AAB University

Faculty of Computer Sciences

Object Oriented Programming

Week 5:

Control Statements

Asst. Prof. Dr. **Mentor Hamiti**
mentor.hamiti@universitetiaab.com

Last Time?!

- Introduction to Classes and Objects
- Defining a Class with a Member Function
- Defining a Member Function with a Parameter
- Data Members, set Functions and get Functions
- Initializing Objects with Constructors
- Placing a Class in a Separate File for Reusability

Today

- **Control Structures**
- **if** Selection Statement
- **if … else** Double-Selection Statement
- **while** Repetition Statement
- **for** Repetition Statement
- **do … while** Repetition Statement
- **switch** Multiple-Selection Statement
- **break** and **continue** Statements

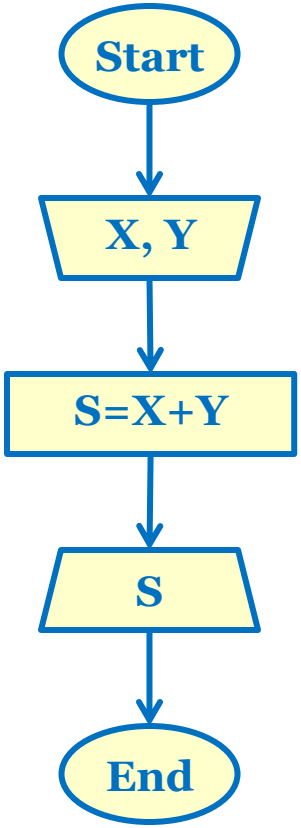
Introduction

- Before writing a program to solve a problem, we must have a thorough understanding of the problem and a carefully planned approach to solving it

- When writing a program, we must also understand the types of building blocks that are available and employ proven program construction techniques

Algorithms

- Any solvable computing problem can be solved by the execution a series of actions in a specific order
- An **algorithm** is **procedure** for solving a problem in terms of
  - the actions to execute and
  - the order in which the actions execute
- Specifying the or-der in which statements (*actions*) execute in a computer program is called **program control**
- Program control can be realised using C++'s control statements



Pseudocode

- Pseudocode is an artificial and informal language that helps us develop algorithms
- Carefully prepared pseudocode can easily be converted to a corresponding C++ program
- Example:

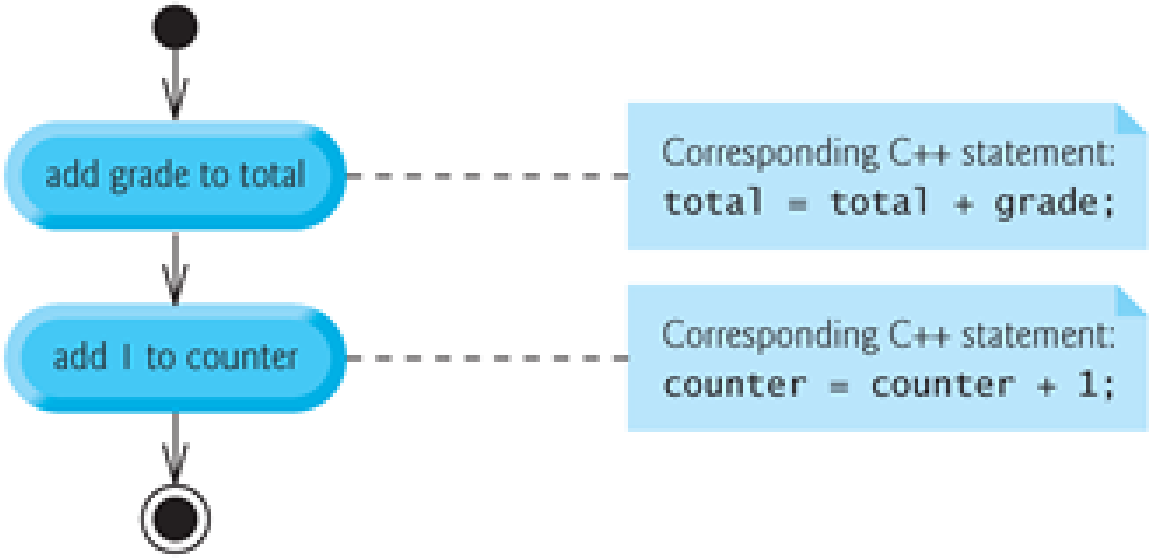
```
1   Prompt the user to enter the first integer
2   Input the first integer
3
4   Prompt the user to enter the second integer
5   Input the second integer
6
7   Add first integer and second integer, store result
8   Display result
```

- Control Structures

Control Structures

- Normally, statements in a program execute one after the other in the order in which they're written - **Sequential execution**
- Various C++ statements enable you to specify that the next statement to execute may be other than the next one in sequence - **Transfer of control**
- All programs could be written in terms of only three control structures:
  - The sequence structure
  - The selection structure and
  - The repetition structure

- The **U**nified **M**odeling **L**anguage (**UML**) **activity diagram**:



- *Activity diagrams are composed of special-purpose symbols*
- *A-diagrams clearly show how control structures operate*
- *Activity diagrams help us develop and represent* ***algorithms****, but many programmers prefer* ***pseudocode***

Control Structures - Selection

- C++ provides three types of selection statements :
  - The if selection statement either performs (selects) an action if a condition (predicate) is true or skips the action if the condition is false
  - The **if...else** selection statement performs an action if a condition is true or performs a different action if the condition is false
  - The **switch** selection statement performs one of many different actions, depending on the value of an integer expression

Control Structures - Selection

- C++ provides three types of selection statements :
  - The **if** selection statement is **a single-selection statement** because it selects or ignores a single action
  - The **if...else** statement is called **a double-selection statement** because it selects between two different actions (or groups of actions).
  - The **switch** selection statement is called **a multiple-selection statement** because it selects among many different actions (or groups of actions).

Control Structures - Repetition

- C++ provides three types of repetition statements (also called looping statements or **loops**) for performing statements repeatedly while a condition (called the loop-continuation **condition**) remains true

- These are the **while**, **do…while** and **for** statements

  - The **while** and **for** statements perform the action (or group of actions) in their bodies zero or more times

  - The **do…while** statement performs the action (or group of actions) in its body at least once

Control Structures - Repetition

- Any C++ programm can be constructed from only seven different types of Control Statements:
  - Sequence
  - if
  - if ... else
  - switch
  - while
  - do ... while
  - for

- Control Structures
- **if** Selection Statement

if Selection Statement

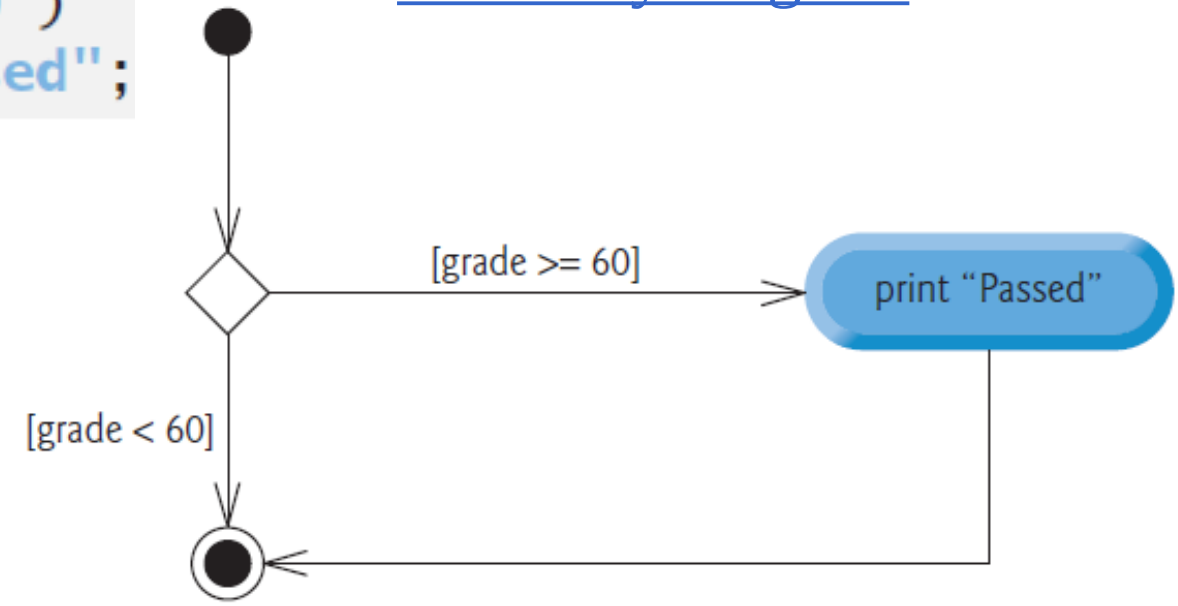
- The pseudocode for **if** selection statement:

If student's grade is greater than or equal to 60
Print "Passed"

- **if** statement C++ code:

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

if activity diagram:



if Selection Statement

- C++ provides the data type **bool** for variables that can hold only the values **true** and **false -** each of these is a C++ keyword

- A decision can be based on any expression - **if** the expression evaluates to **zero**, it's treated as **false**; if the expression evaluates to **nonzero**, it's treated as **true**.

- Control Structures
- **if** Selection Statement
- **if ... else** Double-Selection Statement

if ... else Double-Selection Statement

- **if...else** statement specifies an action to perform when the condition is true and a different action to perform when the condition is false

- Pseudocode:

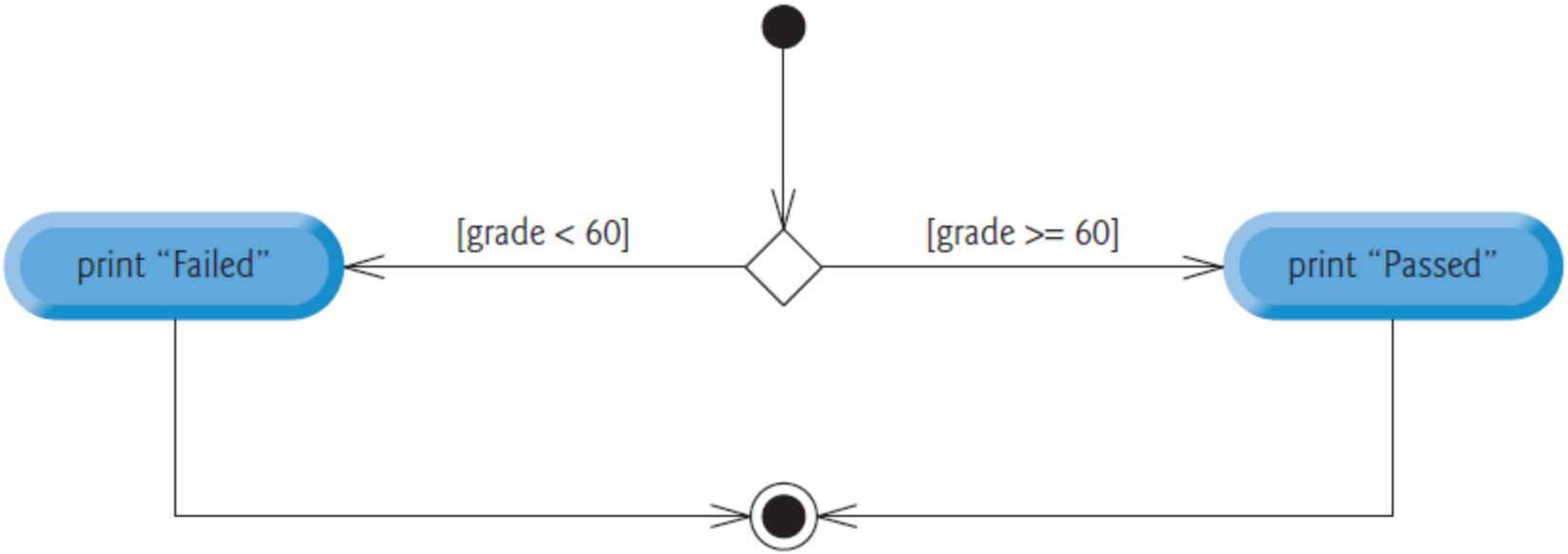
```
If student's grade is greater than or equal to 60
    Print "Passed"
Else
    Print "Failed"
```

- C++ code:

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

if ... else Double-Selection Statement

- **if...else** double - selection statement activity diagram:



*Nested **if ... else** Statement*

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

```
If student's grade is greater than or equal to 90
    Print "A"
Else
    If student's grade is greater than or equal to 80
        Print "B"
    Else
        If student's grade is greater than or equal to 70
            Print "C"
        Else
            If student's grade is greater than or equal to 60
                Print "D"
            Else
                Print "F"
```

*Nested **if ... else** Statement*

- This pseudocode can be written in C++ as:

```
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";
```

*Nested **if ... else** Statement*

- Most programmers write the statement as:

```
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";
```

  - *The two forms are identical except for the spacing and indentation, which the compiler ignores*

*Dangling - **else** problem*

- The C++ compiler always associates an else with the immediately preceding if unless told to do otherwise by the placement of braces { and }

```
if ( x > 5 )
   if ( y > 5 )
      cout << "x and y are > 5";
else
   cout << "x is <= 5";
```

```
if ( x > 5 )
{
   if ( y > 5 )
      cout << "x and y are > 5";
}
else
   cout << "x is <= 5";
```

- Control Structures
- **if** Selection Statement
- **if ... else** Double-Selection Statement
- **while** Repetition Statement

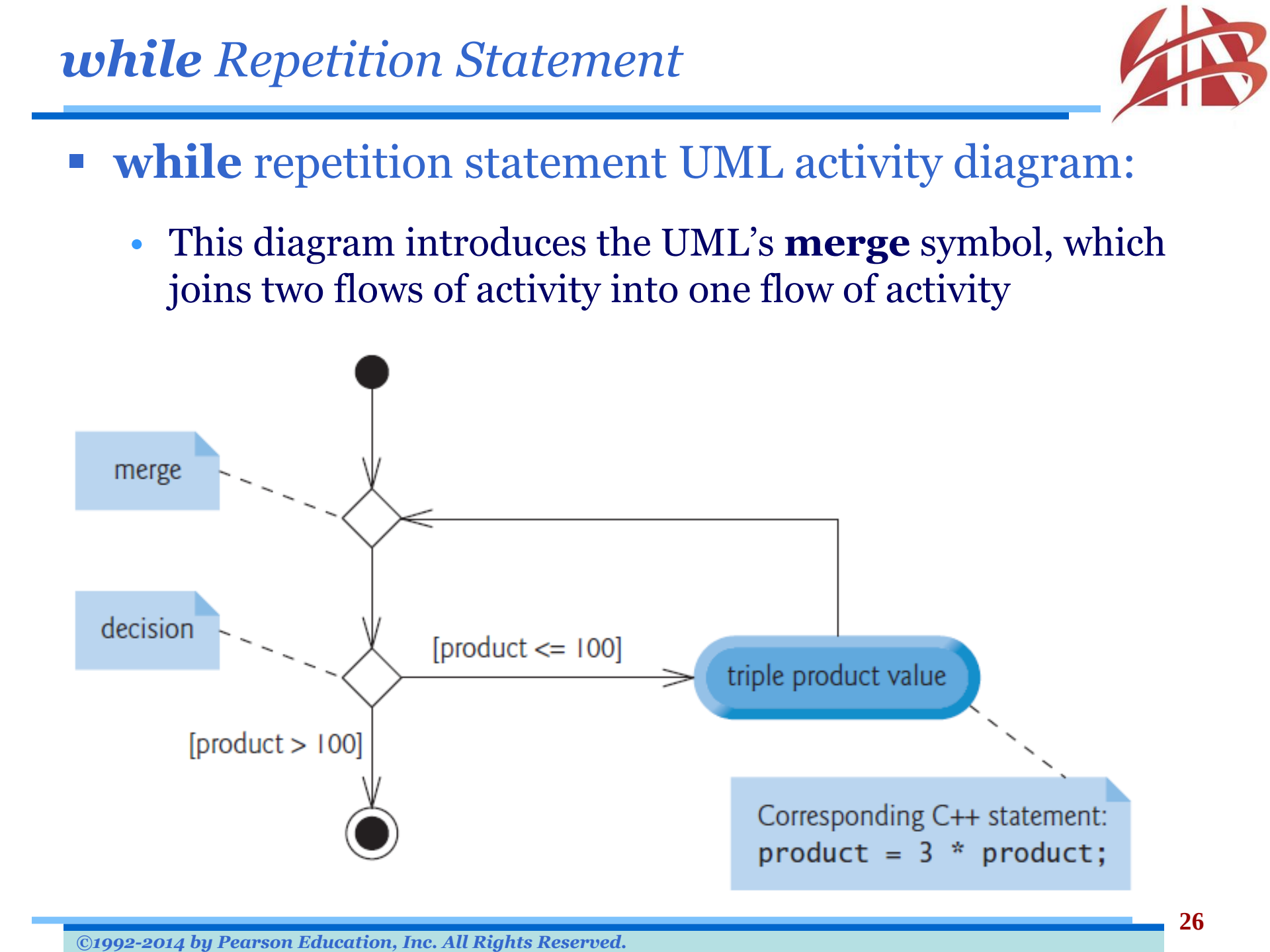
while Repetition Statement

- A repetition statement (also called a looping statement or a **loop**) allows you to specify that a program should **repeat** an action while some condition remains true
- Example:
  - Consider a program segment designed to find the first power of 3 larger than 100. Suppose the integer variable product has been initialized to 3

```
int product = 3;

while ( product <= 100 )
   product = 3 * product;
```

  - *When the following while repetition statement finishes executing, product contains the result*

while Repetition Statement



- **while** repetition statement UML activity diagram:
  - This diagram introduces the UML's **merge** symbol, which joins two flows of activity into one flow of activity

while Repetition Statement

- Example: Counter-controlled repetition

```
// Counter-controlled repetition.
#include <iostream>
using namespace std;

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
} // end main
```

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

- Control Structures
- **if** Selection Statement
- **if … else** Double-Selection Statement
- **while** Repetition Statement
- **for** Repetition Statement

for Repetition Statement

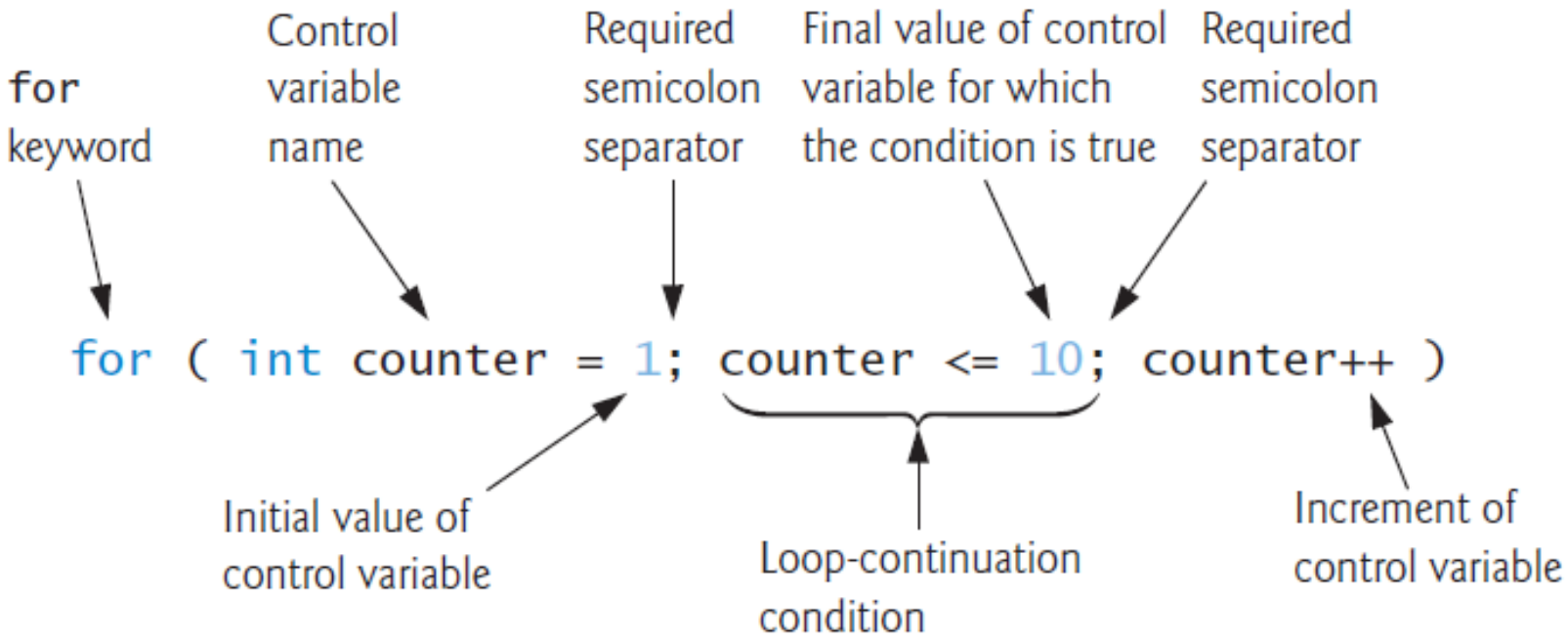
- The **for** repetition statement specifies the counter-controlled repetition details in a single line of code!

```
// Counter-controlled repetition with the for statement
#include <iostream>
using namespace std;

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
} // end main
```

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

for Repetition Statement

- **for** statement components:



for Repetition Statement

- **for vs while**

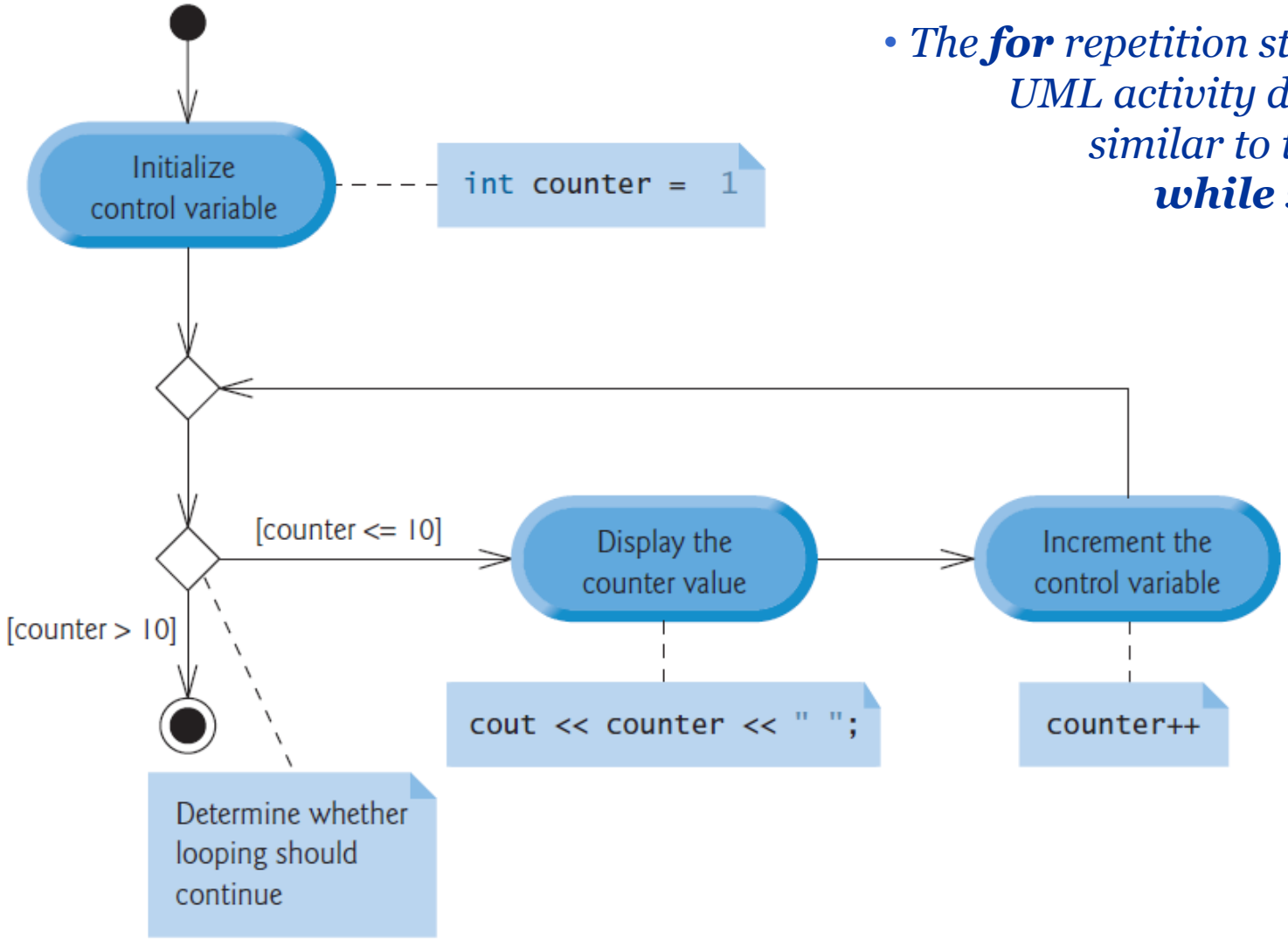
```
for ( initialization; loopContinuationCondition; increment )
    statement
```

or

```
initialization;

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

for Repetition Statement

- **for** Statement UML Activity Diagram



- *The **for** repetition statement's UML activity diagram is similar to that of the **while** statement*

for Repetition Statement

- Examples:

```
for ( int i = 1; i <= 100; ++i )
```

```
for ( int i = 100; i >= 1; --i )
```

```
for ( int i = 7; i <= 77; i += 7 )
```

```
for ( int i = 20; i >= 2; i -= 2 )
```

for Repetition Statement

- Example: Summing the Even Integers from 2 to 20

```
#include <iostream>
using namespace std;

int main()
{
   int total = 0; // initialize total

   // total even integers from 2 through 20
   for ( int number = 2; number <= 20; number += 2 )
      total += number;

   cout << "Sum is " << total << endl; // display results
} // end main
```

```
Sum is 110
```

- Control Structures
- **if** Selection Statement
- **if ... else** Double-Selection Statement
- **while** Repetition Statement
- **for** Repetition Statement
- **do ... while** Repetition Statement

do ... while Repetition Statement

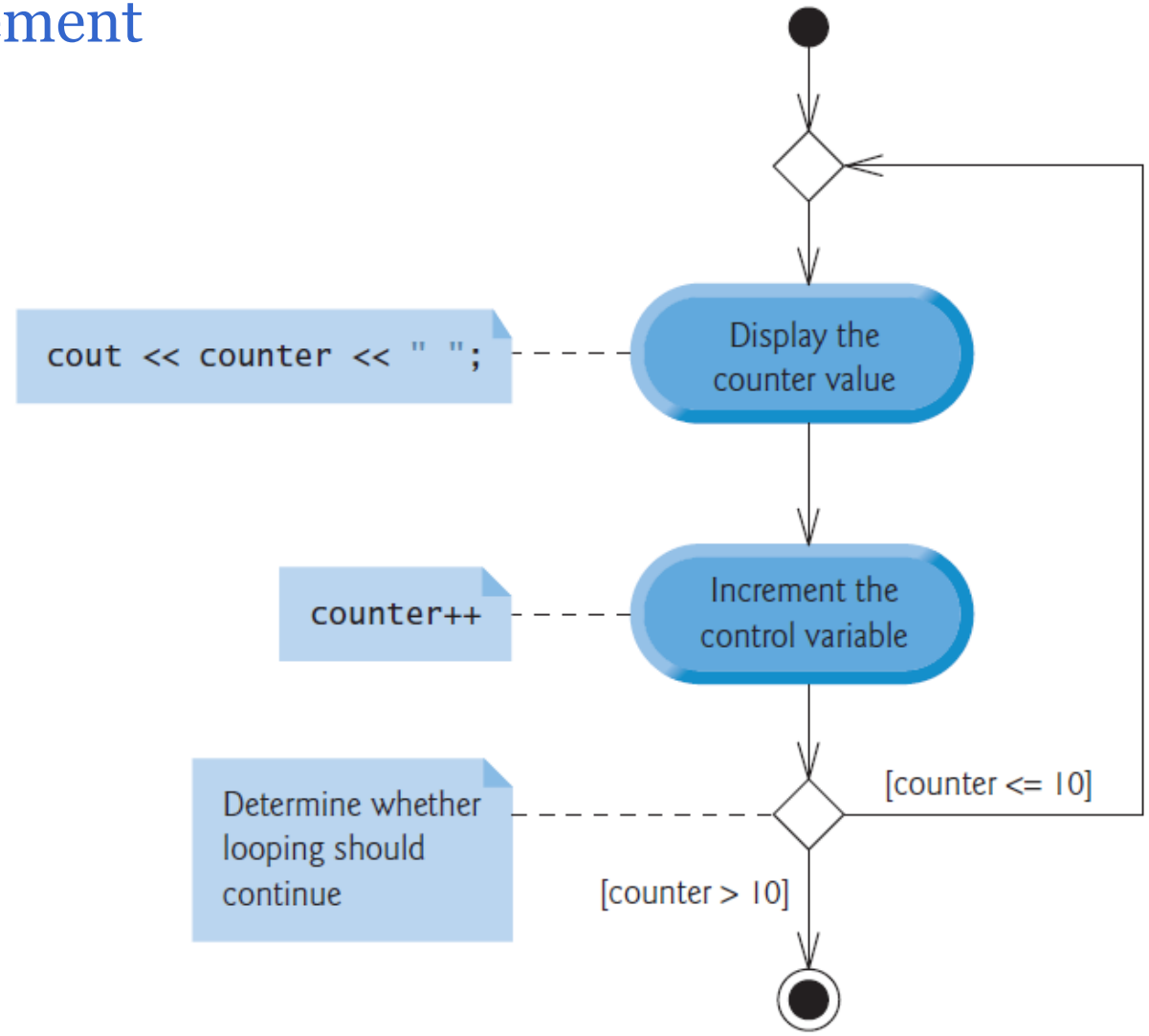
- **do...while** with one statement often is written as follows:

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

  - *The loop-continuation condition is not evaluated until after the loop performs its body at least once*

do ... while Repetition Statement

- UML activity diagram for the **do...while** repetition statement



do ... while Repetition Statement

- Example:

```
// do...while repetition statement.
#include <iostream>
using namespace std;

int main()
{
   int counter = 1; // initialize counter

   do
   {
      cout << counter << " "; // display counter
      ++counter; // increment counter
   } while ( counter <= 10 ); // end do...while

   cout << endl; // output a newline
} // end main
```

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

- Control Structures
- **if** Selection Statement
- **if ... else** Double-Selection Statement
- **while** Repetition Statement
- **for** Repetition Statement
- **do ... while** Repetition Statement
- **switch** Multiple-Selection Statement

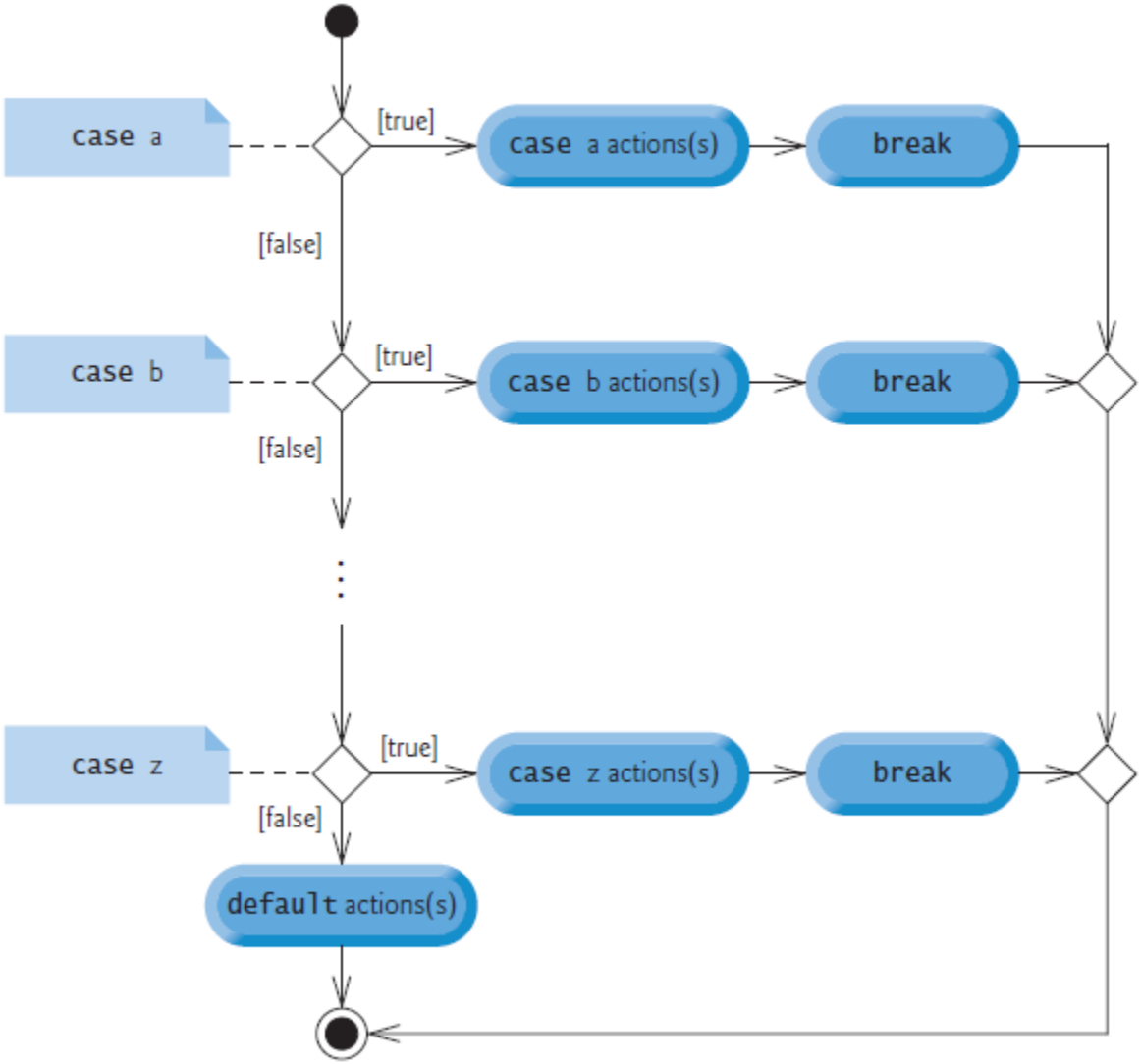
switch Multiple-Selection Statement

- The **switch** multiple-selection statement performs many different actions based on the possible values of a variable or expression
- The **switch** statement consists of a series of **case labels** and an optional **default case**

- Case label
  - The switch statement compares the value of the controlling expression with each case label. If a match occurs, the program executes the statements for that case
- Default Case
  - *If no match occurs between the controlling expression's value and a case label, the default case executes*
  - *If no match occurs in a switch statement that does not contain a default case, program control continues with the first statement after the switch*

switch Multiple-Selection Statement

- **switch** Statement UML Activity Diagram



- Control Structures
- **if** Selection Statement
- **if ... else** Double-Selection Statement
- **while** Repetition Statement
- **for** Repetition Statement
- **do ... while** Repetition Statement
- **switch** Multiple-Selection Statement
- **break** and **continue** Statements

break and continue Statements

- **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

break and continue Statements

- Example:

```
// break statement exiting a for statement.
#include <iostream>
using namespace std;

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 count is 5

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

   cout << "\nBroke out of loop at count = " << count << endl;
} // end main
```

```
1 2 3 4
Broke out of loop at count = 5
```

break and continue Statements

- **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

break and continue Statements

- Example:

```
// continue statement terminating an iteration of a for statement
#include <iostream>
using namespace std;

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

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

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

```
1 2 3 4 6 7 8 9 10
Used continue to skip printing 5
```

- Structured Programming Summary

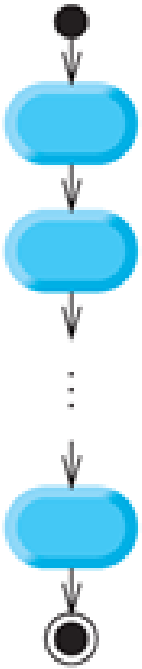
Structured Programming Summary

- Structured programming produces programs that are easier than unstructured programs to **understand**, **test**, **debug**, **modify**, and even prove **correct** in a mathematical sense

- Each control statement (*sequence*, *selection and repetition*) has the single entry point and the single exit point!

Structured Programming Summary

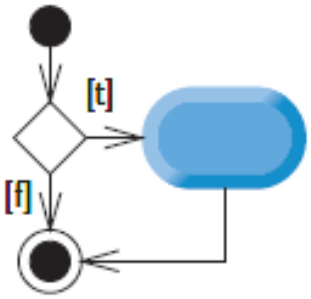
- Sequence:



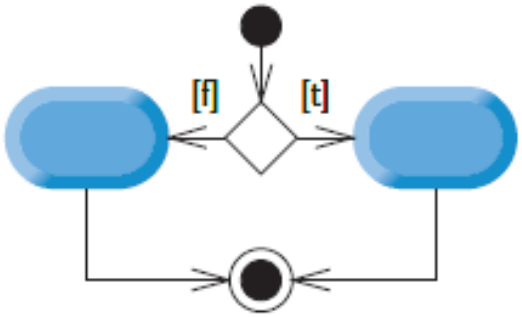
Structured Programming Summary

- Selection:

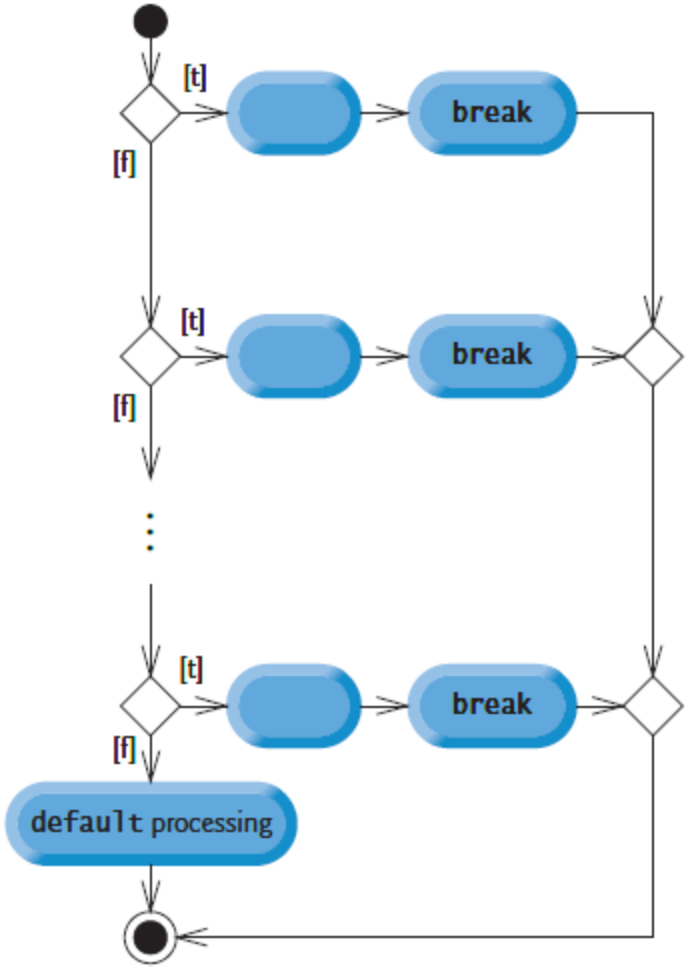
`if` statement
(single selection)



`if...else` statement
(double selection)

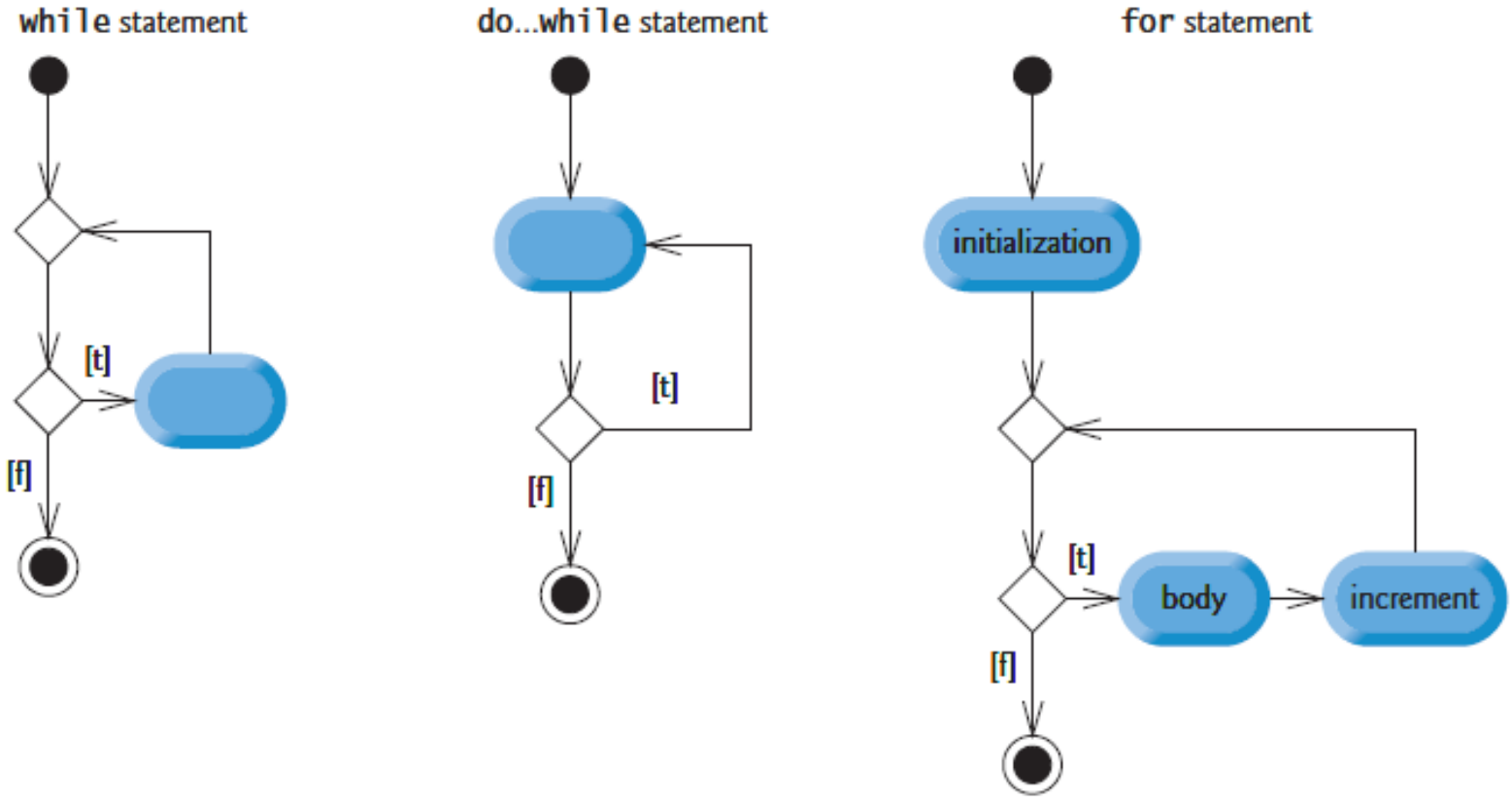


`switch` statement with `breaks`
(multiple selection)



Structured Programming Summary

- Repetition:



- Questions?!

mentor.hamiti@universitetiaab.com