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

Algorithms

All computing problems

- can be solved by executing a series of actions in a specific order

Algorithm

- A procedure determining the
  - Actions to be executed
  - Order in which these actions are to be executed

  Example of a Junior Executive – The order is important!!!
  1) Get out of bed 2) take PJ off 3) Take a Shower
  4) Get dressed 5) eat breakfast 6) carpool to work

Program control

- Specifies the order in which statements are to executed

Pseudocode

Pseudocode

- Artificial, informal language used to develop algorithms
- Similar to everyday English
- Not actually executed on computers
- Allows us to “think out” a program before writing the code for it
- Easy to convert into a corresponding C++ program
- Consists only of executable statements

C++ Keywords

C++ keywords

- Cannot be used as identifiers or variable names.

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				
asm	bool	catch	class	const_cast
delete	dynamic_cast	explicit	false	friend
inline	mutable	namespace	new	operator
private	protected	public	reinterpret_cast	static_cast
template	this	throw	true	try
typeid	typename	using	virtual	wchar_t

Control Structures

Control Structures – 3 types, 7 (total) structures

- **Sequence** structure
  - Built into C++. Programs executed sequentially by default
  - Transfer of control

    When the next statement executed is not the next one in sequence

- **Selection** structures
  - C++ has three types

    `if`

    `if/else`

    `switch`
- **Repetition** structures
  - C++ has three types

    `while`

    `do/while`

    `for`

Control Structures
if Selection Structure

Selection structures

- Used to choose among alternative courses of action
- `if (condition is true) Then`
  - `Execute this statement`

Pseudocode example:

If (student's grade is greater than or equal to 60) Then
Print "Passed" ← **will only print if Grade >= 60**
Print " All Done" ← **will always print this**

- If the condition is **`true`**
  - print statement is executed and program goes on to next statement **(Print " All Done")**
- If the condition is **`false`**
  - **Print "Passed"** statement is **SKIPPED** and the program goes onto the next statement **(Print " All Done")**
- Indenting makes programs easier to read
  - C++ ignores whitespace characters

Decision Making: If, if/else, else

`if` structure

- Test if condition is true or false. If condition is true execute, otherwise ignore

if (conditional statement is true)

execute one statement

if (conditional statement is true)
{

execute all statements with this block

……

……

……

}

{ means START of a block of statements

} means END of a block of statements

Translation of pseudocode statement into C++

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

The if Selection Structure

Flowchart of pseudocode statement

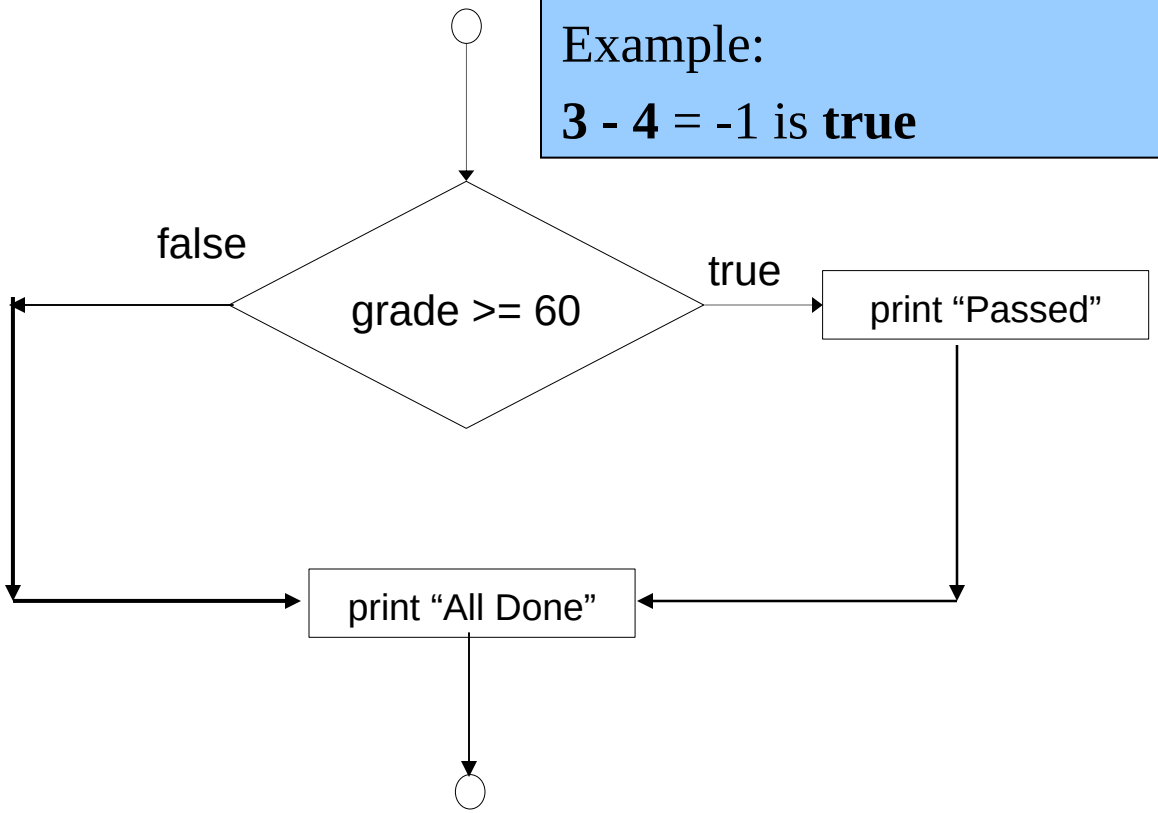
A decision can be made on any expression.

zero - **false**

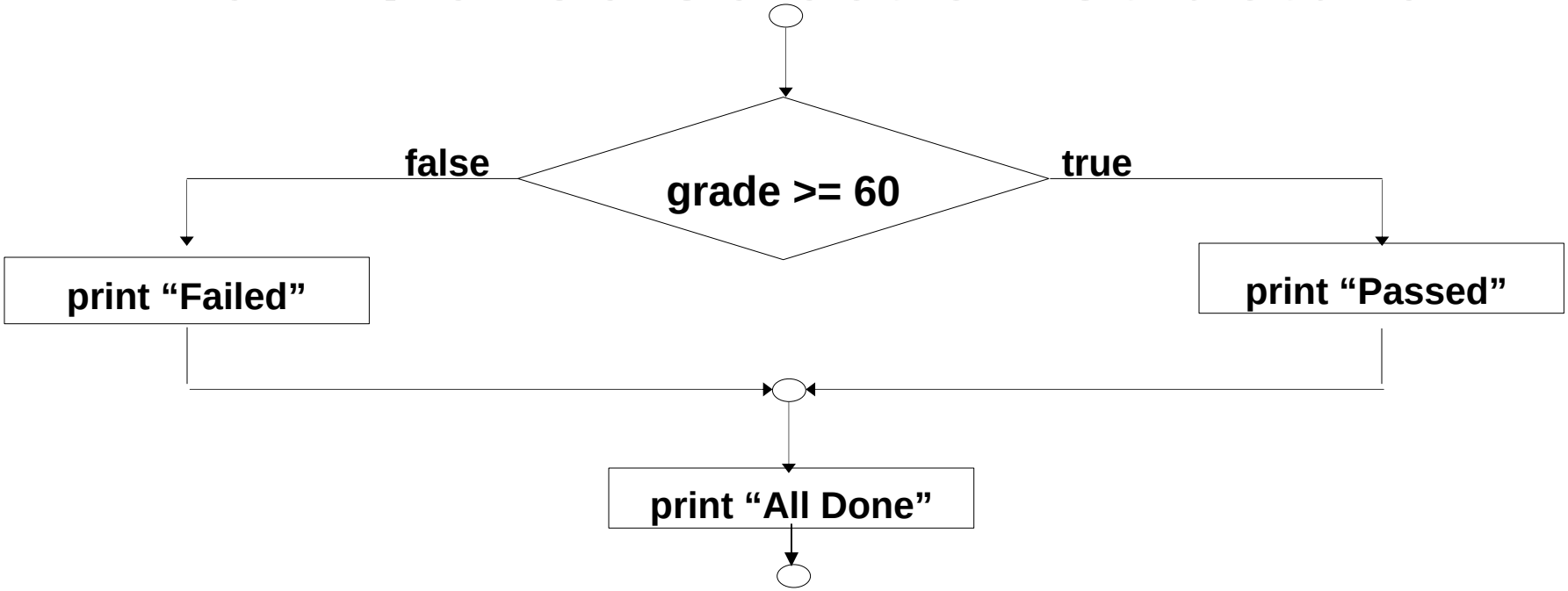
nonzero - **true**

Example:

3 - 4 = -1 is **true**



The `if/else` Selection Structure



Ternary conditional operator (**`?:`**)

- Takes three arguments (condition, value if **`true`**, value if **`false`**)
- Very important and used a lot including in your exams

```
cout << ( grade >= 60 ? “Passed” : “Failed” );
bool FAILED =  grade < 60 ? 1 : 0 ;
```

The `if/else` Selection Structure

Nested **`if/else`** structures

- Test for multiple cases by placing **`if/else`** selection structures inside **`if/else`** selection structures

```
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"
```

- Once a condition is met, the rest of the statements are skipped
- The order of statements is very important!!!!

The if/else Selection Structure

Compound statement

- Set of statements within a pair of braces
- A block of statements

Example

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

- Without the braces,

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

would be automatically executed even if the student passed

Block

- Compound statements with declarations

Switch Statement

switch Statement

SYNTAX

```
switch (Controlling_Expression)
{
    case Constant_1:
        Statement_Sequence_1
        break;
    case Constant_2:
        Statement_Sequence_2
        break;
                    .
                    .
                    .
    case Constant_n:
        Statement_Sequence_n
        break;
    default:
        Default_Statement_Sequence
}
```

You need not place a **break** *statement in each case. If you omit a* **break**, *that case continues until a* **break** *(or the end of the* **switch** *statement) is reached.*

Example – Page 64

EXAMPLE

```
int vehicleClass;
double toll;
cout << "Enter vehicle class: ";
cin >> vehicleClass;

switch (vehicleClass)
{
    case 1:
        cout << "Passenger car.";
        toll = 0.50;
        break;
    case 2:
        cout << "Bus.";
        toll = 1.50;
        break;
    case 3:
        cout << "Truck.";
        toll = 2.00;
        break;
    default:
        cout << "Unknown vehicle class!";
  }
```

If you forget this **break**, *then passenger cars will pay $1.50.*

Syntax Errors Vs Logic Errors

Syntax errors

- Errors caught by compiler

Logic errors

- Errors which have their effect at execution time
  - Non-fatal logic errors

    program runs, but has incorrect output
  - Fatal logic errors

    program exits prematurely

The while Repetition Structure

Repetition structure

- An action is repeated while condition remains true
- Psuedocode

while (there are more items on my shopping list) ← **condition**
Purchase next item and cross it off my list

- **`while`** loop repeated until condition becomes false
- Example

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

Keep repeating **`'product = 2 * product'`** until value of product becomes 1000 or greater

Formulating Algorithms

Counter-controlled repetition

- Loop repeated until counter reaches a certain value.

Definite repetition

- Number of repetitions is known

Example

A class of ten students took a quiz. The grades (integers in the range 0 to 100) for this quiz are available to you. Determine the class average on the quiz. (From Deitel Text)

Design and Algorithm

What is required for us to do?

- Calculate the average and display it
- What is average?
  - Sum of all grades divided by the number of students

What do we need to solve this problem?

- We need grades, sum of all the grades, number of students, average

First pass of the pseudocode

- We will first get (read) ten grades one at a time, adding each grade to "sum" of all the grades. Then divide the "sum" by number of students and display the average.

Counter-Controlled Repetition

Second pass of Pseudocode

```
Set total to zero
Set grade counter to one
While (grade counter is less than or equal to ten)
    Input the next grade
    Add the grade into the total
    Increment the grade counter
repeat

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

Following is the C++ code for this example

```
#include <iostream>
using namespace std;
int main()
{
      int total,       // sum of grades
         gradeCounter, // number of grades entered
          grade,       // one grade
         average;     // average of grades

      // initialization
      total = 0;
      gradeCounter = 1;       // prepare to loop

     // processing phase
     while ( gradeCounter <= 10 ) { // loop 10 times
       cout << "Enter grade: ";    // prompt for input
       cin >> grade;               // input grade
       total = total + grade;      // add grade to
       gradeCounter = gradeCounter + 1;  // increment
     }

     // termination phase
     average = total / 10;  // calculate average, int division
     cout << "Class average is " << average << endl;
return 0;   // indicate program ended successfully
}
```

```
Enter grade: 98
Enter grade: 76
Enter grade: 71
Enter grade: 87
Enter grade: 83
Enter grade: 90
Enter grade: 57
Enter grade: 79
Enter grade: 82
Enter grade: 94
Class average is 81
```

Sentinel-Controlled Repetition

Suppose the problem becomes:

Develop a class-averaging program that will process an arbitrary number of grades each time the program is run

- Unknown number of students - how will the program know to end?

Sentinel value

- Indicates “end of data entry”
- Loop ends when sentinel value is inputted
- Sentinel value is chosen so it cannot be confused with a regular input (such as -1 in this case)

Sentinel-Controlled Repetition

Top-down, stepwise refinement

- begin with a pseudocode representation of the top:

  Determine the class average for the quiz

- Divide top into smaller tasks and list them in order:

  Initialize variables

  Input, sum and count the quiz grades

  Calculate and print the class average

Formulating Algorithms with Top-Down, Stepwise Refinement

Many programs can be divided into three phases:

- **Initialization**
  - Initializes the program variables
- **Processing**
  - Inputs data values and adjusts program variables accordingly
- **Termination**
  - Calculates and prints the final results.
  - Helps the breakup of programs for top-down refinement.

Refine the initialization phase from

Initialize variables

to

Initialize total to zero

Initialize counter to zero

Formulating Algorithms with Top-Down, Stepwise Refinement

Refine

Input, sum and count the quiz grades

to

Input the first grade (possibly the sentinel)
While the user has not as yet entered the sentinel
Add this grade into the running total
Add one to the grade counter
Input the next grade (possibly the sentinel)

Refine

Calculate and print the class average

to

If the counter is not equal to zero
Set the average to the total divided by the counter
Print the average
Else
Print "No grades were entered"

```
1  // Fig. 2.9: fig02_09.cpp
2  // Class average program with sentinel-controlled
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8  using std::ios;
9
10 #include <iomanip>
11
12 using std::setprecision;
13 using std::setiosflags;
14
15 int main()
16 {
17    int total,        // sum of grades
18        gradeCounter, // number of grades entered
19        grade;        // one grade
20    double average;   // number with decimal point for
21
22    // initialization phase
23    total = 0;
24    gradeCounter = 0;
25
26    // processing phase
27    cout << "Enter grade, -1 to end: ";
28    cin >> grade;
29
30    while ( grade != -1 ) {
```

Data type **double** used to represent decimal numbers.

```
31          total = total + grade;
32          gradeCounter = gradeCounter + 1;
33          cout << "Enter grade, -1 to end: ";
34          cin >> grade;
35       }
36
37       // termination phase
38       if ( gradeCounter != 0 ) {
39          average = static_cast< double >( total ) / gradeCounter;
40          cout << "Class average is " << setprecision( 2 )
41               << setiosflags( ios::fixed | ios::showpoint )
42               << average << endl;
43       }
44
45
46
47
48
```

`static_cast<double>()` - treats **`total`** as a **`double`** temporarily. This is an explicit conversion Required because dividing two integers truncates the remainder.
`gradeCounter` is an **`int`**, but it gets *promoted* to **`double`**. Implicit conversion

`setiosflags(ios::fixed | ios::showpoint)` - stream

ers with a fixed number of decimal

decimal point and trailing zeros, even if

unnecessary: **66** printe

`|` - separates multip

`setprecision(2)` - prints only two digits past decimal point.

Programs that use this must include **`<iomanip>`**

```
Enter grade, -1 to end: 75
Enter grade, -1 to end: 94
Enter grade, -1 to end: 97
Enter grade, -1 to end: 88
Enter grade, -1 to end: 70
Enter grade, -1 to end: 64
Enter grade, -1 to end: 83
Enter grade, -1 to end: 89
Enter grade, -1 to end: -1
Class average is 82.50
```

I/O Manipulators

Requries header and optionally the std namespace

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

resetiosflags (long flag)- Clears the specified flags

setbase - Set base for integers (10 or 16).

setfill - Sets the character that will be used to fill spaces in a right-justified display.

setiosflags - Sets the specified flags.

setprecision - Sets the precision for floating-point values.

setw - Specifies the width of the display field.

Examples

```
void main()
{
   double i = 20.00000;
   int j = 20;
   cout << setprecision( 2 ) <<
        setiosflags(ios::fixed | ios::showpoint);
   cout << "num is " << i << endl;
   cout << "num is " << setbase(16) << j << setbase(10) <<
        endl;
   cout << resetiosflags(ios::fixed | ios::showpoint);
   cout << "nums are " << i << ", " << j << endl;
   return;
}
```

Nested control structures

Problem:

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".

We can see that

- The program must process 10 test results. A counter-controlled loop will be used.
- Two counters can be used—one to count the number of students who passed the exam and one to count the number of students who failed the exam.
- Each test result is a number—either a 1 or a 2. If the number is not a 1, we assume that it is a 2.

Top level outline:

Analyze exam results and decide if tuition should be raised

Nested control structures

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

to

Initialize passes to zero

Initialize failures to zero

Initialize student counter to one

Nested control structures

Refine

Input the ten quiz grades and count passes and failures

to

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 and continue

Refine

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

to

Print the number of passes
Print the number of failures
If more than eight students passed
Print "Raise tuition"

```
1  // Fig. 2.11: fig02_11.cpp
2  // Analysis of examination results
3  #include <iostream>
4
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    // initialize variables in declarations
12    int passes = 0,           // number of passes
13        failures = 0,         // number of failures
14        studentCounter = 1,   // student counter
15        result;               // one exam result
16
17    // process 10 students; counter-controlled loop
18    while ( studentCounter <= 10 ) {
19       cout << "Enter result (1=pass,2=fail): ";
20       cin >> result;
21
22       if ( result == 1 )        // if/else nested in while
23          passes = passes + 1;
```

```
24       else
25          failures = failures + 1;
26
27       studentCounter = studentCounter + 1;
28    }
29
30    // termination phase
31    cout << "Passed " << passes << endl;
32    cout << "Failed " << failures << endl;
33
34    if ( passes > 8 )
35       cout << "Raise tuition " << endl;
36
37    return 0;   // successful termination
38 }
```

Program 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
```

Assignment Operators

Assignment expression abbreviations

`c = c + 3;` can be abbreviated as `c += 3;` using the addition assignment operator (also called C++ shortcut)

Statements of the form

```
variable = variable operator expression;
```

can be rewritten as

```
variable operator= expression;
```

Examples of other assignment operators include:

```
d -= 4     (d = d - 4)
e *= 5     (e = e * 5)
f /= 3     (f = f / 3)
g %= 9     (g = g % 9)
```

Increment and Decrement Operators

Increment operator (**++**)

Instead of **`c += 1`** , you can write **`c++ or ++c`**

Decrement operator (**`--`**)

Instead of **`c -= 1,`** you can write **`c-- or --c;`**

Preincrement

- When the operator is used before the variable (**`++c`** or **`--c`**)
- Variable is changed, then the expression it is in is evaluated.

- Postincrement
  - When the operator is used after the variable (**`c++`** or **`c--`**)
  - Expression the variable is in executes, then the variable is changed.

Example **`int c = 5;`**

- **`cout << ++c;`** prints out **6** (**c** is changed before **`cout`** is executed)
- **`cout << c++;`** prints out **5** (**`cout`** is executed before the increment. **c** now has the value of **6**)

Increment and Decrement Operators

When Variable is not in an expression

- Preincrementing and postincrementing have the same effect.

```
++c;
cout << c;
```

and

```
c++;
cout << c;
```

have the same effect.

Essentials of Counter-Controlled Repetition

Counter-controlled repetition requires:

- The name of a control variable (or loop counter).
- The initial value of the control variable.
- The condition that tests for the final value of the control variable (i.e., whether looping should continue).
- The increment (or decrement) by which the control variable is modified each time through the loop.

Example:

```
int counter =1;          //initialization
while (counter <= 10)//repetition condition
{   cout << counter << endl;
     ++counter;          //increment counter
}
```

Essentials of Counter-Controlled Repetition

The declaration

```
int counter = 1;
```

- Names **`counter`**
- Declares **`counter`** to be an integer
- Reserves space for **`counter`** in memory
- Sets **`counter`** to an initial value of **`1`**

The for Repetition Structure

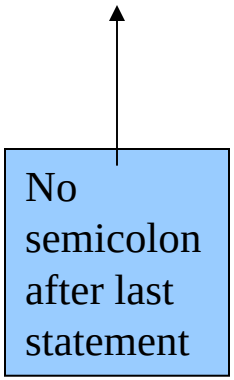
The general format when using **`for`** loops is

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

Example:

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

- Prints the integers from one to ten



The for Repetition Structure

For loops can usually be rewritten as **while** loops:

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

Initialization and increment as comma-separated lists

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

Examples Using the for Structure

- Program to sum the even numbers from 2 to 100

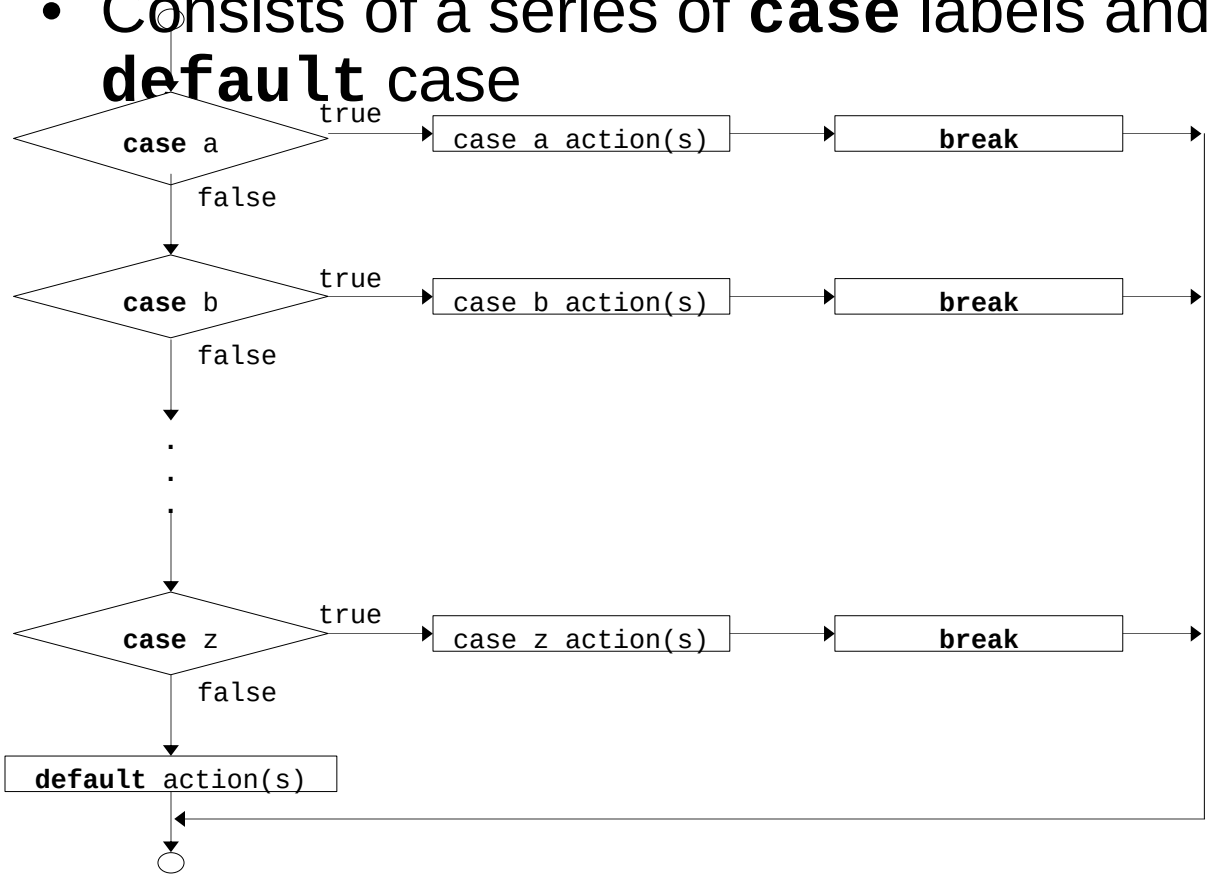
```
1 // Fig. 2.20: fig02_20.cpp
2 // Summation with for
3 #include <iostream>
4
5 using std::cout;
6 using std::endl;
7
8 int main()
9 {
10    int sum = 0;
11
12    for ( int number = 2; number <= 100; number += 2 )
13       sum += number;
14
15    cout << "Sum is " << sum << endl;
16
17    return 0;
18 }
```

```
Sum is 2550
```

The switch Multiple-Selection Structure

`switch`

- Useful when variable or expression is tested for multiple values
- Consists of a series of **`case`** labels and an optional **`default`** case



```
1  // Fig. 2.22: fig02_22.cpp
2  // Counting letter grades
3  #include <iostream>
5  using std::cout;
6  using std::cin;
7  using std::endl;
8
9  int main()
10 {
11    int grade,        // one grade
12        aCount = 0,   // number of A's
13        bCount = 0,   // number of B's
14        cCount = 0,   // number of C's
15        dCount = 0,   // number of D's
16        fCount = 0;   // number of F's
17
18    cout << "Enter the letter grades." << endl
19        << "Enter the EOF character to end input." << endl;
20
21    while ( ( grade = c
22
23       switch ( grade ) {     // switch nested in while
24
25          case 'A':  // grade was uppercase A
26          case 'a':  // or lowercase a
27             ++aCount;
28             break;  // necessary to exit switch
29
30          case 'B':  // grade was uppercase B
31          case 'b':  // or lowercase b
32             ++bCount;
33             break;
```

Notice how the **`case`** statement is used

```
35          case 'C':  // grade was uppercase C
36          case 'c':  // or lowercase c
37             ++cCount;
38             break;
39
40          case 'D':  // grade was upper
41          case 'd':  // or lowercase d
42             ++dCount;
43             break;
44
45          case 'F':  // grade was upper
46          case 'f':  // or lowercase f
47             ++fCount;
48             break;
49
50          case '\n': // ignore newlines,
51          case '\t': // tabs,
52          case ' ':  // and spaces in
53             break;
54
55          default:   // catch all other characters
56             cout << "Incorrect letter grade entered."
57                  << " Enter a new grade." << endl;
58             break;  // optional
59       }
60    }
61
62    cout << "\n\nTotals for each letter grade are:"
63         << "\nA: " << aCount
64         << "\nB: " << bCount
65         << "\nC: " << cCount
66         << "\nD: " << dCount
67         << "\nF: " << fCount << endl;
68
69    return 0;
70 }
```

`break` causes **`switch`** to end and the program continues with the first statement after the **`switch`** structure.

Notice the **`default`** statement.

```
Enter the letter grades.
Enter the EOF character to end input.
a
B
c
C
A
d
f
C
E
Incorrect letter grade entered. Enter a new grade.
D
A
b

Totals for each letter grade are:
A: 3
B: 2
C: 3
D: 2
F: 1
```

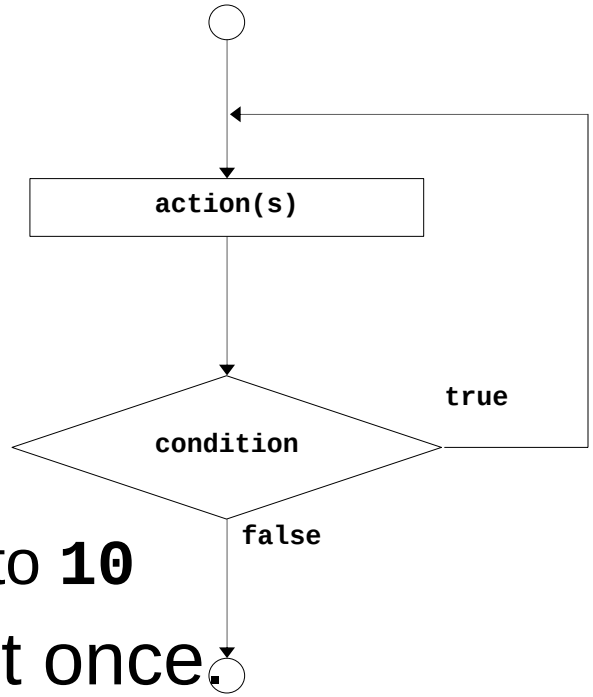
The do/while Repetition Structure

The **`do/while`** repetition structure is similar to the **`while`** structure,

- Condition for repetition tested after the body of the loop is executed

Format:

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



Example (letting counter = 1):

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

- This prints the integers from **`1`** to **`10`**

All actions are performed at least once.

The break and continue Statements

`Break`

- Causes immediate exit from a **`while`**, **`for`**, **`do/while`** or **`switch`** structure
- Program execution continues with the first statement after the structure
- Common uses of the **`break`** statement:
  - Escape early from a loop
  - Skip the remainder of a **`switch`** structure

The break and continue Statements

`Continue`

- Skips the remaining statements in the body of a **`while`**, **`for`** or **`do/while`** structure and proceeds with the next iteration of the loop
- In **`while`** and **`do/while`**, the loop-continuation test is evaluated immediately after the **`continue`** statement is executed
- In the **`for`** structure, the increment expression is executed, then the loop-continuation test is evaluated

Logical Operators

`&&` (logical **`AND`**)

- Returns **`true`** if both conditions are **`true`**

`||` (logical **`OR`**)

- Returns **`true`** if either of its conditions are **`true`**

`!` (logical **`NOT`**, logical negation)

- Reverses the truth/falsity of its condition
- Returns **`true`** when its condition is **`false`**
- Is a unary operator, only takes one condition

Logical operators used as conditions in loops

Expression	Result
`true && false`	`false`
`true || false`	`true`
`!false`	`true`

Equality (==) &Assignment (=) Operators

These errors are damaging because they do not ordinarily cause syntax errors.

- Recall that any expression that produces a value can be used in control structures. Nonzero values are **`true`**, and zero values are **`false`**

Example:

```
if ( payCode == 4 )
    cout << "You get a bonus!" << endl;
```

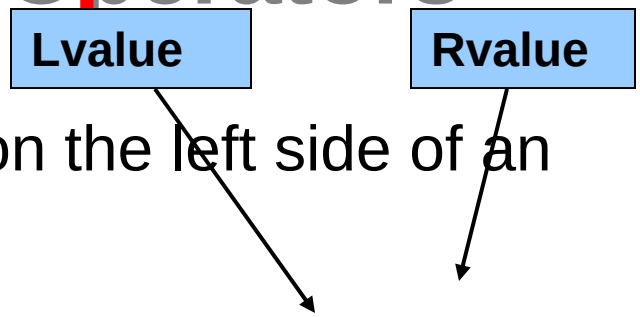
- Checks the paycode, and if it is **`4`** then a bonus is awarded

If **==** was replaced with **=**

```
if ( payCode = 4 )
   cout << "You get a bonus!" << endl;
```

- Sets **`paycode`** to **`4`**
- **`4`** is nonzero, so the expression is **`true`** and a bonus is awarded, regardless of **`paycode`**.

Confusing Equality (==) and Assignment (=) Operators



Lvalues

- Expressions that can appear on the left side of an equation
- Their values can be changed
- Variable names are a common example (as in `x = 4;`)

Rvalues

- Expressions that can only appear on the right side of an equation
- Constants, such as numbers (i.e. you cannot write `4 = x;`)

Lvalues can be used as Rvalues, but not vice versa