4

Control Statements: Part 1

Let's all move one place on.

— Lewis Carroll

The wheel is come full circle.

— William Shakespeare

How many apples fell on Newton's head before he took the hint!

— Robert Frost

All the evolution we know of proceeds from the vague to the definite.

— Charles Sanders Peirce

OBJECTIVES

In this chapter you will learn:

- Basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` and `if...else` selection statements to choose among alternative actions.
- To use the `while` repetition statement to execute statements in a program repeatedly.
- Counter-controlled repetition and sentinel-controlled repetition.
- To use the increment, decrement and assignment operators.

Outline

4.1 Introduction
4.2 Algorithms
4.3 Pseudocode
4.4 Control Structures
4.5 `if` Selection Statement
4.6 `if...else` Double-Selection Statement
4.7 `while` Repetition Statement
4.8 Formulating Algorithms: Counter-Controlled Repetition
4.9 Formulating Algorithms: Sentinel-Controlled Repetition
4.10 Formulating Algorithms: Nested Control Statements
4.11 Assignment Operators
4.12 Increment and Decrement Operators
4.13 (Optional) Software Engineering Case Study: Identifying Class Attributes in the ATM System
4.14 Wrap-Up

4.1 Introduction

- **Before writing a program**
  - **Have a thorough understanding of problem**
  - **Carefully plan your approach for solving it**
- **While writing a program**
  - **Know what "building blocks" are available**
  - **Use good programming principles**

4.2 Algorithms

- **Algorithms**
  - **The actions to execute**
  - **The order in which these actions execute**
- **Program control**
  - **Specifies the order in which actions execute in a program**
  - **Performed in C++ with control statements**

4.3 Pseudocode

- **Pseudocode**
  - **Artificial, informal language used to develop algorithms**
    - **Used to think out program before coding**
      - **Easy to convert into C++ program**
  - **Similar to everyday English**
    - **Only executable statements**
      - **No need to declare variables**
  - **Not executed on computers**

```
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
```

Fig. 4.1 | Pseudocode for the addition program of Fig. 2.5.

4.4 Control Structures

- **Sequential execution**
  - **Statements executed in sequential order**
- **Transfer of control**
  - **Next statement executed is *not* the next one in sequence**
- **Structured programming**
  - **Eliminated `goto` statements**

4.4 Control Structures (Cont.)

- **Only three control structures needed**
  - **No `goto` statements**
  - **Demonstrated by Böhm and Jacopini**
  - **Three control structures**
    - **Sequence structure**
      - **Programs executed sequentially by default**
    - **Selection structures**
      - `if`, `if…else`, `switch`
    - **Repetition structures**
      - `while`, `do…while`, `for`

4.4 Control Structures (Cont.)

- **UML activity diagram**
  - **Models the workflow**
    - **Action state symbols**
      - **Rectangles with curved sides**
    - **Small circles**
      - **Solid circle is the initial state**
      - **Solid circle in a hollow circle is the final state**
    - **Transition arrows**
      - **Represent the flow of activity**
    - **Comment notes**
      - **Connected to diagram by dotted lines**

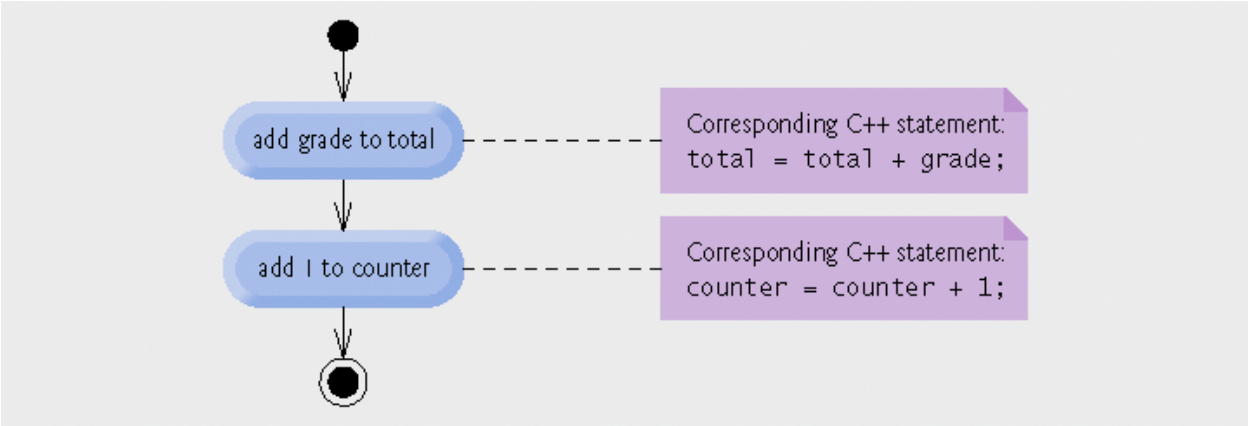


Fig. 4.2 | Sequence-structure activity diagram.

4.4 Control Structures (Cont.)

- **Single-entry/single-exit control statements**
  - **Three types of control statements**
    - **Sequence statement**
    - **Selection statements**
    - **Repetition statements**
  - **Combined in one of two ways**
    - **Control statement stacking**
      - **Connect exit point of one to entry point of the next**
    - **Control statement nesting**

C++ Keywords

Keywords common to the C and C++ programming languages

auto	break	case	char	Const
Continue	default	do	double	Else
enum	extern	float	for	Goto
if	int	long	register	Return
short	signed	sizeof	static	Struct
switch	typedef	union	unsigned	Void
volatile	while			

C++-only keywords

and	and_eq	asm	bitand	Bitor
bool	catch	class	compl	const_cast
delete	dynamic_cast	explicit	export	False
friend	inline	mutable	namespace	New
not	not_eq	operator	or	or_eq
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	Try
typeid	typename	using	virtual	wchar_t
xor	xor_eq			

Fig. 4.3 | C++ keywords.

Common Programming Error 4.1

Using a keyword as an identifier is a syntax error.

Common Programming Error 4.2

Spelling a keyword with any uppercase letters is a syntax error. All of C++'s keywords contain only lowercase letters.

Software Engineering Observation 4.1

Any C++ program we will ever build can be constructed from only seven different types of control statements (sequence, `if`, `if...else`, `switch`, `while`, `do...while` and `for`) combined in only two ways (control-statement stacking and control-statement nesting).

4.5 `if` Selection Statement

- **Selection statements**
  - **Choose among alternative courses of action**
  - **Pseudocode example**
    - ***If student's grade is greater than or equal to 60***

      Print "Passed"
      - **If the condition is `true`**
        - **Print statement executes, program continues to next statement**
      - **If the condition is `false`**
        - **Print statement ignored, program continues**
      - **Indenting makes programs easier to read**
        - **C++ ignores white-space characters**

4.5 `if` Selection Statement (Cont.)

- **Selection statements (Cont.)**
  - **Translation into C++**
    -
```
if ( grade >= 60 )
   cout << "Passed";
```
  - **Any expression can be used as the condition**
    - **If it evaluates to false, it is treated as false**
- **Diamond symbol in UML modeling**
  - **Indicates decision is to be made**
  - **Contains guard conditions**
    - **Test condition**
    - **Follow correct path**

Good Programming Practice 4.1

Consistently applying reasonable indentation conventions throughout your programs greatly improves program readability. We suggest three blanks per indent. Some people prefer using tabs but these can vary across editors, causing a program written on one editor to align differently when used with another.

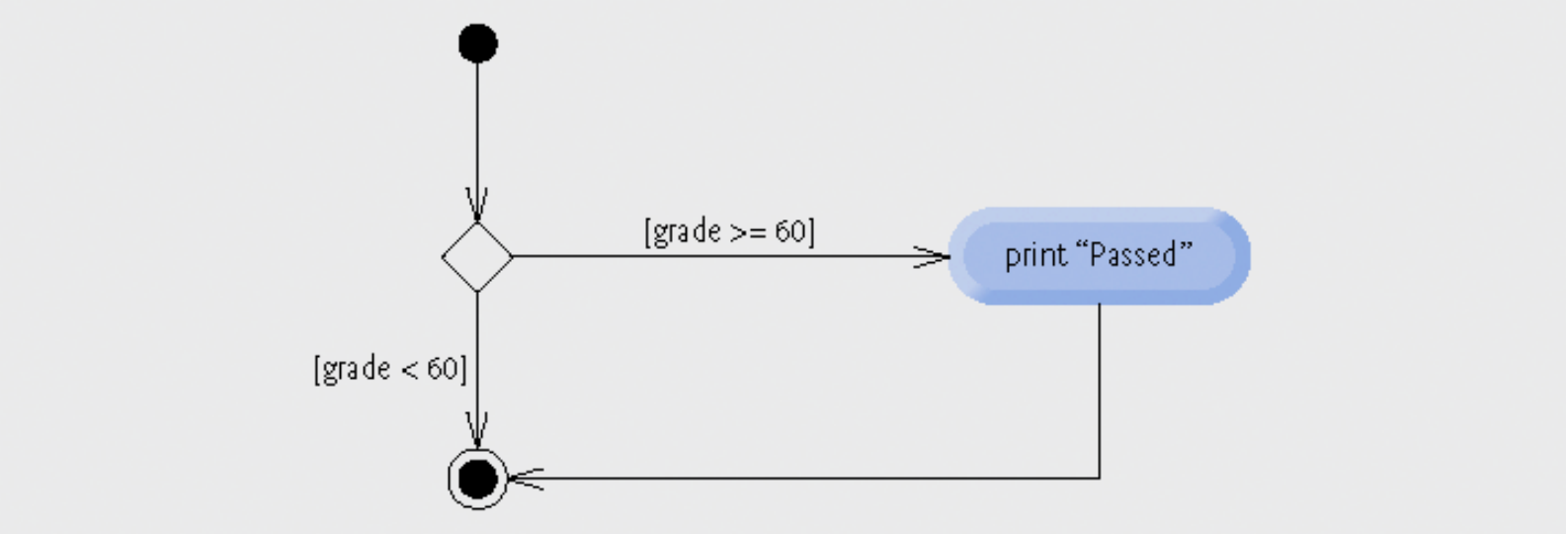


Fig. 4.4 | `if` single-selection statement activity diagram.

Portability Tip 4.1

For compatibility with earlier versions of C, which used integers for Boolean values, the `bool` value `true` also can be represented by any nonzero value (compilers typically use 1) and the `bool` value `false` also can be represented as the value zero.

4.6 `if…else` Double-Selection Statement

- `if`
  - **Performs action if condition true**
- `if…else`
  - **Performs one action if condition is true, a different action if it 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";
```

Good Programming Practice 4.2

Indent both body statements of an `if...else` statement.

Good Programming Practice 4.3

If there are several levels of indentation, each level should be indented the same additional amount of space.

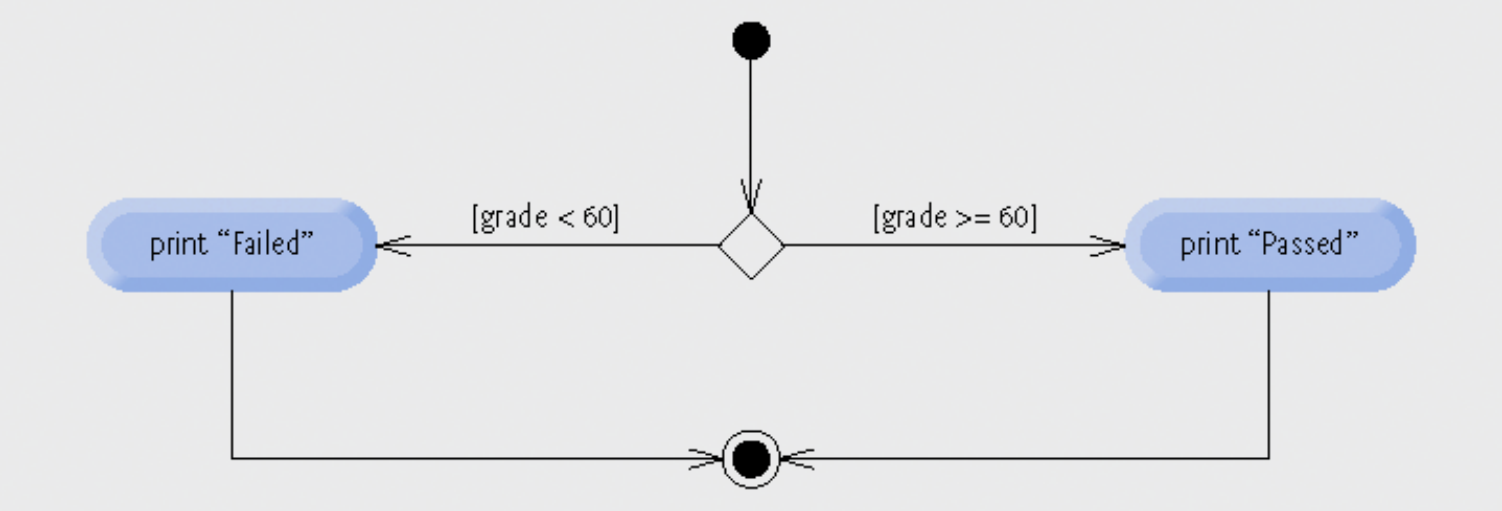
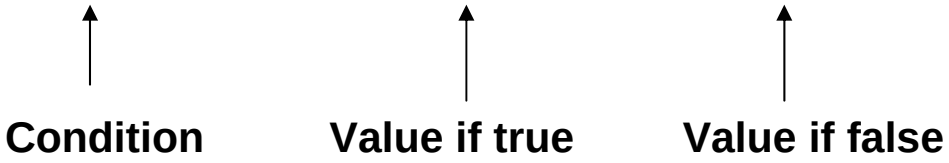


Fig. 4.5 | `if...else` double-selection statement activity diagram.

4.6 `if…else` Double-Selection Statement (Cont.)

- **Ternary conditional operator (`?:`)**
  - **Three arguments (condition, value if `true`, value if `false`)**
- **Code could be written:**
  - `cout << ( grade >= 60 ? “Passed” : “Failed” );`



Error-Prevention Tip 4.1

To avoid precedence problems (and for clarity), place conditional expressions (that appear in larger expressions) in parentheses.

4.6 `if…else` Double-Selection Statement (Cont.)

- **Nested `if…else` statements**
  - **One inside another, test for multiple cases**
  - **Once a condition met, other statements are skipped**
  - **Example**
    - *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"

4.6 `if…else` Double-Selection Statement (Cont.)

- Nested `if…else` statements (Cont.)
  - Written In C++
    -
```
if ( studentGrade >= 90 )
   cout << "A";
else
   if (studentGrade >= 80 )
      cout << "B";
   else
      if (studentGrade >= 70 )
         cout << "C";
      else
         if ( studentGrade >= 60 )
            cout << "D";
         else
            cout << "F";
```

4.6 `if…else` Double-Selection Statement (Cont.)

- **Nested `if…else` statements (Cont.)**
  - **Written In C++ (indented differently)**

```
if ( studentGrade >= 90 )
   cout << "A";
else if (studentGrade >= 80 )
   cout << "B";
else if (studentGrade >= 70 )
   cout << "C";
else if ( studentGrade >= 60 )
   cout << "D";
else
   cout << "F";
```

Performance Tip 4.1

A nested `if...else` statement can perform much faster than a series of single-selection `if` statements because of the possibility of early exit after one of the conditions is satisfied.

Performance Tip 4.2

In a nested `if...else` statement, test the conditions that are more likely to be `true` at the beginning of the nested `if...else` statement. This will enable the nested `if...else` statement to run faster and exit earlier than testing infrequently occurring cases first.

4.6 `if…else` Double-Selection Statement (Cont.)

- **Dangling-`else` problem**
  - **Compiler associates `else` with the immediately preceding `if`**
  - **Example**

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

  - **Compiler interprets as**

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

4.6 `if…else` Double-Selection Statement (Cont.)

- **Dangling-`else` problem (Cont.)**
  - **Rewrite with braces (`{}`)**
    -
```
if ( x > 5 )
{
   if ( y > 5 )
      cout << "x and y are > 5";
}
else
   cout << "x is <= 5";
```
      - **Braces indicate that the second `if` statement is in the body of the first and the `else` is associated with the first `if` statement**

4.6 `if…else` Double-Selection Statement (Cont.)

- **Compound statement**
  - **Also called a block**
    - **Set of statements within a pair of braces**
    - **Used to include multiple statements in an `if` body**
  - **Example**

```
if ( studentGrade >= 60 )
   cout << "Passed.\n";
else
{
   cout << "Failed.\n";
   cout << "You must take this course again.\n";
}
```

  - **Without braces,**

```
cout << "You must take this course again.\n";
```

    always executes

Software Engineering Observation 4.2

A block can be placed anywhere in a program that a single statement can be placed.