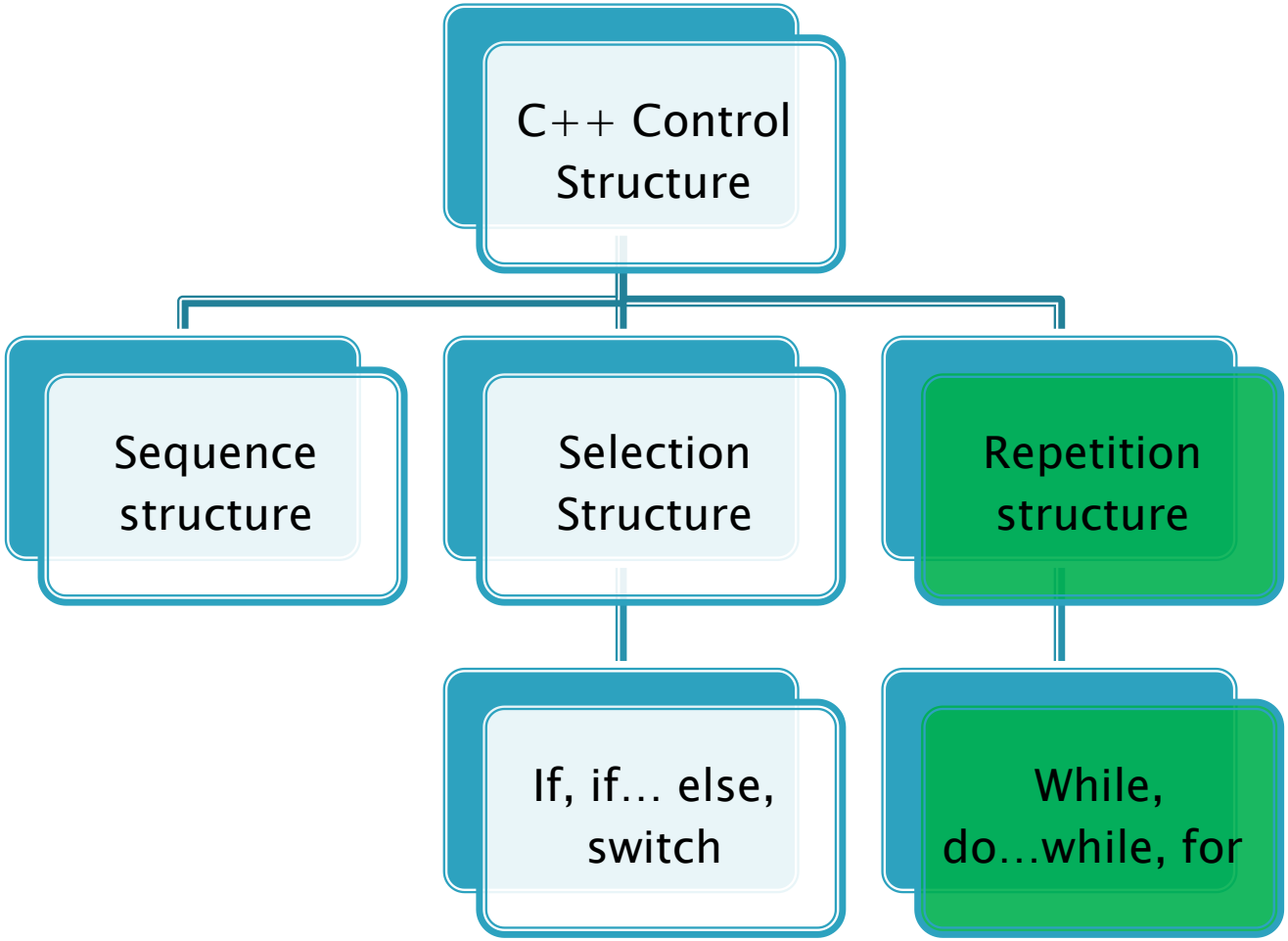
Control Statements

Objectives

- In this chapter you will learn:
  - Essentials of counter-controlled repetition.
  - Use for, while and do … while to execute statements in program repeatedly.
  - Use nested control statements in your program.

Recall: Control structures in C++



Essentials of counter-controlled repetition requires:

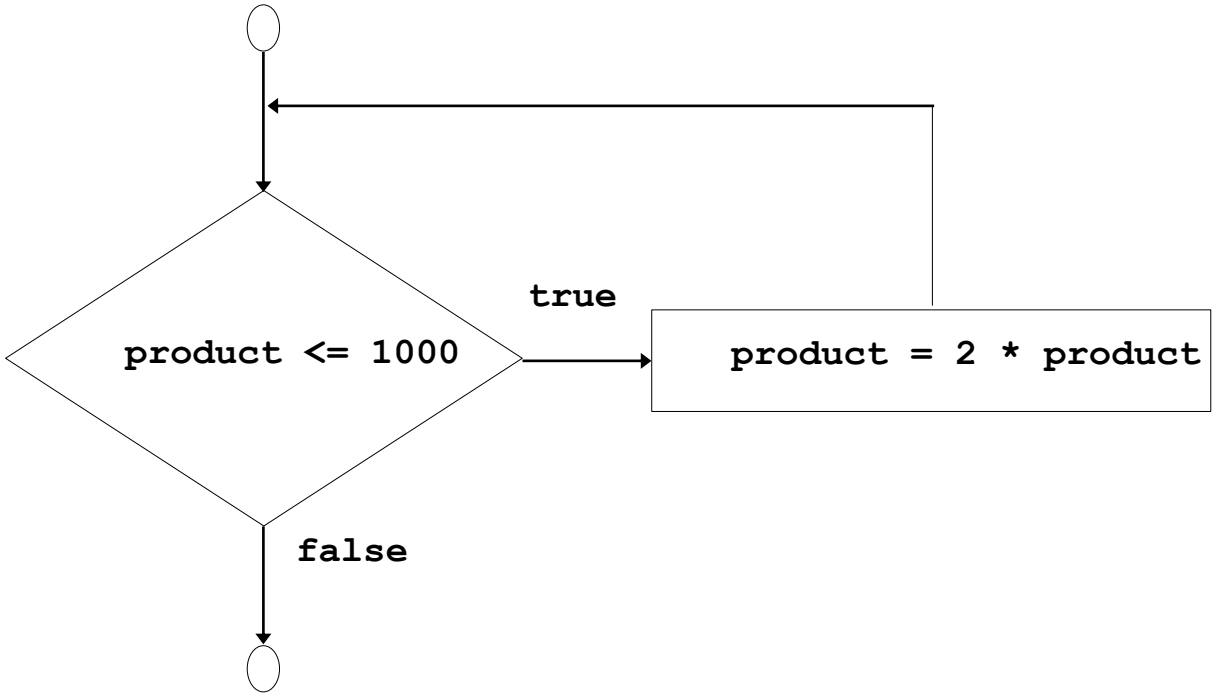
1. The name of a control variable (loop counter)
2. The initial value of the control variable.
3. The loop-continuation condition that test for the final value of the control variable.
4. The increment (or decrement) by which the control variable is modified each time through the loop

While Repetition Structure

- In general any repetition structure
  - Action repeated while some condition remains true
  - Psuedocode
  - **`while`** loop repeated until condition becomes false
- Example

```
int product = 2;
while ( product <= 1000 )
   product = 2 * product;
```

Activity Diagram for while statement



While Repetition Structure

- Syntax:

```
while( expression )
    statement          // loop body
```

- If the controlling expression is true, the loop body is then executed before the controlling expression is evaluated once more.
- If the controlling expression is false, i.e. expression evaluates to false, the program goes on to execute the statement following the while loop.

While Repetition Structure

- If the body of the counter-controlled repetition contains more than one statement, you should surround its body by braces { }.

Example

- Write a c++ programme print numbers from 1 to 10?

Example

```
1
2   // Counter-controlled repetition.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   // function main begins program execution
9   int main()
10  {
11     int counter = 1;              // initialization
12
13     while ( counter <= 10 ) {     // repetition condition
14        cout << counter << endl;   // display counter
15        ++counter;                 // increment
16
17     } // end while
18
19     return 0;   // indicate successful termination
20
21  } // end function main
```

Essentials of counter-controlled repetition requires:

- The declaration

```
int counter = 1;
```

  - Names **`counter`**
  - Declares **`counter`** to be an integer
- Sets **`counter`** to an initial value of **`1`**
- The loop continuation condition determines either the value of the control variable is less than or equal to 10.
- The loop counter is incremented by 1 each time the loop's body is performed.

while example

- Write a pesudocode to find the average of the students grade for five subjects in the class.
- Pseudocode:

Set total to zero

Set grade counter to one

While grade counter is less than or equal to five
Input the next grade
Add the grade into the total
Add one to the grade counter

Set the class average to the total divided by ten
Print the class average

while example

```
#include <iostream>
using namespace std;
int main ()
{

 int total=0;    //sum of the grade entered by user
 int gradecounter=1; //number of the grade to be entered next
 int grade;  //grade value entered by user

 while ( gradecounter <= 5 )  // loop 5 times
 {
    cout << "Enter grade" << endl;
    cin>>grade;  //input next grade
    total=total+grade;  //add grade to total
    gradecounter++;     // increment counter by 1
   }
 cout<<total/5;    //display the average of grades
}
```

for Repetition Structure

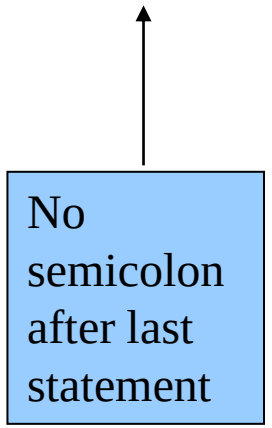
- General format when using **`for`** loops

```
for ( initialization; LoopContinuationTest; increment )
   statement
```

- Example

```
for( int counter = 1; counter <= 10; counter++ )
  cout << counter << endl;
```

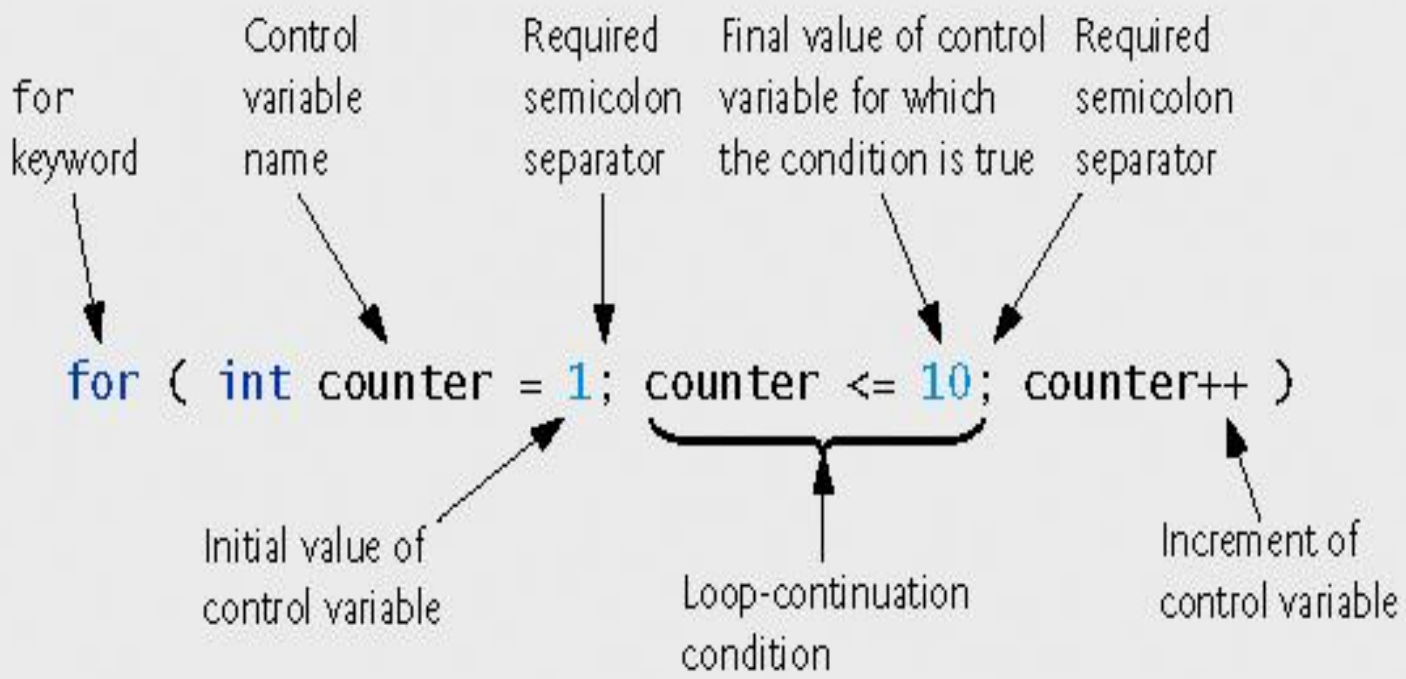
  - Prints integers from one to ten



for Repetition Structure

- As we said in `while`, If you need to repeat more than one statement in a program loop, you must place the statements in a block marked by braces
  { }.

for Repetition Structure



for Repetition example

```
1
2   // Counter-controlled repetition with the for structure.
3   #include <iostream>
4
5   using namespace std;
6
7
8   // function main begins program execution
9   int main()
10  {
11     // Initialization, repetition condition and incrementing
12     // are all included in the for structure header.
13
14     for ( int counter = 1; counter <= 10; counter++ )
15        cout << counter << endl;
16
17     return 0;   // indicate successful termination
18
19  } // end function main
```

for Repetition Structure

- **`for`** loops can usually be rewritten as **`while`** loops

```
initialization;
while (loopContinuationCondition)
{
    statement
    increment;
}
```

- Initialization and increment
  - For multiple variables, use comma-separated lists

```
for (int i = 0, j = 0;  j + i <= 10; j++,i++)
    cout << j + i << endl;
```

Optional expressions in the for statement header

- All the three expressions in the for statement header are optional .
  - The two semicolon are required.
- Omitting the *loopContinuationCondition*:
  - C++ assumes that the condition is true.
- Omitting the initialization expression:
  - The counter variable must be initialized earlier in the program.
- Omitting increment expression:
  - The increment is calculated by statement in the body.

Examples:

```
int count;
for( count = 1; count <= 10; ++count)
    cout << count
         << ". loop" << endl;
```

```
for( int i = 0; i < 10; cout << i++ )
    ;
```

```
for(;;)
```

```
for( ; expression; )
```

The counter variable

- If the initialization expression declares the control variable , the control variable can be used only in the body of the for statements.
- This is what we called variable scope.

Example:

```
1
2   // Summation with for.
3   #include <iostream>
4
5   using namespace std;
6
7
8   // function main begins program execution
9   int main()
10  {
11     int sum = 0;                        // initialize sum
12
13     // sum even integers from 2 through 100
14     for ( int number = 2; number <= 100; number += 2 )
15        sum += number;                   // add number to sum
16
17     cout << "Sum is " << sum << endl;   // output sum
18     return 0;                           // successful termination
19
20  } // end function main
```

```
Sum is 2550
```

Tracing the above example:

number	sum += number	sum
2	0+=2	2
4	2+=4	6
6	6+=6	12
8	12+=8	20
10	20+=10	30
....	....	....
100	2450+=100	2550

Examples Using the for Statement

- Vary control variable from 1 to 5 in increments of 1
  - `for ( int i = 1; i <= 5; i++ )`
- Vary control variable from 5 to 1 in decrements of 1
  - `for ( int i = 5; i >= 1; i-- )`
- Vary control variable from 7 to 77 in steps of 7
  - `for ( int i = 7; i <= 77; i += 7 )`
- Vary control variable from 20 to 2 in steps of -2
  - `for ( int i = 20; i >= 2; i -= 2 )`
- Vary control variable over the sequence: 2, 5, 8, 11, 14, 17, 20
  - `for ( int i = 2; i <= 20; i += 3 )`
- Vary control variable over the sequence: 99, 88, 77, 66, 55, 44, 33, 22, 11, 0
  - `for ( int i = 99; i >= 0; i -= 11 )`

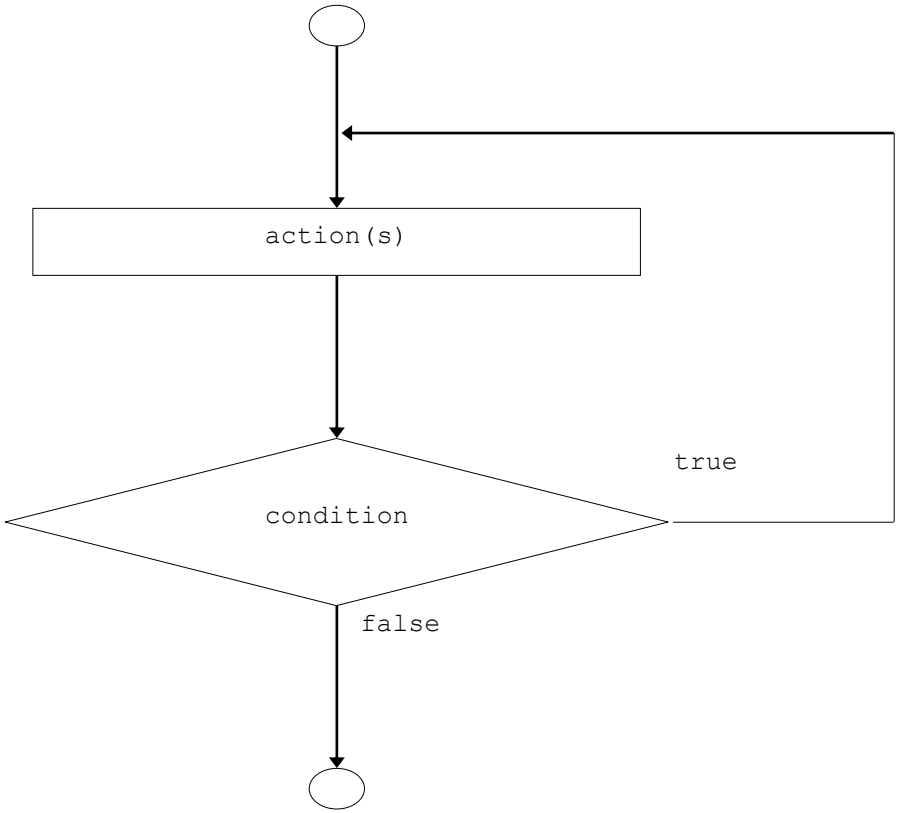
do… while Repetition Structure

- The do…while repetition statement is similar to the while statement.
- In the while:
  - The loop continuation condition test occurs at the beginning of the loop before the body of the loop executes.
- In the do … while:
  - The loop continuation condition test occurs after the loop body executes.
  - The loop body always executes at least once.
  - Recommended to use braces in the do.. While to avoid confusing with while statements.

do…while

- Format

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



do…while example

```
1.  #include <iostream>
2.  using namespace std;
3.  int main(){
4.      int counter=1; //initialize counter
5.      do
6.      {
7.          cout<<counter<<" "; // display counter
8.          counter++;  // increment counter
9.      }
1.  while(counter<=10); //end do...while
10.     cout<<endl;
11.     return 0;
12. }
```

Output :

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

do…while example

```
1
2   // Using the do/while repetition structure.
3   #include <iostream>
4
5   using std::cout;
6   using std::endl;
7
8   // function main begins program execution
9   int main()
10  {
11     int counter = 1;              // initialize counter
12
13     do {
14        cout << counter << "  ";   // display counter
15     } while ( ++counter <= 10 );  // end do/while
16
17     cout << endl;
18
19     return 0;   // indicate successful termination
20
21  } // end function main
```

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

Nesting Loops

- Loops can be nested, that is, the loop body can also contain a loop.

Nested Control Structures

- Problem statement

  A college has a list of test results (1 = pass, 2 = fail) for 10 students. Write a program that analyzes the results. If more than 8 students pass, print "Raise Tuition".

- Notice that
  - Program processes 10 results
    - Fixed number, use counter-controlled loop
  - Two counters can be used
    - One counts number that passed
    - Another counts number that fail
  - Each test result is 1 or 2
    - If not 1, assume 2

Nested Control Structures cont..

- Top level outline

  Analyze exam results and decide if tuition should be raised

- First refinement

  Initialize variables

  Input the ten quiz grades and count passes and failures

  Print a summary of the exam results and decide if tuition should be raised

- Refine

  Initialize variables:

  Initialize passes to zero

  Initialize failures to zero

  Initialize student counter to one

Nested Control Structures cont..

- Refine

Input the ten quiz grades and count passes and failures:

While student counter is less than or equal to ten

Input the next exam result

If the student passed

Add one to passes

Else

Add one to failures

Add one to student counter

Nested Control Structures cont..

- Refine

  Print a summary of the exam results and decide if tuition should be raised:

  Print the number of passes

  Print the number of failures

  If more than eight students passed

  Print "Raise tuition"

- Program next

Nested Control Structures cont(code)

```
2  // Analysis of examination results.
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  // function main begins program execution
10 int main()
11 {
12    // initialize variables in declarations
13    int passes = 0;           // number of passes
14    int failures = 0;         // number of failures
15    int studentCounter = 1;   // student counter
16    int result;               // one exam result
17
18    // process 10 students using counter-controlled loop
19    while ( studentCounter <= 10 ) {
20
21       // prompt user for input and obtain value from user
22       cout << "Enter result (1 = pass, 2 = fail): ";
23       cin >> result;
24
```

Nested Control Structures cont(code)

```
25          // if result 1, increment passes; if/else nested in while
26          if ( result == 1 )          // if/else nested in while
27             passes = passes + 1;
28
29          else  // if result not 1, increment failures
30             failures = failures + 1;
31
32          // increment studentCounter so loop eventually terminates
33          studentCounter = studentCounter + 1;
34
35       } // end while
36
37       // termination phase; display number of passes and failures
38       cout << "Passed " << passes << endl;
39       cout << "Failed " << failures << endl;
40
41       // if more than eight students passed, print "raise tuition"
42       if ( passes > 8 )
43          cout << "Raise tuition " << endl;
45       return 0;   // successful termination
47    } // end function main
```

Nested Control Structures cont(output)

```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Passed 6
Failed 4
```

Nested Control Structures cont(output)

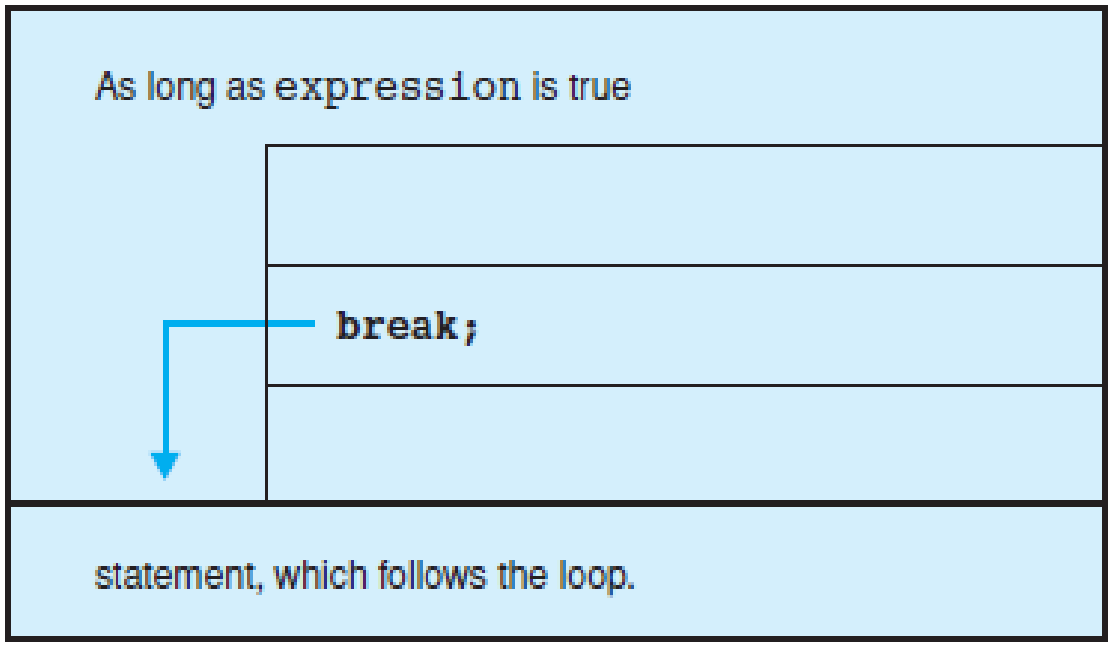
```
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 2
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Enter result (1 = pass, 2 = fail): 1
Passed 9
Failed 1
Raise tuition
```

Break

- The break statement exits from a switch or loop immediately.
- You can use the break keyword to stop a loop for any reason.

Break cont..

Structogram for `break` within a `while` statement



Break cont(example)

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.   int i = 0;
6.      do
7.      {  i++;
8.                  cout<<"before the break\n";
9.         break;
10.                 cout<< "after the break, should never print\n";
11.      } while (i < 3);
12.       cout<< "after the do loop\n";
13.       system("pause");
14.       return 0;
15. }
```

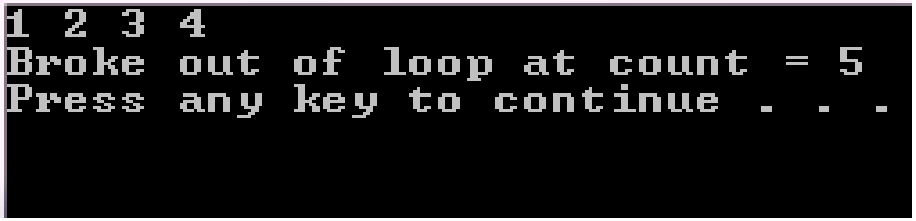
Output :

```
before the break
after the do loop
```

Break cont(example)

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  {
5.      int count;
6.      for (count = 1;count<=10;count ++)
7.      {
8.          if (count == 5)
9.          break;
10.         cout<< count<< " ";
11.     }
12.     cout<<"\nBroke out of loop at count = "<<count<<endl;
13.     return 0;
14. }
```

Output :



Continue

- The continue statement can be used in loops and has the opposite effect to break, that is, the next loop is begun immediately.
- In while, do…while:
  - The loop continuation condition evaluates immediately after the continue statement executes.
- In for statement:
  - The increment expression executes, then the loop-continuation test evaluates.

Continue cont(example)..

```
1.  #include <iostream>
2.  using namespace std;
3.  int main()
4.  { int i = 0;
5.      do
6.    {           i++;
7.                cout<<"before the continue\n";
8.                continue;
9.                cout<< "after the continue, should never print\n";
10.     } while (i < 3);
11.      cout<< "after the do loop\n";
12.      return 0;
13. }
```

Continue cont(Output)..

Output :

```
before the continue
before the continue
before the continue
after the do loop
Press any key to continue . . .
```

Continue cont(example)..

```
int main()
{
   for (int count = 1;count<=10;count++)
   {     if(count == 5)
                    continue;
         cout << count << " ";
   }
   cout<<"\nUsed continue to skip printing 5"<<endl;
   return 0;
}
```

Continue cont(output)..

Output :