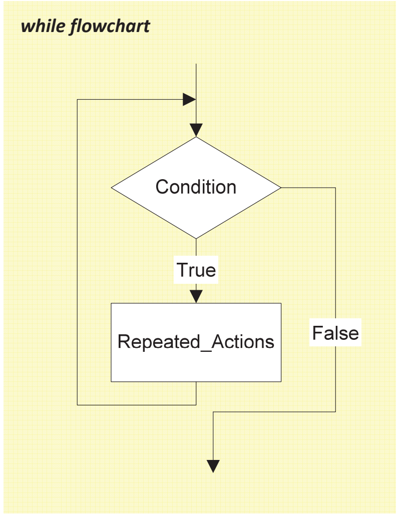


C++ loop constructs

while statement

```
while (Condition)
{
  Repeated_Actions;
}
```

while flowchart



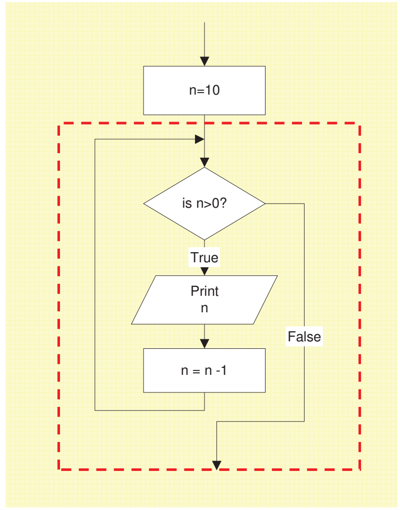
while statement

Example: *This* `while` *statement prints numbers 10 down to 1*

Note that, the first line (n=10) is actually not a part of the loop statement.

```
n=10;
while (n>0)
{
  cout << n <<“ “;
  n=n-1;
}
```

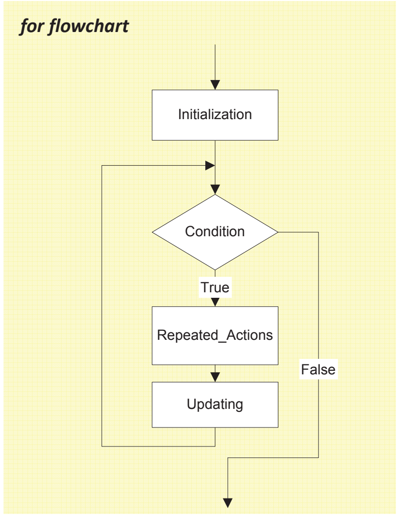
```
Output:

10 9 8 7 6 5 4 3 2 1
```



for statement

```
for (Initialization; Condition; Updating)
{
  Repeated_Actions;
}
```

for flowchart

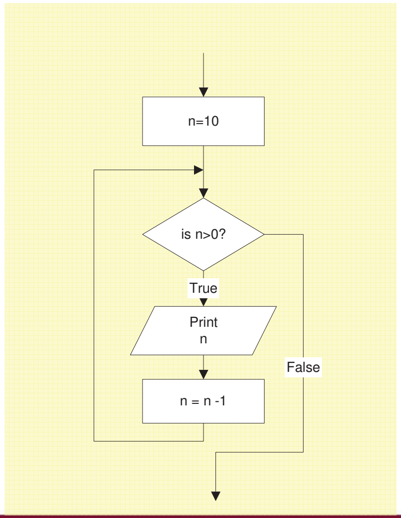


for statement

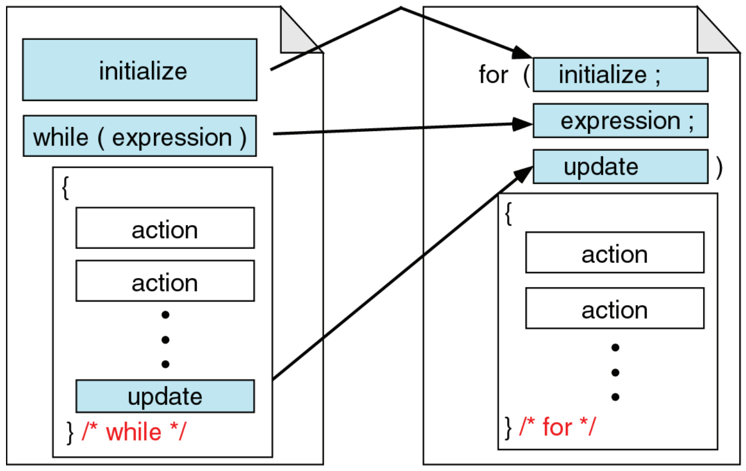
Example: This `for` *statement prints numbers 10 down to 1*

```
for (n=10; n>0; n=n-1)
{
  cout << n <<" ";
}
```

```
Output:

10 9 8 7 6 5 4 3 2 1
```



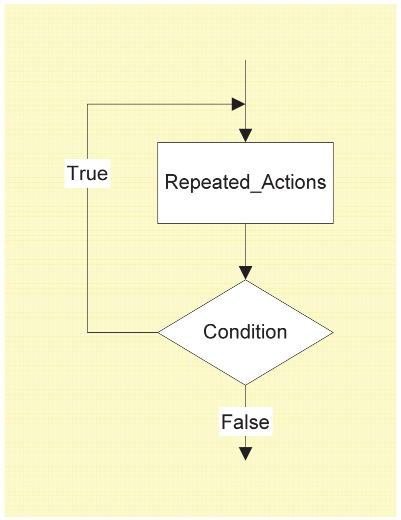
for vs. while statements



Comparing `for` and `while` loops

do...while statement

```
do
{
  Repeated_Actions;
} while (Condition);
```

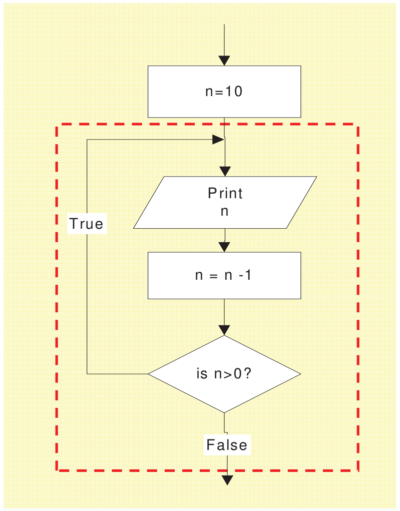


do...while statement

Example: This `do…while` *statement prints numbers 10 down to 1*

Note that, the first line (n=10) is actually not a part of the loop statement.

```
n=10;
do
{
  cout << n << " ";
  n=n-1;
} while (n>0);
```



Loop statements

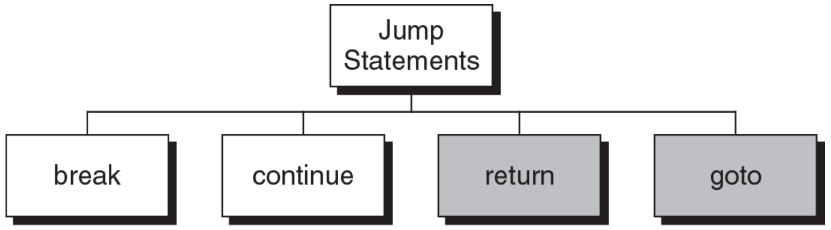
- If the body part has only one statement, then the bracket symbols, { } may be omitted.
- Example: These two `for` statements are equivalent.

```
for (n=10; n>0; n=n-1)
{
 cout << n;
}
```

```
for (n=10; n>0; n=n-1)
  cout << n;
```

Jump statements

- You have learn that, the repetition of a loop is controlled by the loop condition.
- C++ provides another way to control the loop, by using jump statements.
- There are four jump statements:



Breaking Out of a Loop

- Can use **break** to terminate execution of a loop
- Use sparingly if at all – makes code harder to understand
- When used in an inner loop, terminates that loop only and returns to the outer loop

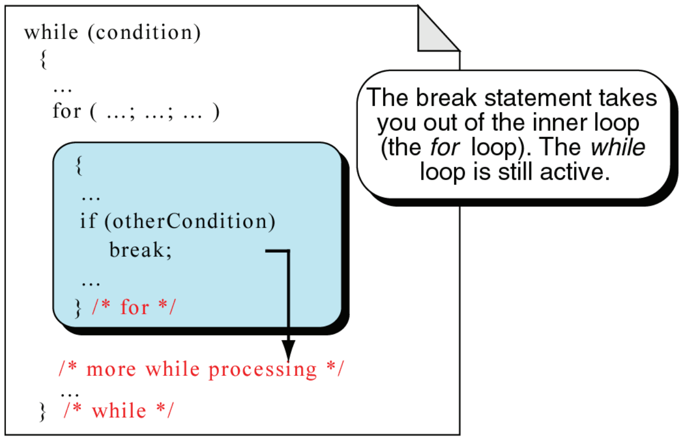
break statement

- It causes a loop to terminate

Example:

```
for (n=10; n>0; n=n-1)
{
 if (n<8) break;
 cout << n << " ";
}
```

break statement



```
while (condition)
 {
   ...
   for ( ...; ...; ... )
     {
      ...
      if (otherCondition)
         break;
      ...
     } /* for */
   /* more while processing */
   ...
 }  /* while */
```

The break statement takes you out of the inner loop (the *for* loop). The *while* loop is still active.

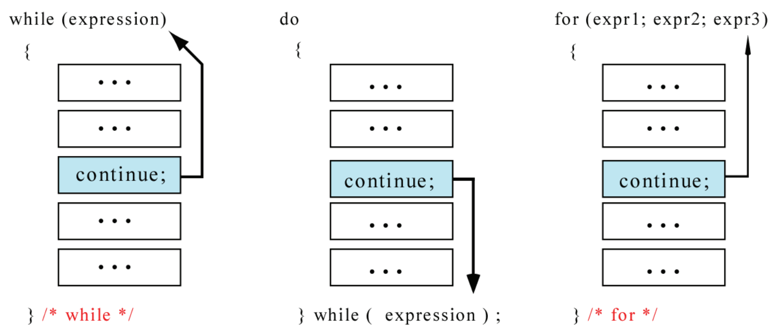
break **an inner loop**

The **continue** Statement

- Can use **continue** to go to end of loop and prepare for next repetition
  - **while** and **do-while** loops go to test and repeat the loop if test condition is true
  - **for** loop goes to update step, then tests, and repeats loop if test condition is true
- Use sparingly – like **break**, can make program logic hard to follow

continue statement

- In `while` and `do…while` loops, the `continue` statement transfers the control to the loop condition.
- In `for` loop, the `continue` statement transfers the control to the updating part.



```
while (expression)        do                        for (expr1; expr2; expr3)
  {                         {                         {
     ...                       ...                       ...
     ...                       ...                       ...
     continue;                 continue;                 continue;
     ...                       ...                       ...
     ...                       ...                       ...
  } /* while */             } while (  expression ) ;  } /* for */
```

The *continue* statement

continue statement

Example:

```
for (n=10; n>0; n=n-1)
{
 if (n%2==1) continue;
 cout << n <<" ";
}
```

`continue` statement

Example:

```
n = 10;
while (n>0)
{
 cout << n << " ";
 if (n%2==1) continue;
 n = n -1;
}
```

`return` statement

- You will learn this statement in Chapter 4 – Function.
- It causes a function to terminate.

Example:

```
void print_numbers()
{ int n=10;
  int i;

  while (n>0)
  {
     for (i=n;i>0; i--)
     {
       if (i%2==1) continue;

       if (i%4==0) break;

       if (n==6) return;

       cout <<i <<" ";
     }
     cout << endl;
     n=n-1;
   }
}
```

`return` statement

- When to use `return`?
- *Example:* the following functions are equivalent

```
float calc_point(char grade)
{
 float result;

 if (grade=='A') result = 4.0;
 else if (grade=='B') result = 3.0;
 else if (grade=='C') result = 2.5;
 else if (grade=='D') result = 2.0;
 else result = 0.0;

 return result;
}
```

```
float calc_point(char grade)
{
 if (grade=='A') return 4.0;
 if (grade=='B') return 3.0;
 if (grade=='C') return 2.5;
 if (grade=='D') return 2.0;
 return 0.0;
}
```

The *else* part of each *if* statement may be omitted. It has never been reached.

`return` statement

```
float calc_point3(char grade)
{
 float result;

 switch (grade)
 {
   case 'A': result = 4.0;
             break;

   case 'B': result = 3.0;
             break;

   case 'C': result = 2.5;
             break;

   case 'D': result = 2.0;
             break;

   default:  result =0.0;
 }

 return result;
}
```

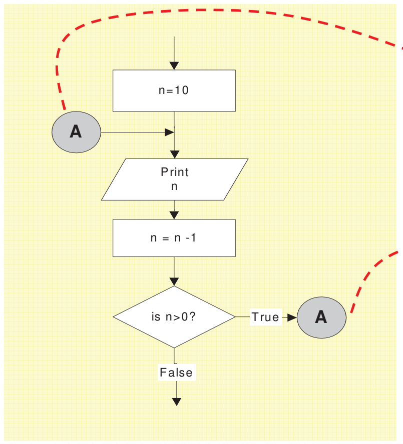
```
float calc_point4(char grade)
{
 switch (grade)
 {
   case 'A': return 4.0;

   case 'B': return 3.0;

   case 'C': return 2.5;

   case 'D': return 2.0;
 }
 return 0.0;
}
```

The *break* statement of each *case* may be omitted. It has never been reached.

goto statement

- It is used to translate connector symbols - jump to another part inside a program.
- But, it is not recommended to use - it may cause unstructured programs.

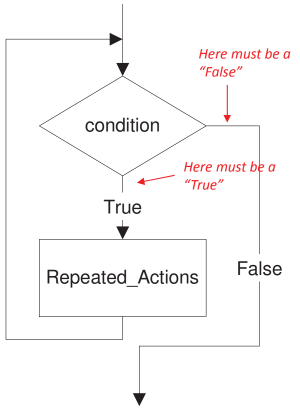
Example:



```
n=10;

A:
   cout <<n <<" ";
   n = n -1;

   if (n>0) goto A;
```

Translating flowchart to C++ code

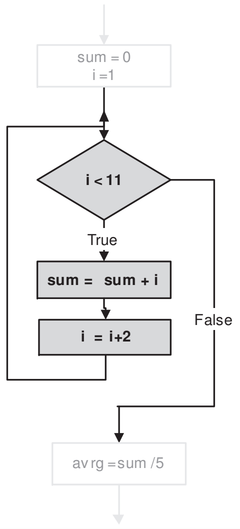
Pattern 1



```
while (condition)
{
 Repeated_Actions;
}
```

Translating flowchart to C++ code

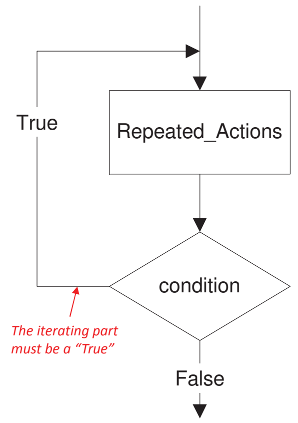
Example: *Calculate the average of odd numbers 1 to 9*



```
sum = 0;
i=1;
while (i<11)
{
 sum = sum + i;
 i = i + 2;
}
avrg = sum/5.0;
```

Translating flowchart to C++ code

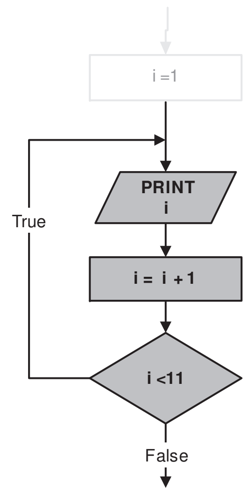
Pattern 2



```
do
{
 Repeated_Actions;
} while(condition);
```

Translating flowchart to C++ code

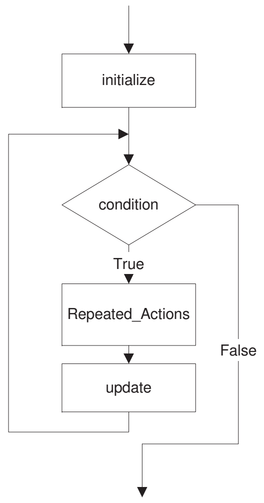
Example: *Prints numbers 1 to 10*



```
i=1;
do
{
 cout <<i <<endl;
 i = i + 1;
} while (i<11);
```

Translating flowchart to C++ code

Pattern 3



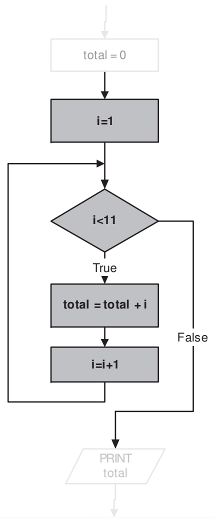
```
for (initialize; condition; update)
{
 Repeated_Actions;
}
```

or

```
initialize;
while (condition)
{
 Repeated_Actions;
 update;
}
```

Translating flowchart to C++ code

Example: *Print the total of numbers 1 to 10*



```
total = 0;
for (i=1; i<11; i++)
{
 total = total + i;
}
cout <<total;
```

or

```
total = 0;
i=1;
while (i<11)
{
 total = total + i;
 i++;
}
cout <<total;
```

Deciding Which Loop to Use

- `while`: pretest loop (loop body may not be executed at all)
- `do-while`: post test loop (loop body will always be executed at least once)
- `for`: pretest loop (loop body may not be executed at all); has initialization and update code; is useful with counters or if precise number of repetitions is known

Nested Loops

- A nested loop is a loop inside the body of another loop
- Example:

outer loop

inner loop

```
for (row = 1; row <= 3; row++)
{
   for (col = 1; col <= 3; col++)
   {
      cout << row * col << endl;
   }
}
```

Notes on Nested Loops

- Inner loop goes through all its repetitions for each repetition of outer loop
- Inner loop repetitions complete sooner than outer loop
- Total number of repetitions for inner loop is product of number of repetitions of the two loops. In previous example, inner loop repeats 9 times

In-Class Exercise

- How many times the outer loop is executed? How many times the inner loop is executed? What is the output?

```
#include <iostream>
using namespace std;
int main()
{   int x, y;
    for(x=1;x<=8;x+=2)
       for(y=x;y<=10;y+=3)
         cout<<"\nx = " <<x << "  y = "<<y;
    system("PAUSE");
    return 0;}
```