Computer Programming
Lecture 5

Hung-Yu Wei
Department of Electrical Engineering
National Taiwan University

Chapter 4

- More on control statements
  - repetition
    - for
    - do…while
  - selection
    - switch (multiple selection)

Counter-controlled repetition

- Counter = control variable
- Essential elements
  - Name of a control variable (loop counter)
  - Initial value of the control variable
  - Continuation condition for the loop
  - The increment (or decrement) of the control variable

Example: Essential elements

```
int counter;  //name of the control variable
counter = 1;       //initial value
while ( counter <= 10 ) // continuation condition
{
     cout << counter << " ";
     counter++; // increment control variable
} // end while
```

```
1  // Fig. 4.1: fig04_01.cpp
2  // Counter-controlled repetition.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int counter = 1; // declare and initialize control variable
9
10    while ( counter <= 10 ) // loop-continuation condition
11    {
12       cout << counter << " ";
13       counter++; // increment control variable by 1
14    } // end while
15
16    cout << endl; // output a newline
17 } // end main
```

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

Fig. 4.1 | Counter-controlled repetition.

for

- Syntax

  for (initialization; condition; increment)
  - increment or decrement
- Example 1

  int n;

  for(n=1;n<=10;n++)
- Example 2

  for(int n=1;n<=10;n++)

Compare *for* with *while*

- For

```
for (initialization; loopCondition;increment)
 {
        statement;
}
```

- While

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

Same example: for v.s. while

```
while ( counter <= 10 )
 {
    cout << counter << " ";
    counter++;
 }
```

```
for ( int counter = 1; counter <= 10; counter++ )
{
    cout << counter << " ";
}
```

1 2 3 4 5 6 7 8 9 10 Output Results

Using { } with for

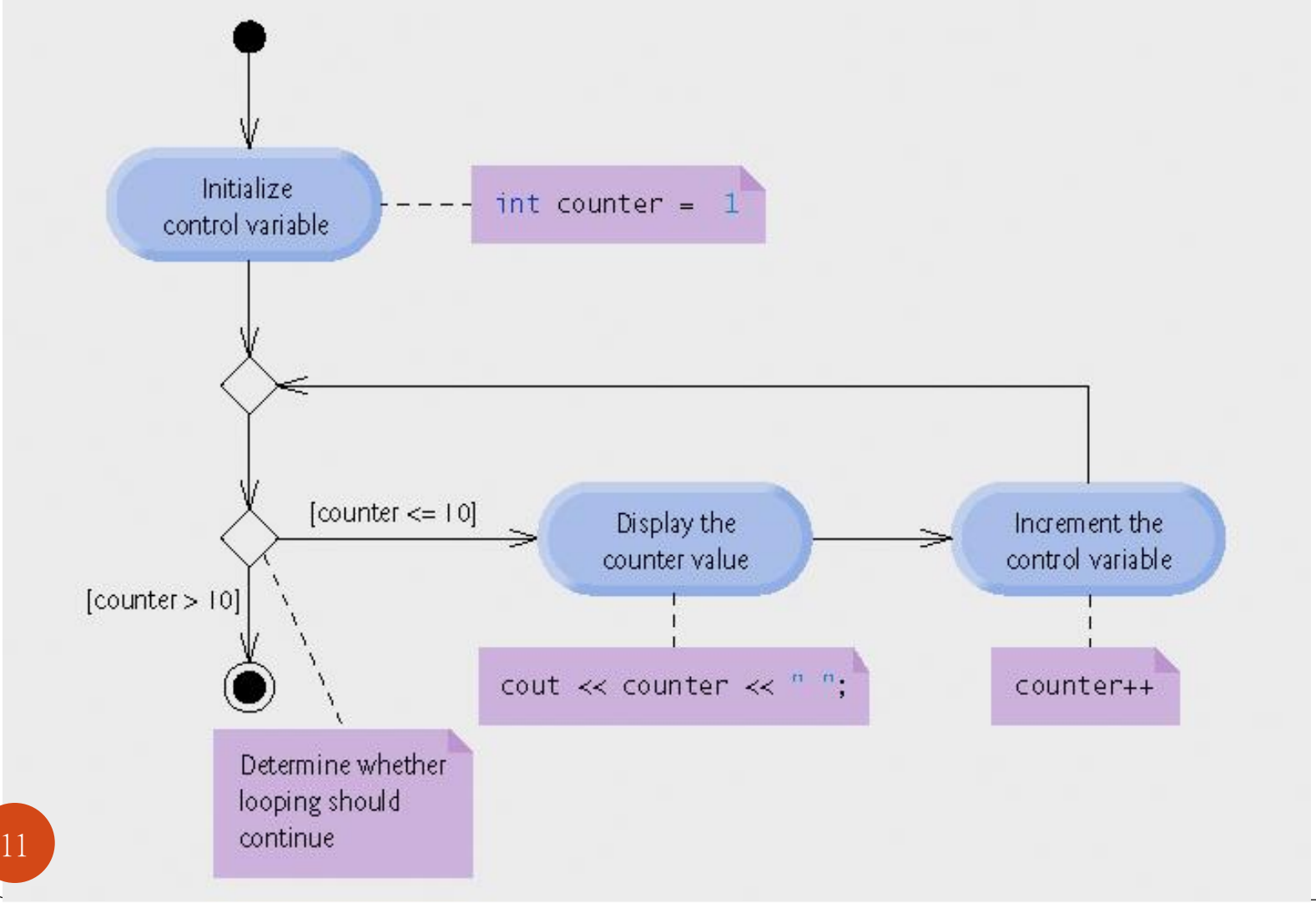
- Similar to the *while* and *if* case
- For multiple statements
- Example

```
for ( ;   ; )
{
        statement1;
        statement2;
        ……
 }
```

increment

- They are the same
  - Counter=Counter+1;
  - Counter += 1;
  - ++Counter;
  - Counter++;

UML: *for* loop


Initialize control variable
int counter = 1
[counter <= 10]
Display the counter value
Increment the control variable
[counter > 10]
cout << counter << " ";
counter++
Determine whether looping should continue

Example

- Sum even integers from 2 through 20

```
int main()
{
   int total = 0;
   for ( int number = 2; number <= 20; number += 2 )
   {
            total += number;
    }
         cout << "Sum is " << total << endl;
         return 0;
}
```

- Results
  - Sum is 110

do …while

- Syntax

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

- Similar to *while*
- Difference
  - while: check the condition first, and then execute the statements
  - do..while: execute the statements first, and then check the condition

Print 1,2,…,10

```
1  // Fig. 4.7: fig04_07.cpp
2  // do...while repetition statement.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int counter = 1; // initialize counter
9
10    do
11    {
12       cout << counter << " "; // display counter
13       counter++; // increment counter
14    } while ( counter <= 10 ); // end do...while
15
16    cout << endl; // output a newline
17 } // end main
```

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

Fig. 4.7 | do...while repetition statement.

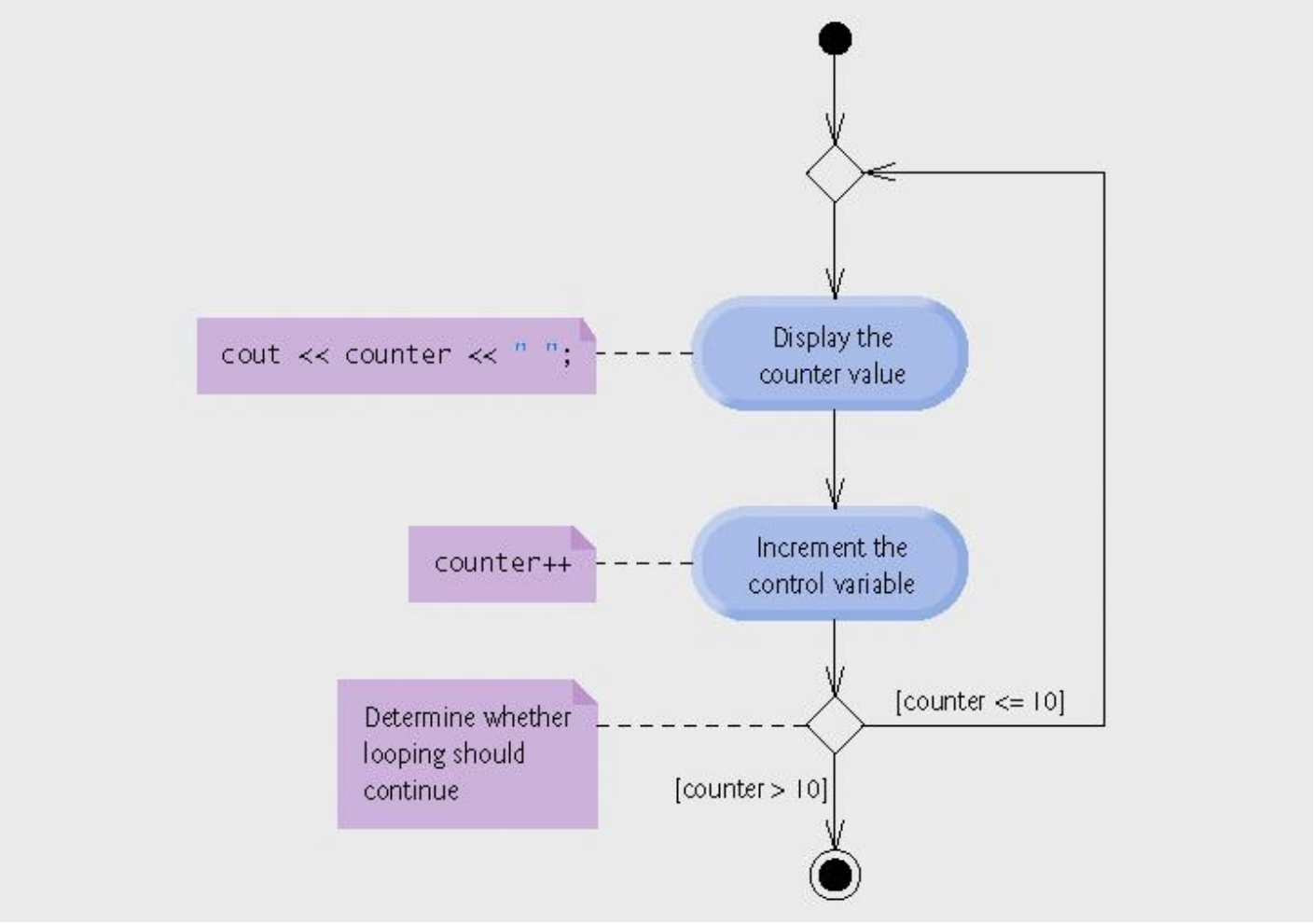
```
int counter = 1;
while ( counter <= 10 )
 {
    cout << counter << " ";
    counter++;
 }
```

```
for ( int counter = 1; counter <= 10; counter++ )
{
    cout << counter << " ";
}
```

```
 int counter = 1;
 do
  { cout << counter << " ";
    counter++;
  } while ( counter <= 10 );
```

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

UML: do…while


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

switch

- Multiple selection
  - In contrast to single selection (*if*)
- Syntax

```
switch (controlling expression)
{
        case 1:
                statements;
                break;
        case 2:
                statements;
                break;
        default:
                statements;
                break;
}
```

Switch and IF

```
switch (x) {
 case 1:
   cout << "x is 1";
   break;
 case 2:
   cout << "x is 2";
   break;
 default:
   cout << “x is not 1 nor 2";
}
```

```
if (x == 1) {
  cout << "x is 1";
}
else if (x == 2) {
  cout << "x is 2";
}
else {
  cout << " x is not 1 nor 2 ";
}
```

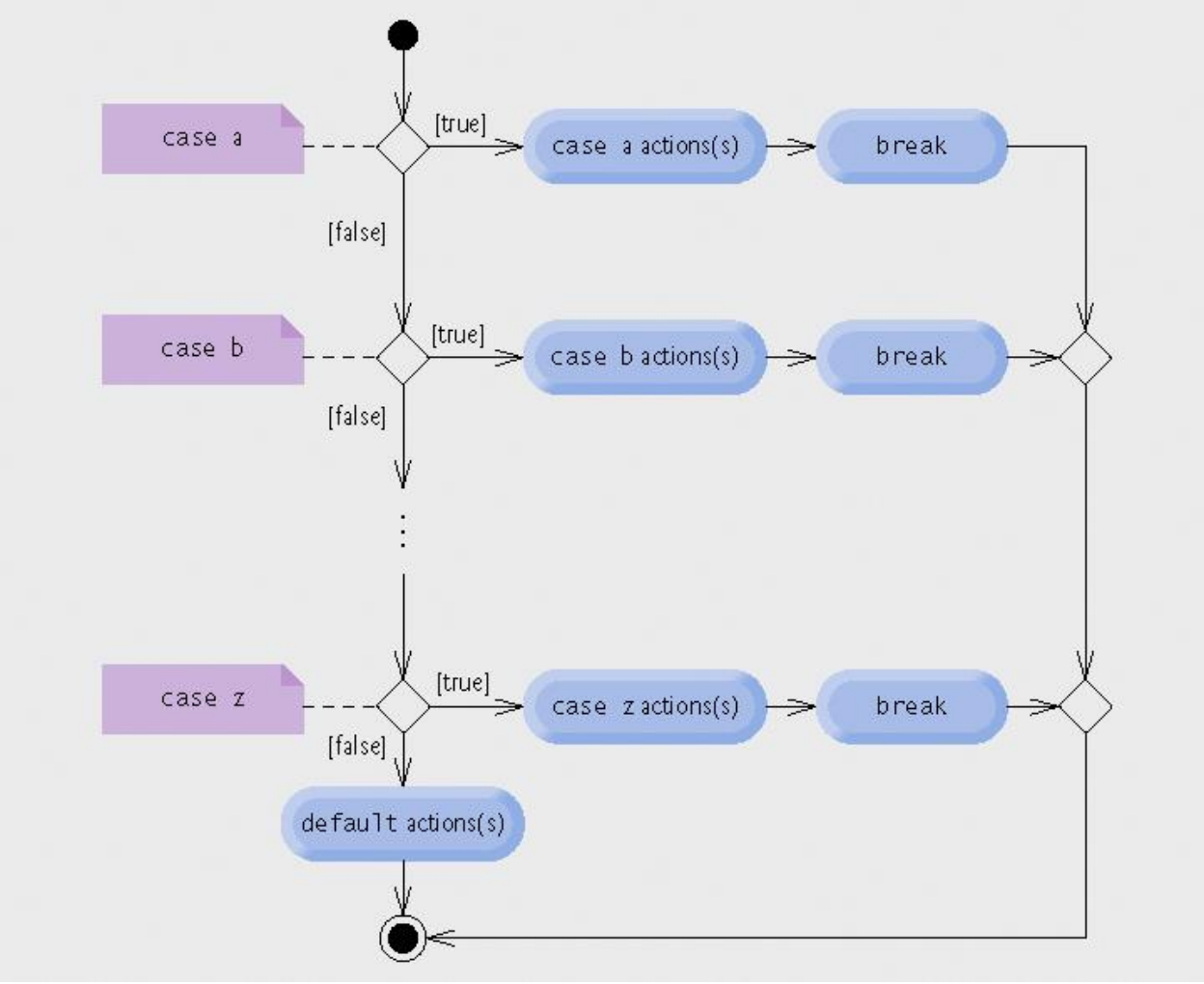
Switch and break(w/o break)

```
switch (x) {
  case 1:
  case 2:
  case 3:
    cout << "x is 1, 2 or 3";
    break;
  default:
    cout << "x is not 1, 2 nor 3";
  }
```

default in *switch*

- Usually, *default case* is put at the end of the switch statement
- Useful to handle exceptions
- Help you avoid bugs

UML: switch


case a
[true]
case a actions(s)
break
[false]
case b
[true]
case b actions(s)
break
[false]
case z
[true]
case z actions(s)
break
[false]
default actions(s)

break and *continue*

- Use within *while*, *for*, *do while*, *switch* statements
- break
  - Exit from the whole statement immediately
- continue
  - Skip the remaining statement body

Two example programs

- Simple *for* loop
  - Display 1,2,…,10
  - (1) Use "*break*" to escape from the whole loop
  - (2) Use "*continue*" to skip 1 iteration

```
1  // Fig. 4.11: fig04_11.cpp
2  // break statement exiting a for statement.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     int count; // control variable also used after loop terminates
9
10    for ( count = 1; count <= 10; count++ ) // loop 10 times
11    {
12       if ( count == 5 )
13          break; // break loop only if x is 5
14
15       cout << count << " ";
16    } // end for
17
18    cout << "\nBroke out of loop at count = " << count << endl;
19 } // end main
```

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

Fig. 4.11 | `break` statement exiting a `for` statement.

```
1  // Fig. 4.12: fig04_12.cpp
2  // continue statement terminating an iteration of a for statement.
3  #include <iostream>
4  using namespace std;
5
6  int main()
7  {
8     for ( int count = 1; count <= 10; count++ ) // loop 10 times
9     {
10       if ( count == 5 ) // if count is 5,
11          continue;      // skip remaining code in loop
12
13       cout << count << " ";
14    } // end for
15
16    cout << "\nUsed continue to skip printing 5" << endl;
17 } // end main
```

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

Fig. 4.12 | `continue` statement terminating a single iteration of a `for` statement.

Logical Operators

- Handle more complex conditions
- How do we use if with 2 conditions
- Old answer

```
if (condition1)
{
        if (condition2)
        {
                statements;
        }
}
```

- New answer
  - Use logical operators!

&&

- Logical AND
  - Both conditions are **true**
  - *Intersection* of 2 *sets*
- Example

```
if ( gender ==1 && age >=65)
        seniorFemles++;
```

- For clarity

```
( gender ==1) && (age >=65)
```

Truth table (AND)

expression 1	expression 2	expression 1 **&&** expression 2
FALSE	FALSE	FALSE
FALSE	TRUE	FALSE
TRUE	FALSE	FALSE
TRUE	TRUE	TRUE

||

- Logical OR
  - Either or both conditions are true
  - Union of 2 sets
- Example

```
if ( (semesterAvg =>90) || (FinalExam>=90))
        cout << "You get grade A";
```

Truth table (OR)

expression 1	expression 2	expression 1 || expression 2
FALSE	FALSE	FALSE
FALSE	TRUE	TRUE
TRUE	FALSE	TRUE
TRUE	TRUE	TRUE

!

- Logical NOT
  - Reverse a condition
- Example

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

- Truth table

expression	! expression
FALSE	TRUE
TRUE	FALSE

Review: structured programming

- Only 3 forms of control
  - Sequence
  - Selection
  - Repetition
- 3 types of selections
  - If (single selection)
  - If…else (double selection)
  - Switch (multiple selection)

Review: structured programming

- 3 types of repetitions
  - While
  - Do while
  - For
- C++ program can be written with only
  - Sequence
  - Control statements
    - If
    - while

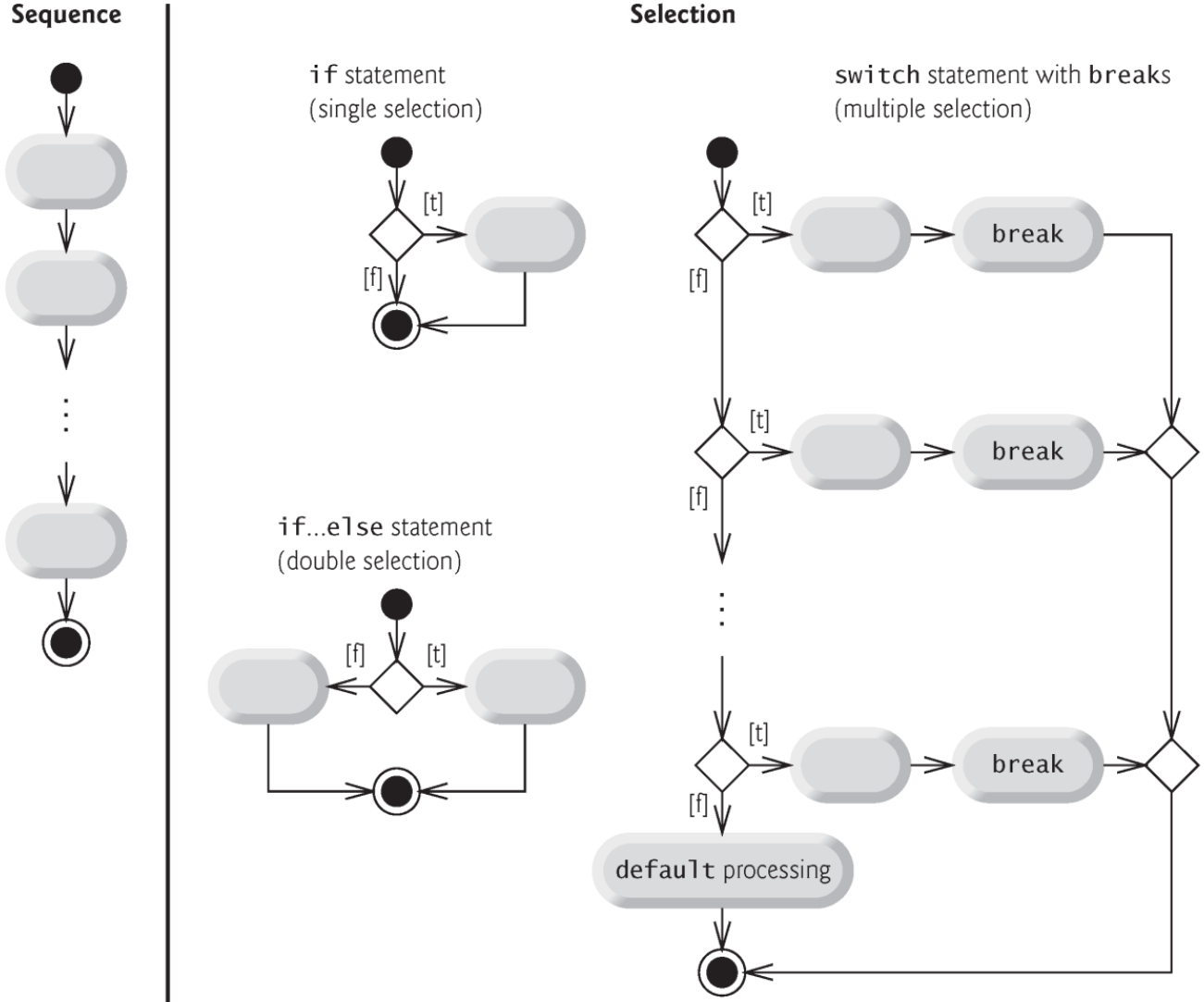


Fig. 4.18 | C++'s single-entry/single-exit sequence, selection and repetition statements. (Part 1 of 2.)

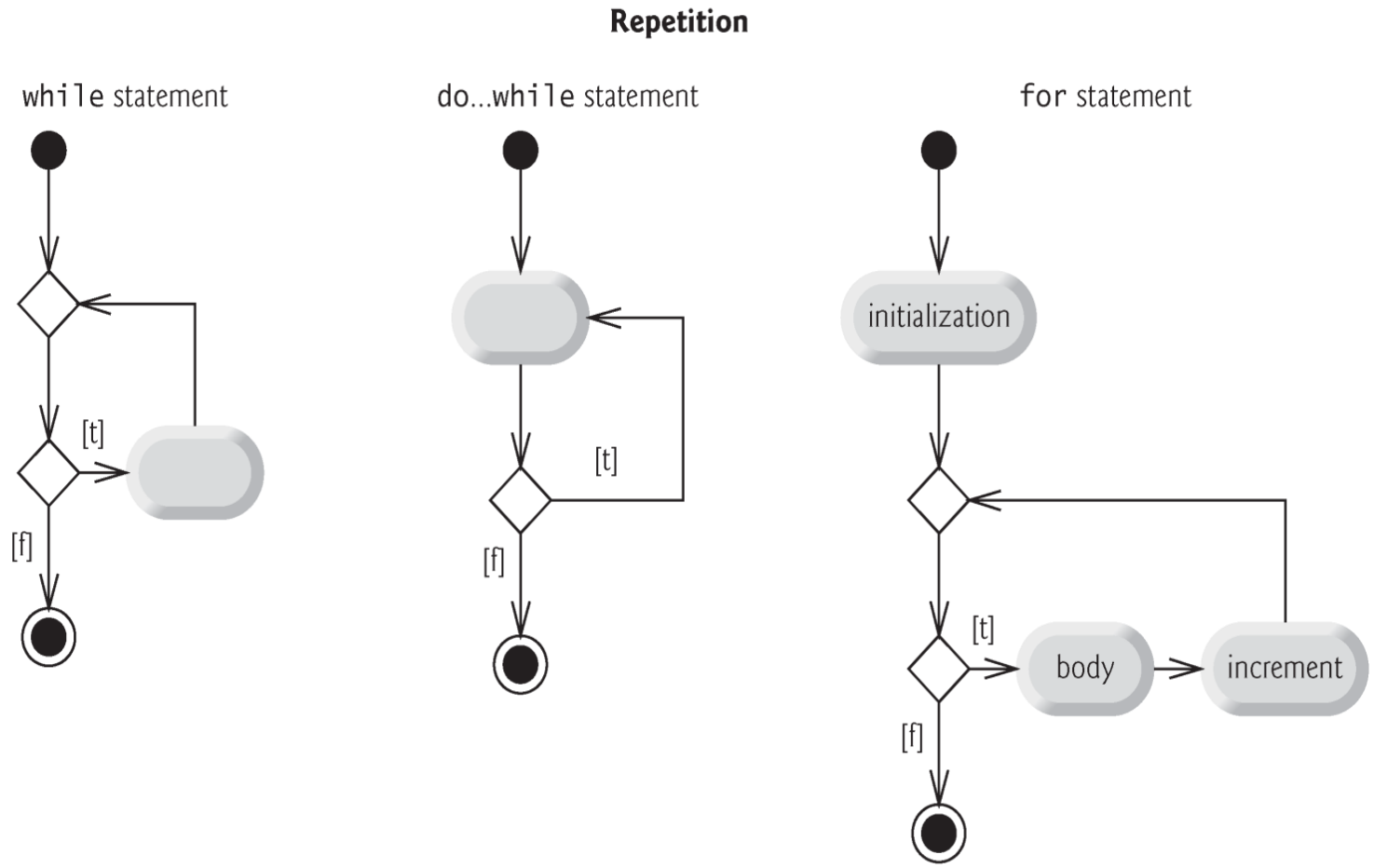


Fig. 4.18 | C++'s single-entry/single-exit sequence, selection and repetition statements. (Part 2 of 2.)

Review: structured programming

- Note that while is sufficient to provide all 3 types of repetition
- Note that if is sufficient to provide all 3 types of selection
- But…
  - You should select the appropriate type to use

Chapter 3: Control Statements (I)

- Concepts
  - Algorithm
  - Pseudocode
- Selection
  - Single selection
    - If
  - Double selection
    - If … else
- Repetition
  - while
- Algorithms
  - Counter-controlled repetition
  - Sentinel-controlled repetition
  - Nested control structure
- Operators
  - Assignment operators
    - += *=
  - Increment/decrement operators
    - ++ --
  - Logic operator
    - && || !

Chapter 4: Control Statements (II)

- Repetition
  - For
  - Do … while
- Selection
  - Multiple selection
    - switch
- More control statement
  - break
  - Continue
- Concepts
  - Structured programming
  - = v.s. ==