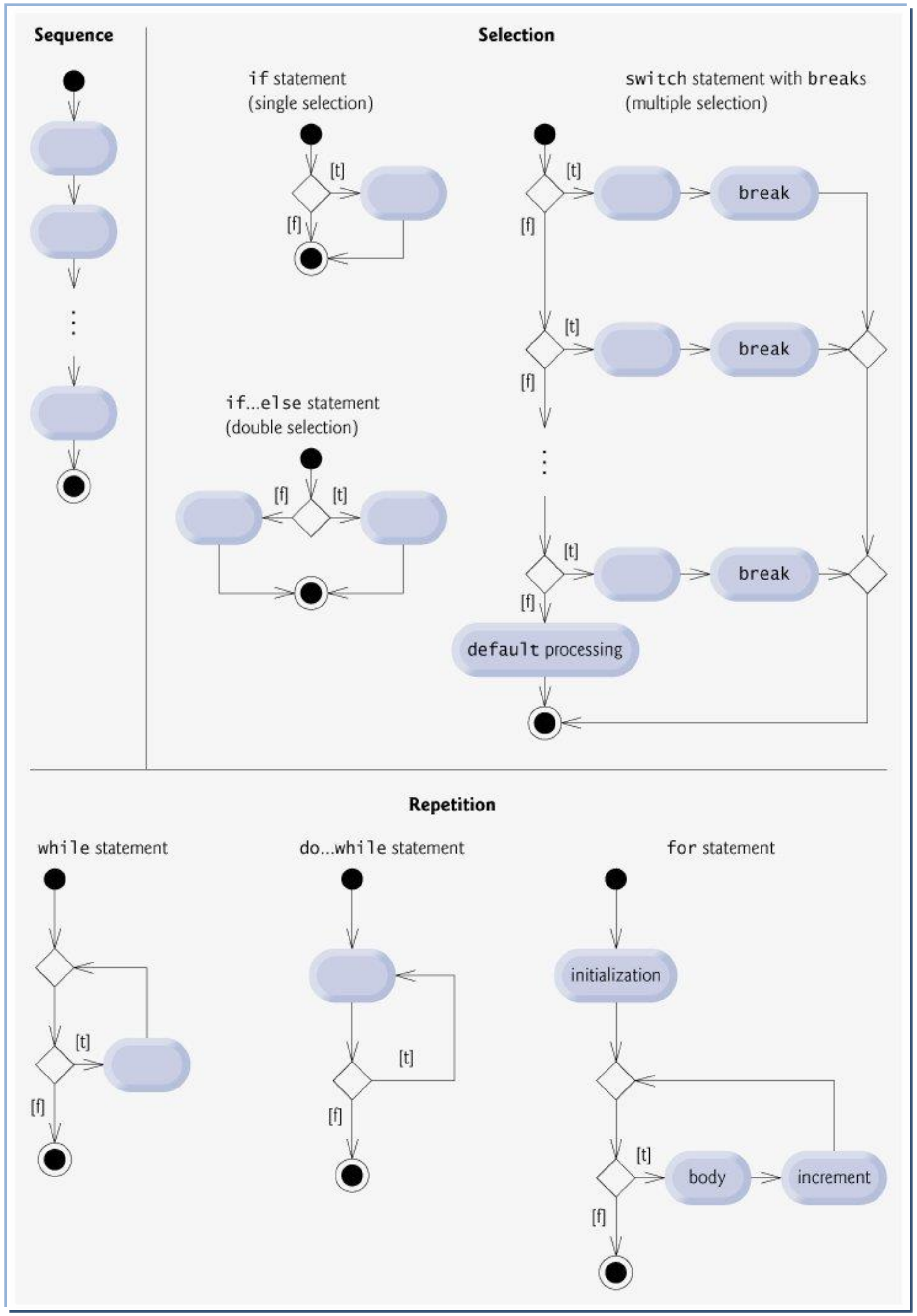
C++: Control Structures

Original author: Dr. Ha Viet Uyen Synh.

Agenda



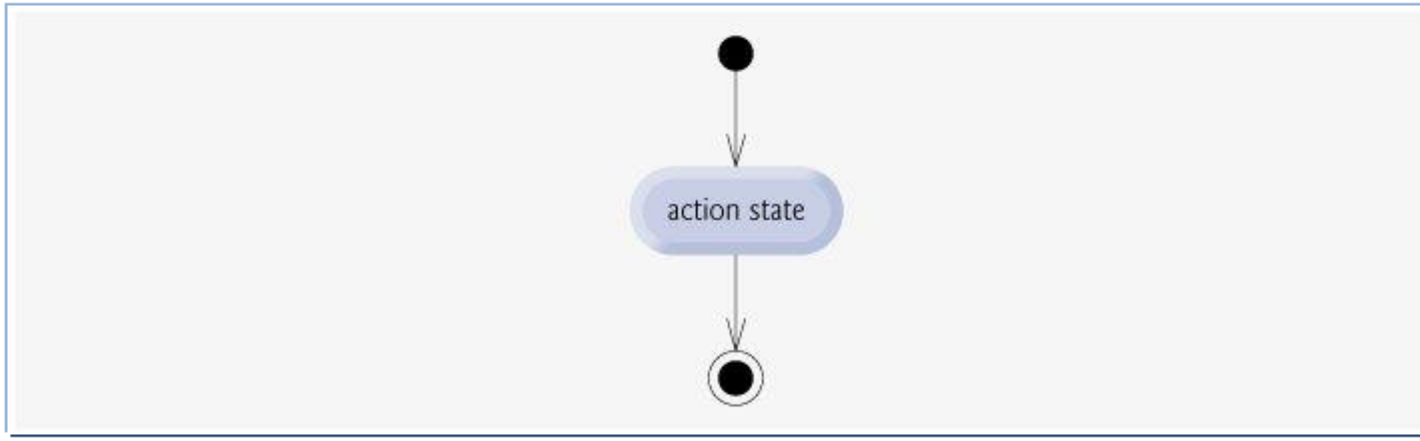
Sequence
Selection
if statement
(single selection)
switch statement with breaks
(multiple selection)
[t]
[f]
break
break
break
if...else statement
(double selection)
[f]
[t]
default processing
Repetition
while statement
do...while statement
for statement
initialization
body
increment

Rules for forming structured programs

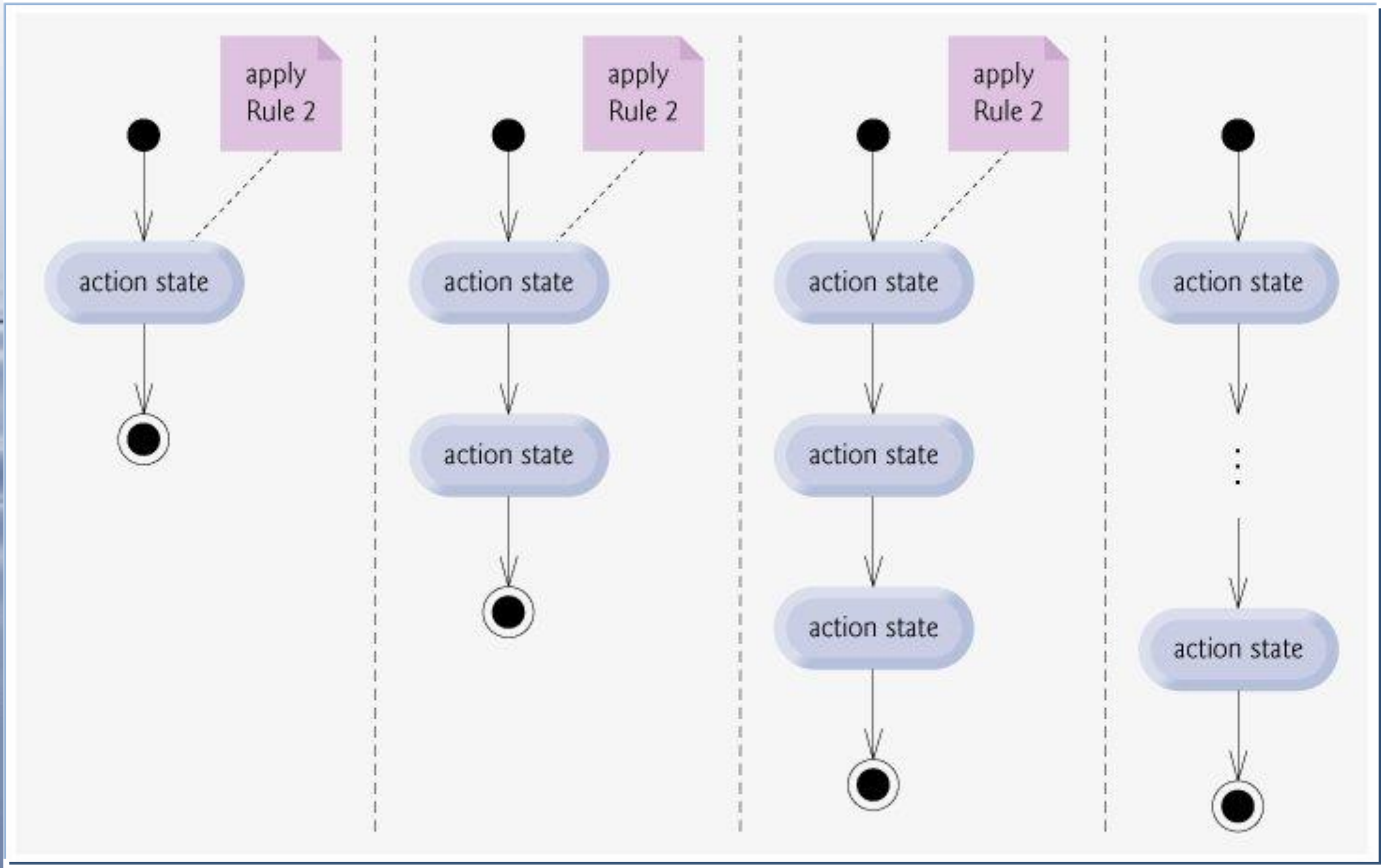
Rules for Forming Structured Programs

- Begin with the "simplest activity diagram“.
- Any action state can be replaced by two action states in sequence _ **Stacking rule .**
- Any action state can be replaced by any control statement (sequence, if, if…else, switch, while, do…while or for) _ **Nesting rule.**

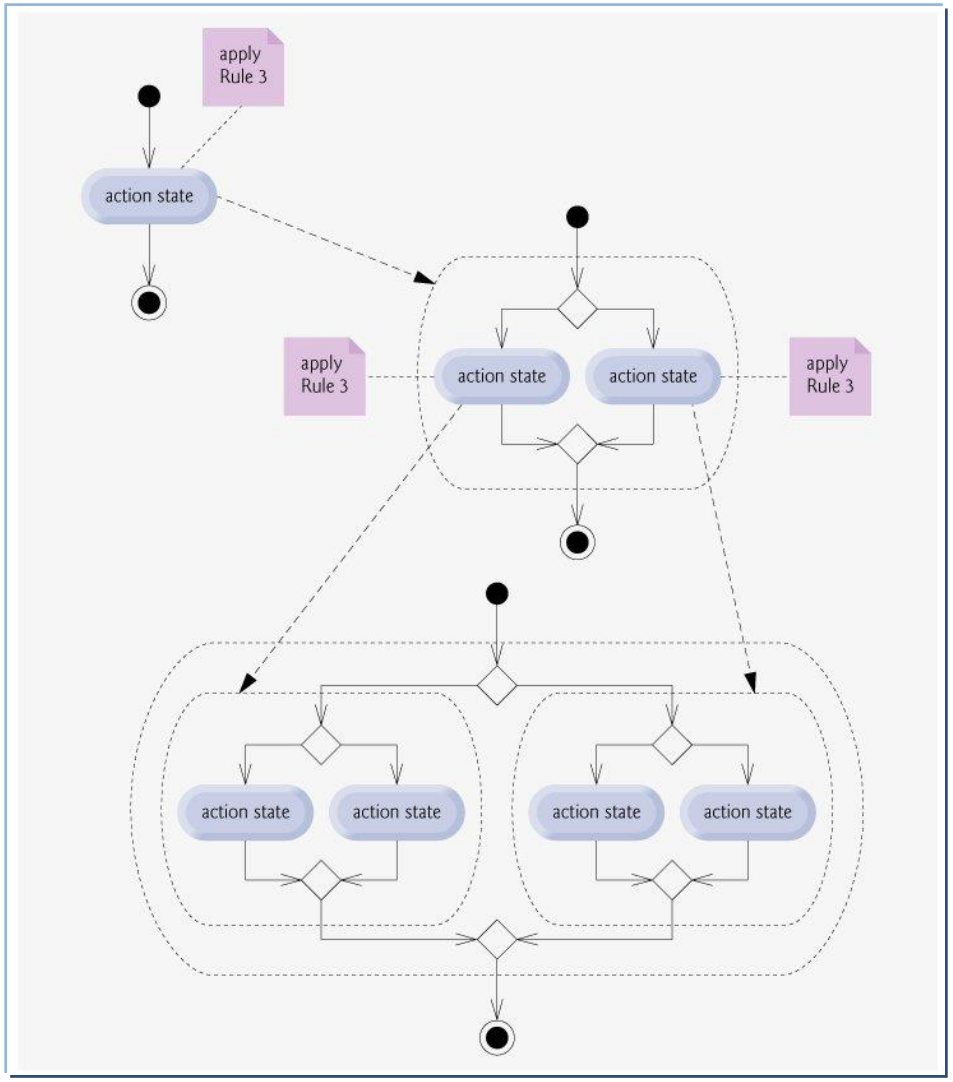
Rules 2 and 3 can be applied as often as you like and in any order.



Applying Stacking Rule



Applying Nesting Rule



if Selection Statement

The `if` selection statement allows us to state that an action (sequence of statements) should happen only when some condition is true:

```
if (condition)
    action;
```

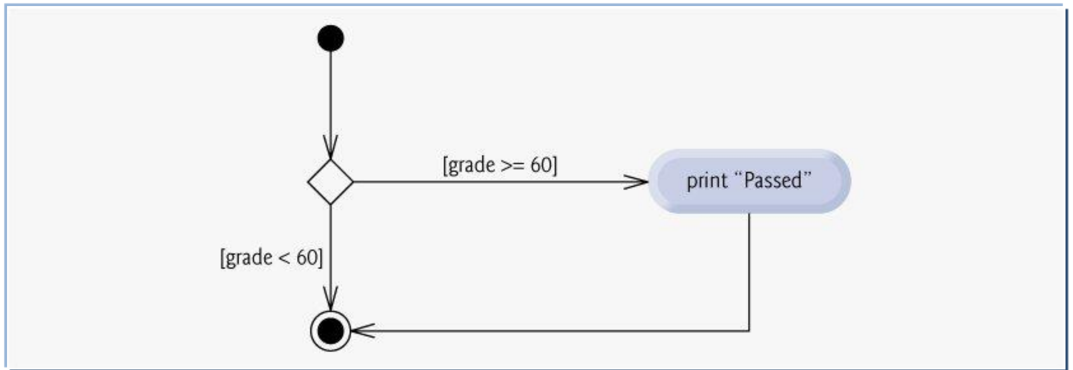
The *condition* used in an `if` (and other control structures) is a Boolean value - either `true` or `false`.

In C++:

the value 0 is `false`

anything else is `true`

```
if ( grade >= 60 )
        cout << "Passed";
```



if...else Double-Selection Statement

The if single-selection statement performs an indicated action only when the condition is TRUE; otherwise the action is skipped.

The if...else double-selection statement allows the programmer to specify an action to perform when the condition is true and a different action to perform when the condition is false.

```
if ( condition )
      action if true
else
            action if false
```

Nested if...else Statements

Nested **if...else** statements test for multiple cases by placing if...else selection statements inside other if...else selection statements.

```
if ( studentGrade >= 90 ) // 90 and above gets "A"
        cout << "A";
else if ( studentGrade >= 80 ) // 80-89 gets "B"
        cout << "B";
else if ( studentGrade >= 70 ) // 70-79 gets "C"
        cout << "C";
else if ( studentGrade >= 60 ) // 60-69 gets "D"
        cout << "D";
else // less than 60 gets "F"
        cout << "F";
```

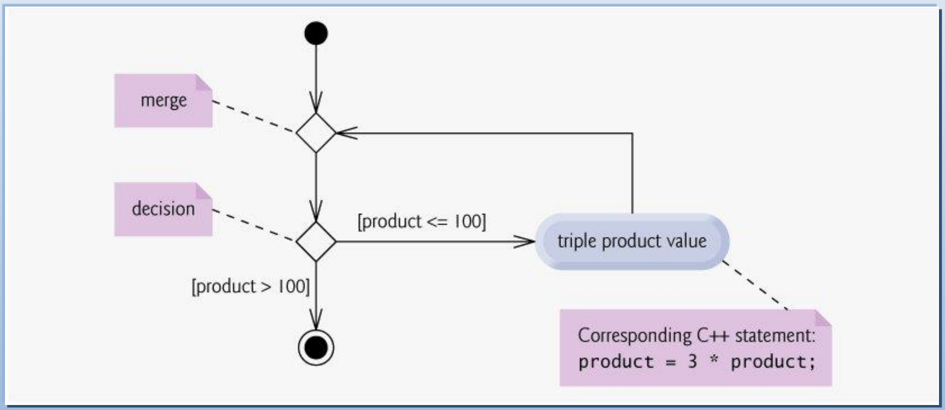
```
if ( studentGrade >= 90 ) // 90 and above gets "A"
        cout << "A";
else
        if ( studentGrade >= 80 ) // 80-89 gets "B"
                cout << "B";
        else
                if ( studentGrade >= 70 ) // 70-79 gets "C"
                        cout << "C";
                else
                        if ( studentGrade >= 60 ) // 60-69 gets "D"
                                cout << "D";
                        else // less than 60 gets "F"
                                cout << "F";
```

while Repetition Statement

The `while` repetition statement supports repetition - the same statement (or compound statement) is repeated until the condition is false.

```
while (condition)
do something;
```

```
// Calculate 3^n with 3^n<=100
int product = 3;
while ( product <= 100)
            product = 3 * product;
```



Counter-controlled repetition

```
// Fig. 5.1: fig05_01.cpp
// Counter-controlled repetition.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
int counter = 1; // declare and initialize control variable

while ( counter <= 10 ) // loop-continuation condition
{
        cout << counter << " ";
        counter++; // increment control variable by 1
} // end while

cout << endl; // output a newline
return 0; // successful termination
} // end main
```

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

for Repetition Statement

The `for` repetition statement is often used for loops that involve counting.

You can write any `for` loop as a `while` (and any `while` as a `for`).

```
for (initialization; condition; update)
        dosomething;
```

- initialization is a statement that is executed at the beginning of the loop (and never again).

- the body of the loop is executed as long as the condition is true.

- the update statement is executed each time the body of the loop has been executed (and before the condition is checked)

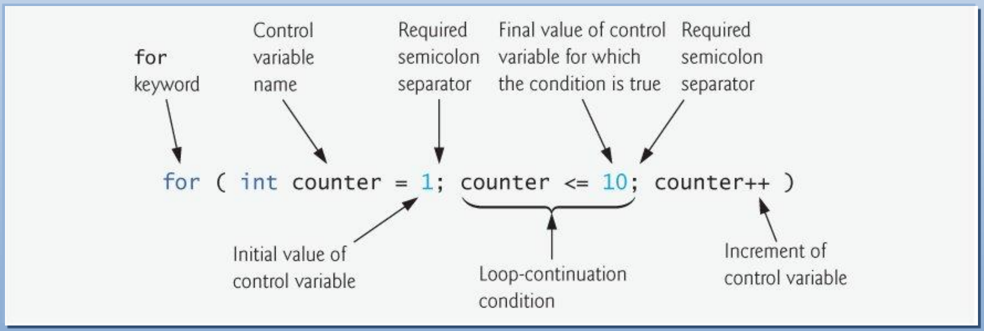
Counter-controlled repetition with the for statement.

```
// Fig. 5.2: fig05_02.cpp
// Counter-controlled repetition with the for statement.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
// for statement header includes initialization,
// loop-continuation condition and increment.
for ( int counter = 1; counter <= 10; counter++ )
cout << counter << " ";

cout << endl; // output a newline
return 0; // indicate successful termination
} // end main
```

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



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

for keyword
Control variable name
Required semicolon separator
Final value of control variable for which the condition is true
Required semicolon separator
Initial value of control variable
Loop-continuation condition
Increment of control variable

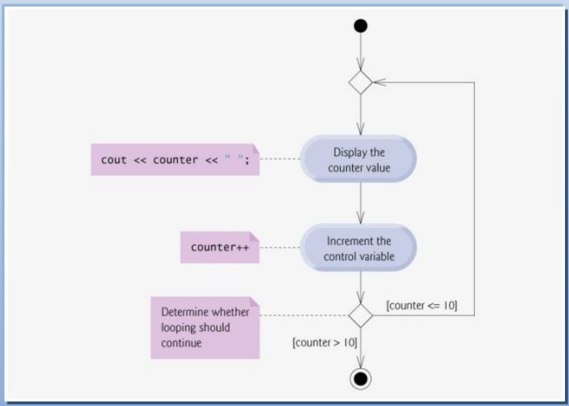
do...while Repetition Statement

The do...while repetition statement is similar to the while statement.

In the while statement, the loop-continuation condition test occurs at the beginning of the loop before the body of the loop executes. The do...while statement tests the loop-continuation condition after the loop body executes; therefore, the loop body always executes at least once.

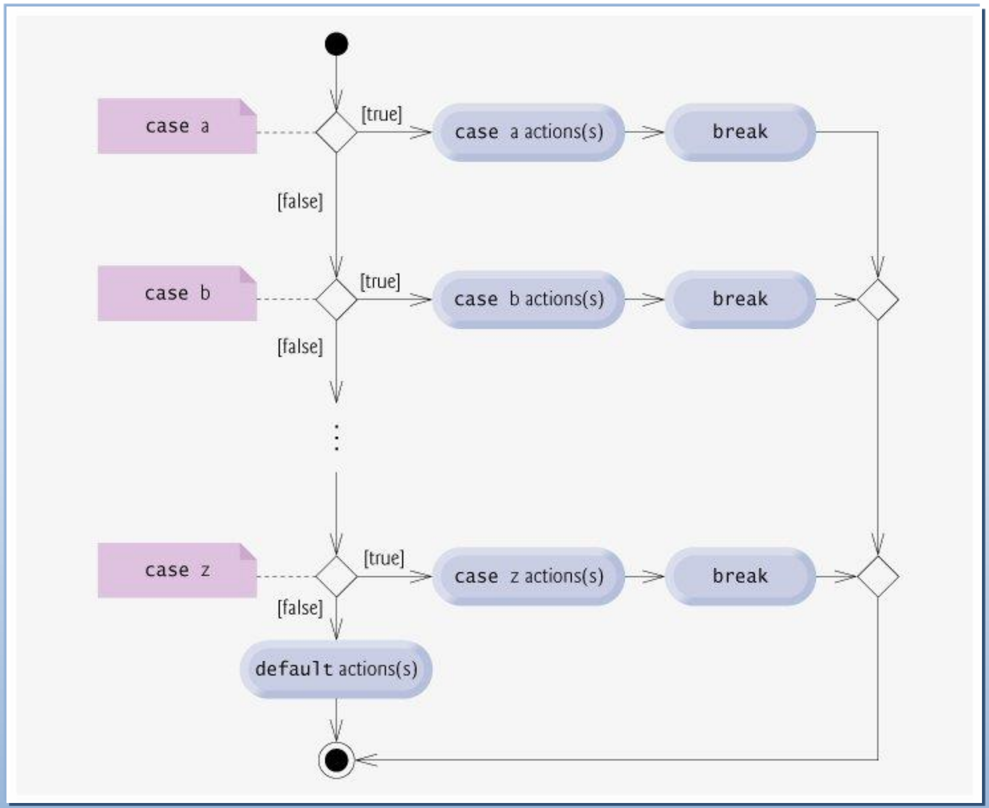
When a do...while terminates, execution continues with the statement after the while clause. Note that it is not necessary to use braces in the do...while statement if there is only one statement in the body; however, most programmers include the braces to avoid confusion between the while and do...while statements

```
do
        somestuff;
while ( condition );
```


cout << counter << " ";
Display the counter value
counter++
Increment the control variable
Determine whether looping should continue
[counter <= 10]
[counter > 10]

switch Multiple-Selection Statement

switch multiple-selection statement to perform many different actions based on the possible values of a variable or expression. Each action is associated with the value of a **constant integral expression**



break Statement

The break statement, when executed in a while, for, do...while or switch statement, causes immediate exit from that statement.

Program execution continues with the next statement.

Common uses of the break statement are to escape early from a loop or to skip the remainder of a switch statement

```
// Fig. 5.13: fig05_13.cpp
// break statement exiting a for statement.
#include <iostream>
using std::cout;
using std::endl;

int main()
{
int count; // control variable also used after loop terminates

for ( count = 1; count <= 10; count++ ) // loop 10 times
{
if ( count == 5 )
break; // break loop only if x is 5

cout << count << " ";
} // end for

cout << "\nBroke out of loop at count = " << count << endl;
return 0; // indicate successful termination
} // end main
```

continue Statement

The continue statement, when executed in a while, for or do...while statement, skips the remaining statements in the body of that statement and proceeds with the next iteration of the loop.

In while and do...while statements, the loop-continuation test evaluates immediately after the continue statement executes.

In the for statement, the increment expression executes, then the loop-continuation test evaluates.

```cpp
// Fig. 5.14: fig05_14.cpp
// continue statement terminating an iteration
#include <iostream>
using std::cout;
using std::endl;

int main()
{
   for ( int count = 1; count <= 10; count++ )
   {
      if ( count == 5 ) // if count is 5,
         continue;      // skip remaining code

      cout << count << " ";
   } // end for

 cout<<"\nUsed continue to skip printing 5"<<endl;
   return 0;
} // end main
```

Any Questions?