Programming Techniques I SCJ1013

Making Decisions

Dr Masitah Ghazali

Relational Operators & Logical Operators

Relational Operators

- Used to compare numbers to determine relative order
- Operators:

>	Greater than
<	Less than
>=	Greater than or equal to
<=	Less than or equal to
==	Equal to
!=	Not equal to

Relational Expressions

- Boolean expressions – `true` or `false`
- Examples:

`12 > 5` is `true`

`7 <= 5` is `false`

if `x` is 10, then

`x == 10` is `true`,

`x != 8` is `true`, and

`x == 8` is `false`

Relational Expressions

- Can be assigned to a variable:

```
result = x <= y;
```

- Assigns `0` for `false`, `1` for `true`
- Do not confuse `=` and `==`

Logical Operators

- Used to create relational expressions from other relational expressions
- Operators, meaning, and explanation:

&&	AND	New relational expression is true if both expressions are true
||	OR	New relational expression is true if either expression is true
!	NOT	Reverses the value of an expression – true expression becomes false, and false becomes true

Logical Operators - examples

```
int x = 12, y = 5, z = -4;
```

`(x > y) && (y > z)`	`true`
`(x > y) && (z > y)`	`false`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`

Exercise Week 7_1

- What will the following program display?

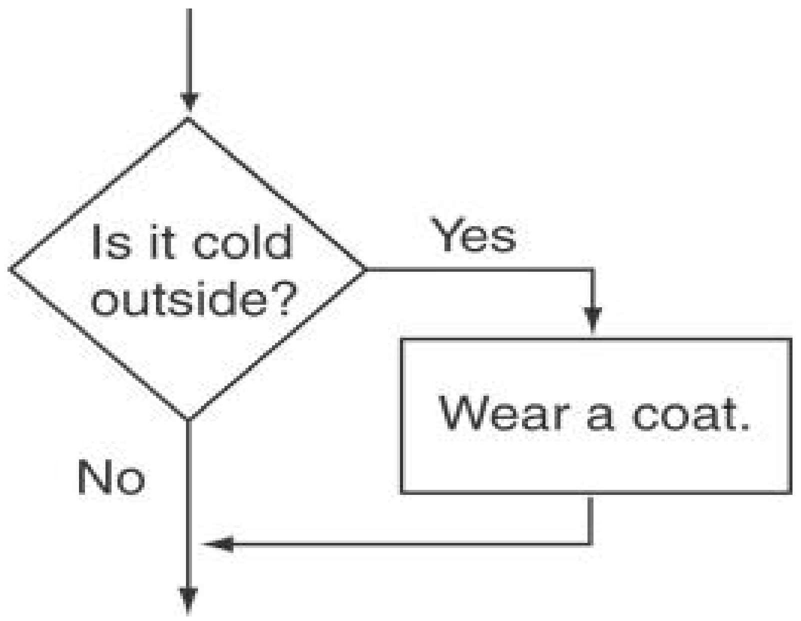
```
int main(){
  int a=0, b = 2, x = 4, y=0;

   cout << (a == b) << endl;
   cout << (a != y)<< endl;
   cout<< (b <= x) <<endl;
   cout << (y > a) << endl;

   return 0;
}
```

The `if` Statement

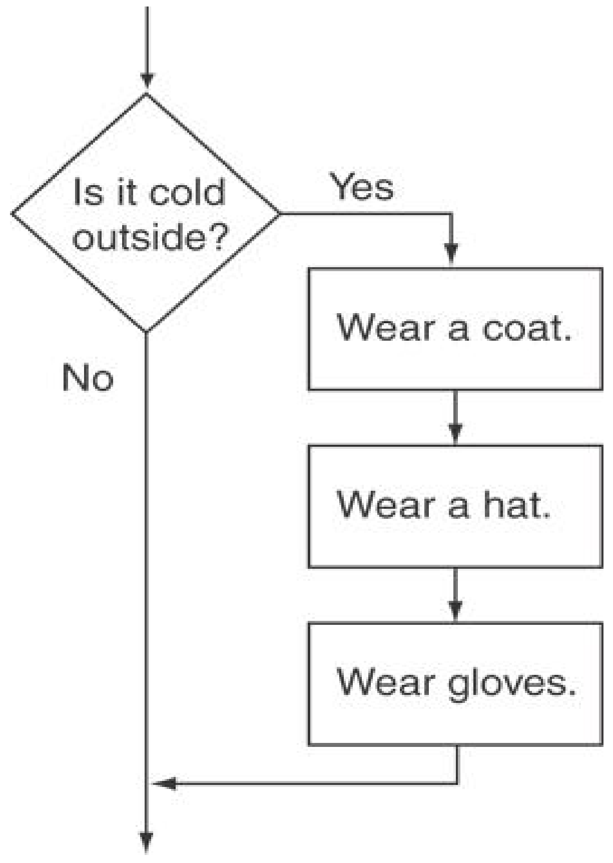
The `if` Statement

- Allows statements to be conditionally executed or skipped over
- Models the way we mentally evaluate situations:
  - "If it is raining, take an umbrella."
  - "If it is cold outside, wear a coat."

Flowchart for Evaluating a Decision



Flowchart for Evaluating a Decision



The `if` Statement

- General Format:

```
  if (expression)
statement;
```

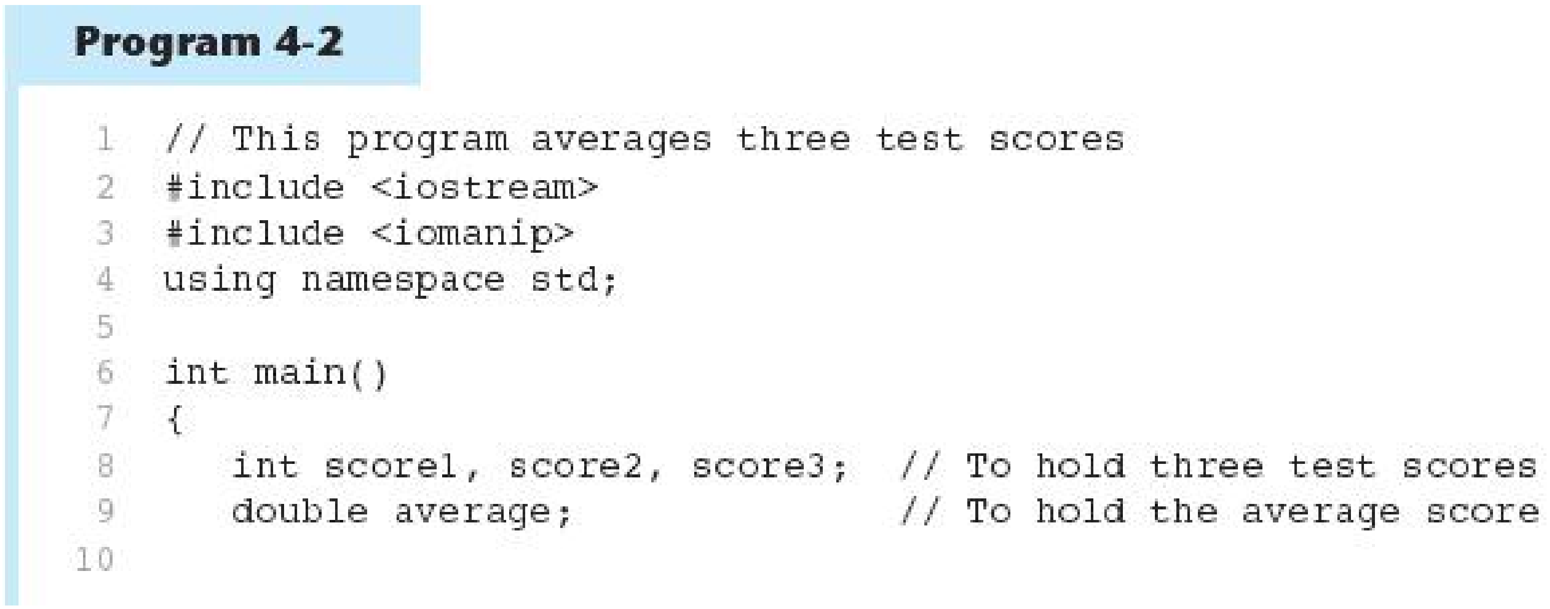
`if` statement – what happens

To evaluate:

```
if (expression)
statement;
```

- If the *`expression`* is `true`, then *`statement`* is executed.
- If the *`expression`*is `false`, then *`statement`* is skipped.

`if` statement – example

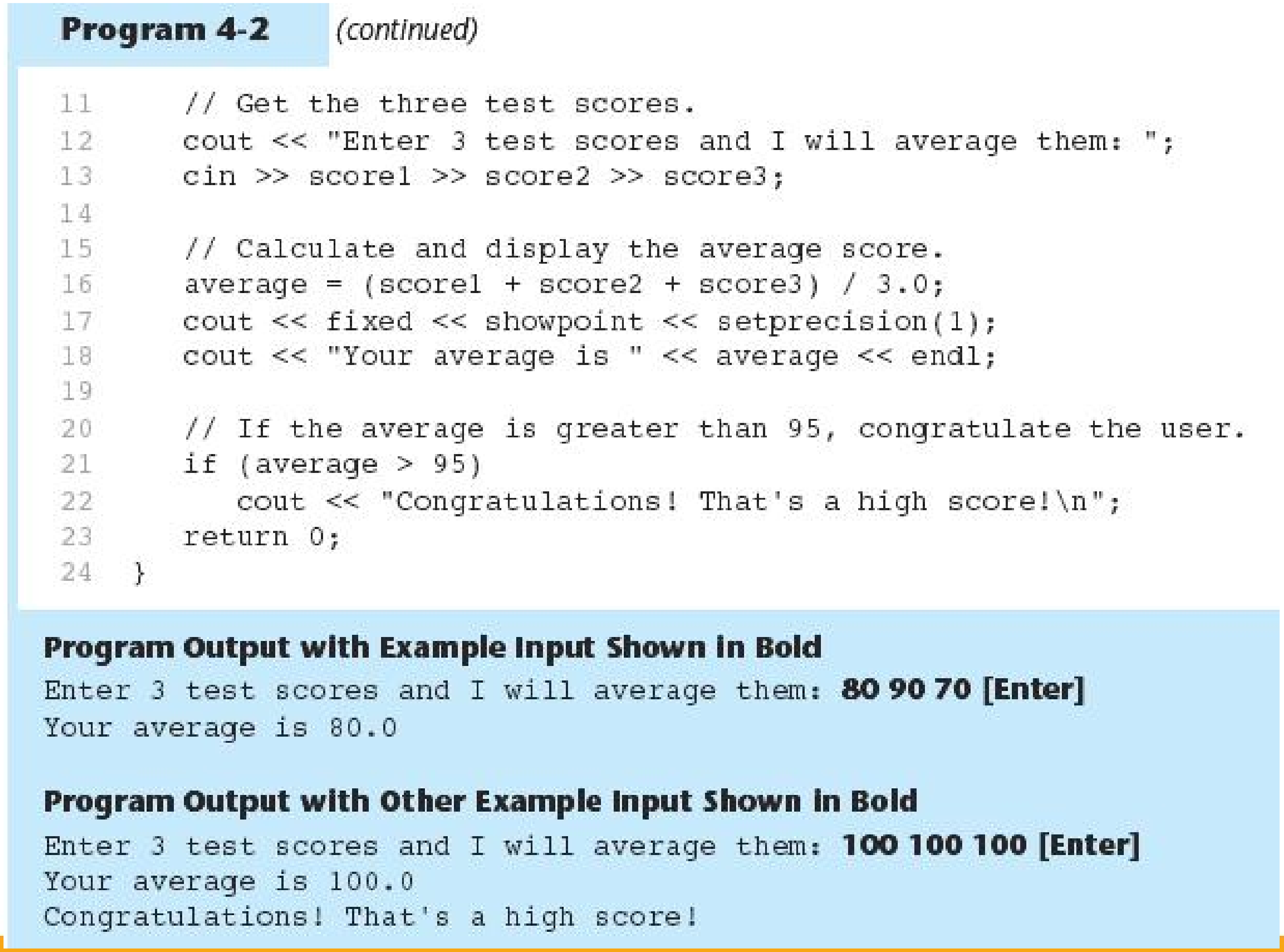


Program 4-2

```
// This program averages three test scores
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   int score1, score2, score3;   // To hold three test scores
   double average;               // To hold the average score

```

(Program Continues)

`if` statement – example



Program 4-2 *(continued)*

```
    // Get the three test scores.
    cout << "Enter 3 test scores and I will average them: ";
    cin >> score1 >> score2 >> score3;

    // Calculate and display the average score.
    average = (score1 + score2 + score3) / 3.0;
    cout << fixed << showpoint << setprecision(1);
    cout << "Your average is " << average << endl;

    // If the average is greater than 95, congratulate the user.
    if (average > 95)
       cout << "Congratulations! That's a high score!\n";
    return 0;
}
```

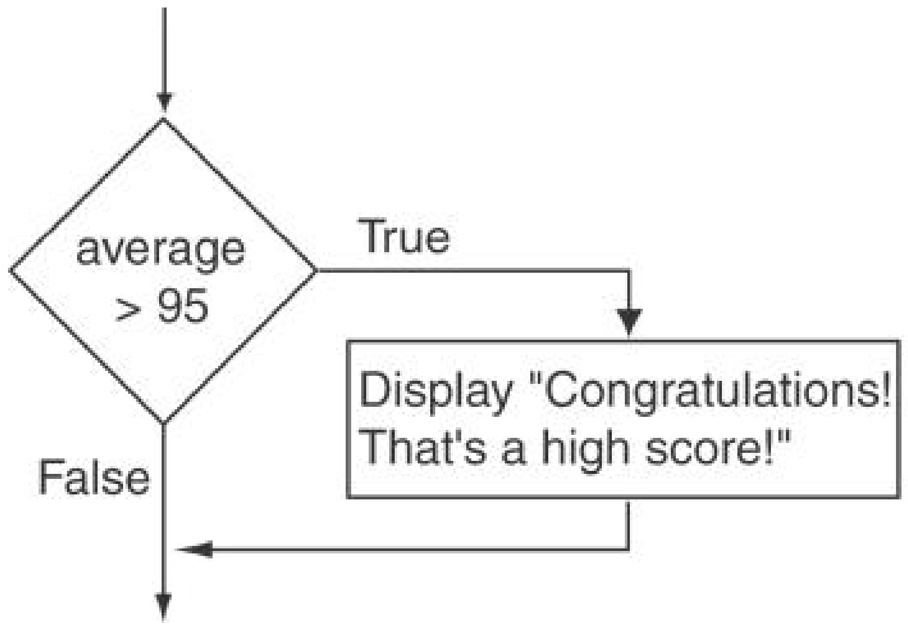
Program Output with Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 80 90 70 [Enter]
Your average is 80.0
```

Program Output with Other Example Input Shown in Bold

```
Enter 3 test scores and I will average them: 100 100 100 [Enter]
Your average is 100.0
Congratulations! That's a high score!
```

Flowchart for Lines 21 and 22



`if` statement notes

- Do not place `;` after `(`*`expression`*`)`
- Place *`statement;`* on a separate line after `(`*`expression`*`)`, indented:

```
if (score > 90)
     grade = 'A';
```

- Be careful testing `float`s and `double`s for equality
- `0` is `false`; any other value is `true`

Exercise Week 7_2

- Refer to Lab 7, Exe 1 No. 13(i) in pg. 88.
- Explain and draw the flowchart

Flags

Flags

- Variable that signals a condition
- Usually implemented as a `bool` variable
- As with other variables in functions, must be assigned an initial value before it is used

Exercise Week 7_3

- Trace the following programs if the input is 22 and 68

```
int main()
{
   double mark;
   bool pass=false; //this conditon does not yet exist

   cout<<"Enter your mark>>";
   cin>>mark;

   if (mark >=30)
        pass=true;
   if (pass)
        cout<<"You pass the test"<<endl;
   if (!pass)
        cout<<"You fail the test"<<endl;
   cout<<"Program end";

   return 0;
}
```

Expanding the `if` Statement

Expanding the `if` Statement

- To execute more than one statement as part of an `if` statement, enclose them in `{ }`:

```
 if (score > 90)
   {
         grade = 'A';
         cout << "Good Job!\n";
   }
```

- `{ }` creates a **block** of code

Exercise Week 7_4 (Solve the problem)

- Identify the logic errors, and correct them
  - The program will display “Pass” message, calculate the carry mark and display the carry mark if the student pass the test.
  - If the student fail the test the program will display “Fail” message and display a message to instruct the student to re-sit the test

```
int main()
{
   double mark, final20p = 0;

   cout << “Enter your mark >> “;
   cin >> mark;

   if (mark >= 30)
      cout << “TEST 1 -> Pass” << endl;
      final20p = ((20.0/100.0) * mark);
         cout << “Contribution to final mark “ << final20p;
   if (mark < 30)
      cout << “TEST 1 -> Fail” << endl;
      cout << “Please re-sit TEST 1” << endl;

   system (“pause”);
   return 0;
}
```

The `if/else` Statement

The `if/else` Statement

- Provides two possible paths of execution
- Performs one statement or block if the *`expression`* is true, otherwise performs another statement or block.

The `if/else` Statement

- General Format:

```
if (expression)
statement1;   // or block
  else
statement2;   // or block
```

`if/else` – what happens

To evaluate:

```
if (expression)
statement1;
  else
statement2;
```

- If the *`expression`* is `true`, then *`statement1`* is executed and *`statement2`* is skipped.
- If the *`expression`* is *`false`*, then *`statement1`* is skipped and *`statement2`* is executed.

`if/else` – example

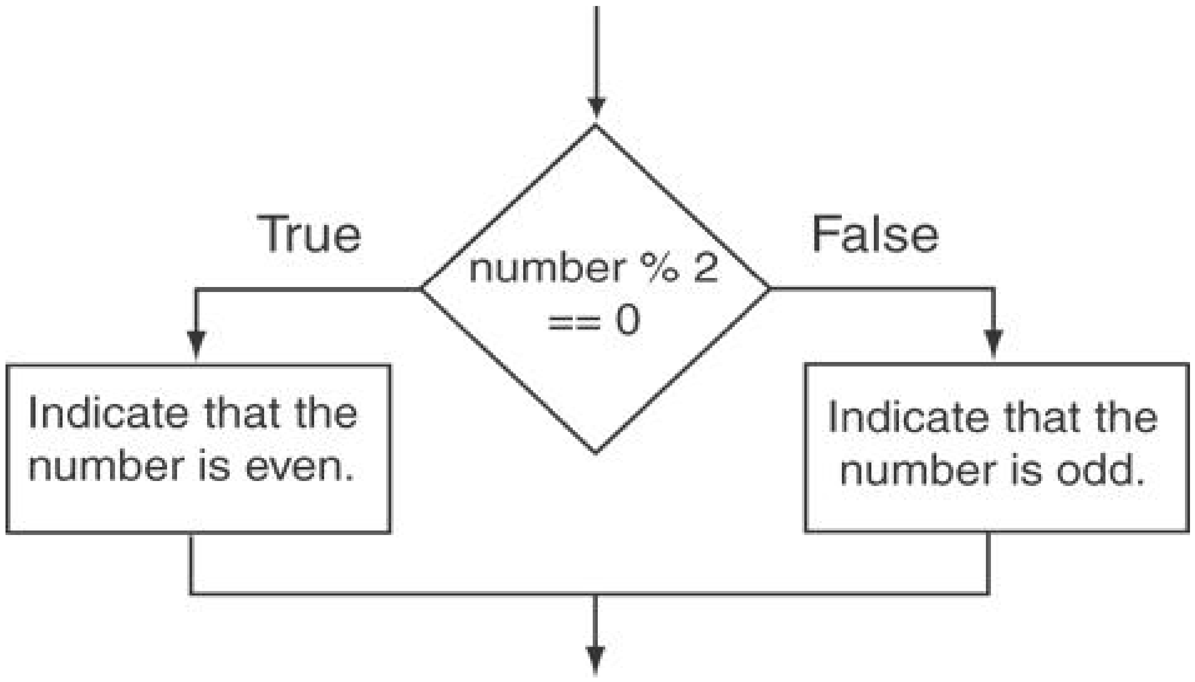
Program 4-8

```
// This program uses the modulus operator to determine
// if a number is odd or even. If the number is evenly divisible
// by 2, it is an even number. A remainder indicates it is odd.
#include <iostream>
using namespace std;

int main()
{
   int number;

   cout << "Enter an integer and I will tell you if it\n";
   cout << "is odd or even. ";
   cin >> number;
   if (number % 2 == 0)
      cout << number << " is even.\n";
   else
      cout << number << " is odd.\n";
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter an integer and I will tell you if it
is odd or even. 17 [Enter]
17 is odd.
```

Flowchart for Lines 14 through 18


True
number % 2 == 0
False
Indicate that the number is even.
Indicate that the number is odd.

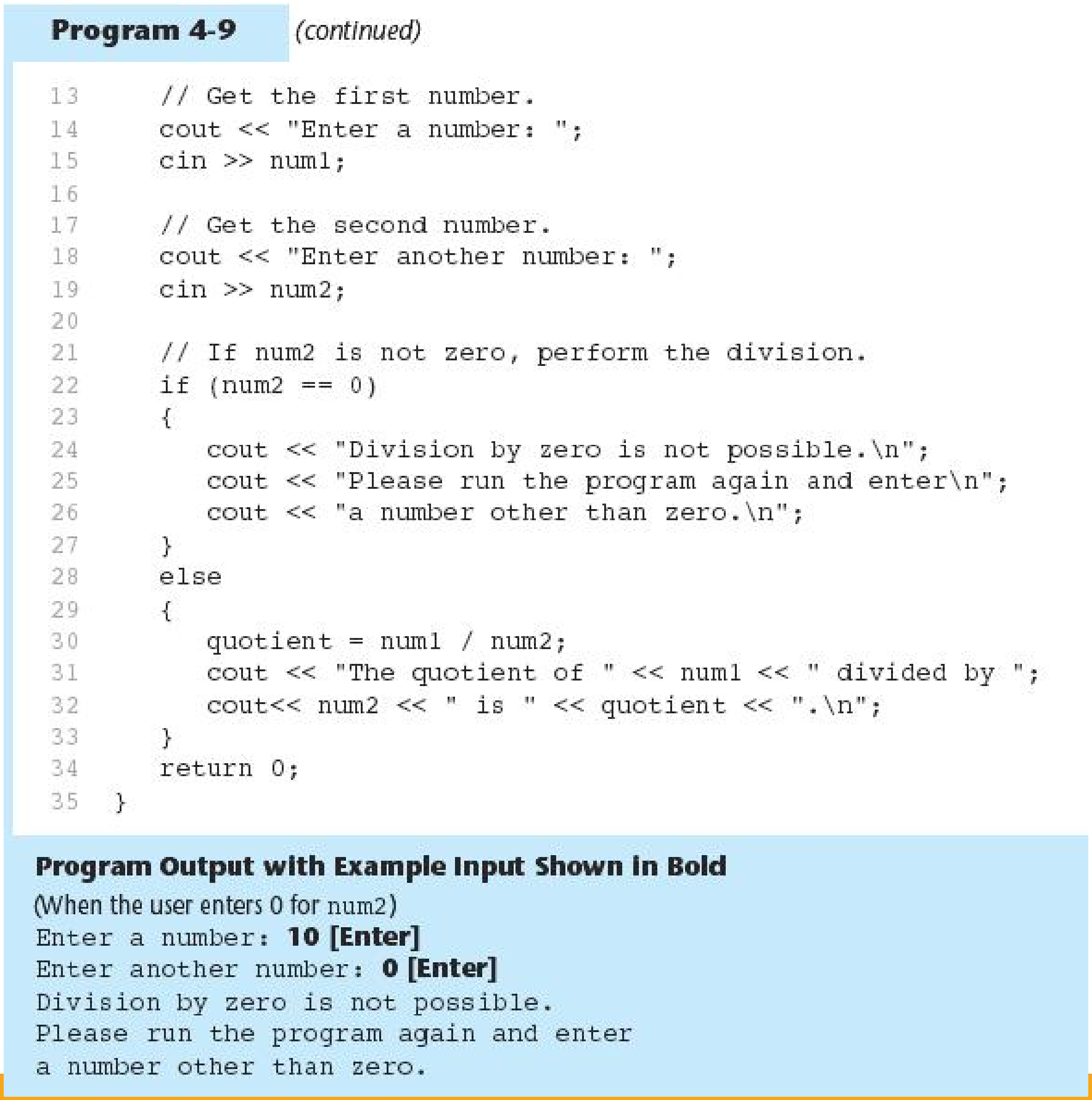
`if/else` – example

Program 4-9

```cpp
// This program asks the user for two numbers, num1 and num2.
// num1 is divided by num2 and the result is displayed.
// Before the division operation, however, num2 is tested
// for the value 0. If it contains 0, the division does not
// take place.
#include <iostream>
using namespace std;

int main()
{
   double num1, num2, quotient;

```

(Program Continues)

if/else – example



Program 4-9 *(continued)*

```
    // Get the first number.
    cout << "Enter a number: ";
    cin >> num1;

    // Get the second number.
    cout << "Enter another number: ";
    cin >> num2;

    // If num2 is not zero, perform the division.
    if (num2 == 0)
    {
       cout << "Division by zero is not possible.\n";
       cout << "Please run the program again and enter\n";
       cout << "a number other than zero.\n";
    }
    else
    {
       quotient = num1 / num2;
       cout << "The quotient of " << num1 << " divided by ";
       cout<< num2 << " is " << quotient << ".\n";
    }
    return 0;
 }
```

Program Output with Example Input Shown in Bold

(When the user enters 0 for num2)

```
Enter a number: 10 [Enter]
Enter another number: 0 [Enter]
Division by zero is not possible.
Please run the program again and enter
a number other than zero.
```

Exercise Week 7_5

- Refer back to Lab 7, Exe 2, No. 13, pg. 88.
- Solve the problem in (ii) and (iii)

The `if/else if` Statement

The `if/else if` Statement

- Chain of `if` statements that test in order until one is found to be true
- Also models thought processes:
  – "If it is raining, take an umbrella,
    else, if it is windy, take a hat,
    else, take sunglasses"

`if/else if` format

```
if (expression)
statement1;   // or block
else if (expression)
statement2;   // or block
     .
     . // other else ifs
     .
else if (expression)
statementn;   // or block
```

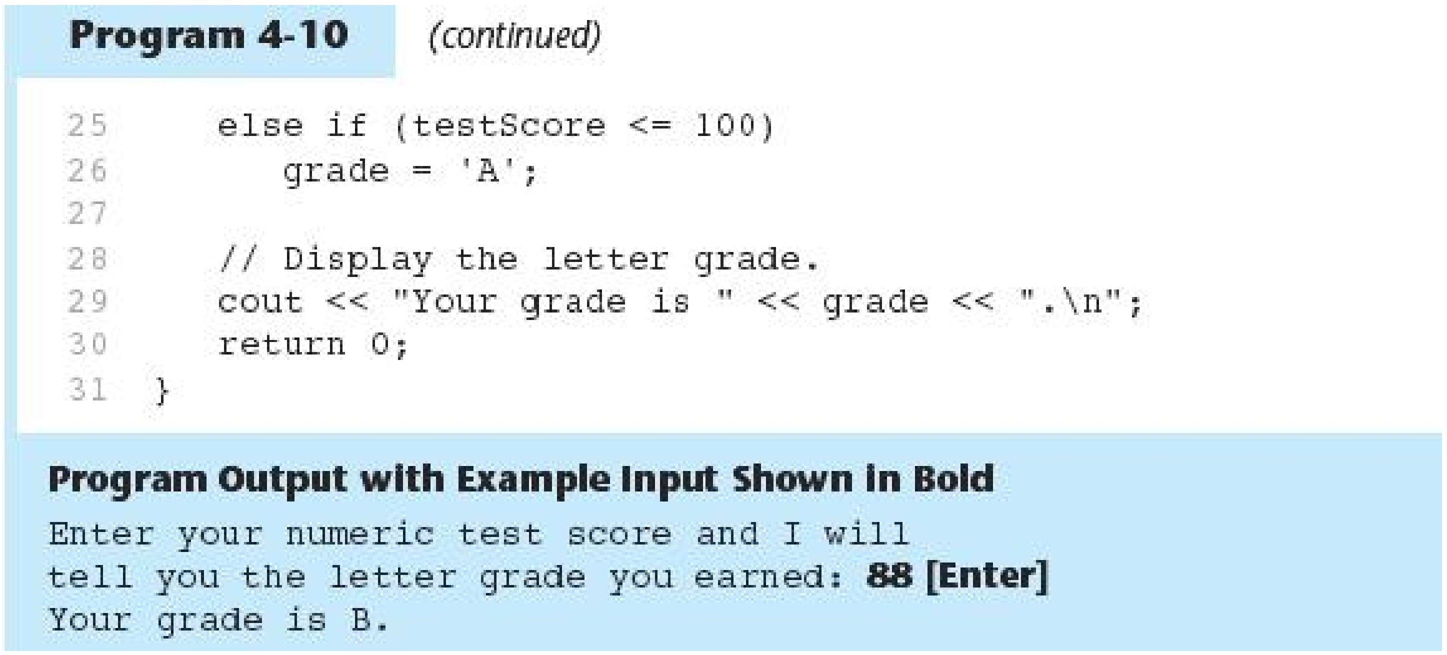
`if/else if` format - example

Program 4-10

```
// This program uses an if/else if statement to assign a
// letter grade (A, B, C, D, or F) to a numeric test score.
#include <iostream>
using namespace std;

int main()
{
   int testScore;   // To hold a numeric test score
   char grade;      // To hold a letter grade

   // Get the numeric test score.
   cout << "Enter your numeric test score and I will\n";
   cout << "tell you the letter grade you earned: ";
   cin >> testScore;

   // Determine the letter grade.
   if (testScore < 60)
      grade = 'F';
   else if (testScore < 70)
      grade = 'D';
   else if (testScore < 80)
      grade = 'C';
   else if (testScore < 90)
      grade = 'B';
```

(Program Continues)

`if/else if` format - example



Program 4-10 *(continued)*

```
    else if (testScore <= 100)
       grade = 'A';

    // Display the letter grade.
    cout << "Your grade is " << grade << ".\n";
    return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter your numeric test score and I will
tell you the letter grade you earned: 88 [Enter]
Your grade is B.
```

Exercise Week 7_6

- Refer to Lab 7, Exe 1, No. 11 in pg. 86.
- Draw a flowchart for Program 7.1
- Refer to Lab 7, Exe 1, No. 17 in pg. 91.
- Draw a flowchart for Program 7.4

- Discuss the differences.

Using a Trailing `else`

Using a Trailing `else`

- Used with `if/else if` statement when none of the *`expressions`* are true
- Provides default statement/action
- Used to catch invalid values, other exceptional situations

From Program 4-12

```
if (testScore < 60)
   cout << "Your grade is F.\n";
else if (testScore < 70)
   cout << "Your grade is D.\n";
else if (testScore < 80)
   cout << "Your grade is C.\n";
else if (testScore < 90)
   cout << "Your grade is B.\n";
else if (testScore <= 100)
   cout << "Your grade is A.\n";
else
   cout << "We do not give scores higher than 100.\n";
```

Menus

Menus

- Menu-driven program: program execution controlled by user selecting from a list of actions
- Menu: list of choices on the screen
- Menus can be implemented using `if/else if` statements

Menu-driven program organization

- Display list of numbered or lettered choices for actions
- Prompt user to make selection
- Test user selection in *`expression`*
  - if a match, then execute code for action
  - if not, then go on to next *`expression`*

Exercise Week 7_7

Jump to 'switch' – slide 81

- Refer to Lab 7, Exe. 1, No. 19, Program 7.5. in pg. 95.
- Use if / else….if to select the menu

- Use trailing `else` to print "We don't have any"

Nested `if` Statements

Nested `if` Statements

- An `if` statement that is part of the `if` or `else` part of another `if` statement
- Can be used to evaluate more than one condition:

```
if (score < 100)
  {
      if (score > 90)
          grade = 'A';
  }
```

Notes on coding nested `if`s

- An `else` matches the nearest `if` that does not have an `else`:

```
if (score < 100)
        if (score > 90)
            grade = 'A';
        else ...// goes with second if,
                // not first one
```

- Proper indentation helps greatly

Exercise Week 7_8

- Write nested `if` statements that perform the following test:

  If `amount1` is greater than 10 and `amount2` is less than 100, display the greater of the two.

Logical Operators

Logical Operators

- Used to create relational expressions from other relational expressions
- Operators, meaning, and explanation:

&&	AND	New relational expression is true if both expressions are true
||	OR	New relational expression is true if either expression is true
!	NOT	Reverses the value of an expression – true expression becomes false, and false becomes true

Logical Operators - examples

```
int x = 12, y = 5, z = -4;
```

`(x > y) && (y > z)`	`true`
`(x > y) && (z > y)`	`false`
`(x <= z) || (y == z)`	`false`
`(x <= z) || (y != z)`	`true`
`!(x >= z)`	`false`

The && Operator in Program 4-16

```
// Determine the user's loan qualifications.
if (employed == 'Y' && recentGrad == 'Y')
{
   cout << "You qualify for the special ";
   cout << "interest rate.\n";
}
```

The || Operator in Program 4-17

```
// Determine the user's loan qualifications.
if (income >= 35000 || years > 5)
   cout << "You qualify.\n";
```

The ! Operator in Program 4-18

```
// Determine the user's loan qualifications.
if (!(income >= 35000 || years > 5))
{
   cout << "You must earn at least $35,000 or have\n";
   cout << "been employed for more than 5 years.\n";
```

Logical Operators - notes

- `!` has highest precedence, followed by `&&`, then `||`
- If the value of an expression can be determined by evaluating just the sub-expression on left side of a logical operator, then the sub-expression on the right side will not be evaluated (*short circuit evaluation*)

```
 !(x > 2)
   !x > 2
```

Checking Numeric Ranges with Logical Operators

Checking Numeric Ranges with Logical Operators

- Used to test to see if a value falls **inside** a range:

```
if (grade >= 0 && grade <= 100)
        cout << "Valid grade";
```

- Can also test to see if value falls **outside** of range:

```
if (grade <= 0 || grade >= 100)
        cout << "Invalid grade";
```

- Cannot use mathematical notation:

```
   if (0 <= grade <= 100) //doesn't work!
```

Validating User Input

Validating User Input

- Input validation: inspecting input data to determine whether it is acceptable
- Bad output will be produced from bad input
- Can perform various tests:
  - Range
  - Reasonableness
  - Valid menu choice
  - Divide by zero

From Program 4-19

```
// Get the numeric test score.
cout << "Enter your numeric test score and I will\n";
cout << "tell you the letter grade you earned: ";
cin >> testScore;

if (testScore < 0 || testScore > 100) //Input validation
{
   // An invalid score was entered.
   cout << testScore << " is an invalid score.\n";
   cout << "Run the program again and enter a value\n";
   cout << "in the range of 0 to 100.\n";
}
else
{
   // Determine the letter grade.
   if (testScore < 60)
      grade = 'F';
   else if (testScore < 70)
      grade = 'D';
   else if (testScore < 80)
      grade = 'C';
   else if (testScore < 90)
      grade = 'B';
   else if (testScore <= 100)
      grade = 'A';

   // Display the letter grade.
   cout << "Your grade is " << grade << endl;
}
```

More About Variable Definitions and Scope

More About Variable Definitions and Scope

- Scope of a variable is the block in which it is defined, from the point of definition to the end of the block
- Usually defined at beginning of function
- May be defined close to first use

From Program 4-21

```
int main()
{
   // Get the annual income.
   cout << "What is your annual income? ";
   double income;    //variable definition
   cin >> income;

   if (income >= 35000)
   {
      // Get the number of years at the current job.
      cout << "How many years have you worked at "
           << "your current job? ";
      int years;      //variable definition
      cin >> years;

      if (years > 5)
         cout << "You qualify.\n";
      else
      {
         cout << "You must have been employed for\n";
         cout << "more than 5 years to qualify.\n";
      }
   }
```

Still More About Variable Definitions and Scope

- Variables defined inside { } have local or block scope
- When inside a block within another block, can define variables with the same name as in the outer block.
  - When in inner block, outer definition is not available
  - Not a good idea

Exercise Week 7_9

- What will the following program display if user enter `test1` 40 and `test2` 30?

```
int main ()
{
 int test1;
 cout<<"Enter Test 1 score: ";
 cin>>test1;

 int test2;
 cout<<"Enter Test 2 score: ";
 cin>>test2;

 int sum=test1+test2;
```

```
if (sum>60)   {
 int bonus=10;
 test1+=bonus; test2+=bonus;
 int sum=test1+test2;
 cout<<"Test 1 with bonus:"
<<test1<<endl;
 cout<<"Test 2 with bonus:"
<<test2<<endl;
 cout<<"Sum with bonus:"
<<sum<<endl;
}
cout<<"Test 1 : "
<<test1<<endl;
cout<<"Test 2 : "
<<test2<<endl;
cout<<"Sum : "<<sum<<endl;
return 0;
}
```

Comparing Strings

Comparing Strings

- You cannot use relational operators with C-strings
- Must use the `strcmp` function to compare C-strings
- `strcmp` compares the ASCII codes of the characters in the C-strings. Comparison is character-by-character

Comparing Strings

The expression

`strcmp(str1, str2)`

compares thestrings `str1` and `str2`

- It returns `0` if the strings are the same
- It returns a negative number if `str1 < str2`
- It returns a positive number if `str1 > str2`

Comparing Strings - example

Program 4-24

```
// This program correctly tests two C-strings for equality
// with the strcmp function.
#include <iostream>
#include <cstring>
using namespace std;

int main()
{
   const int SIZE = 40;
   char firstString[SIZE], secondString[SIZE];

   // Get two strings
   cout << "Enter a string: ";
   cin.getline(firstString, SIZE);
   cout << "Enter another string: ";
   cin.getline(secondString, SIZE);

   // Compare them with strcmp.
   if (strcmp(firstString, secondString) == 0)
      cout << "You entered the same string twice.\n";
   else
      cout << "The strings are not the same.\n";
   return 0;
}
```

Comparing Strings - example

Program Output with Example Input Shown in Bold

```
Enter a string: **Alfonso [Enter]**
Enter another string: **Alfonso [Enter]**
You entered the same string twice.
```

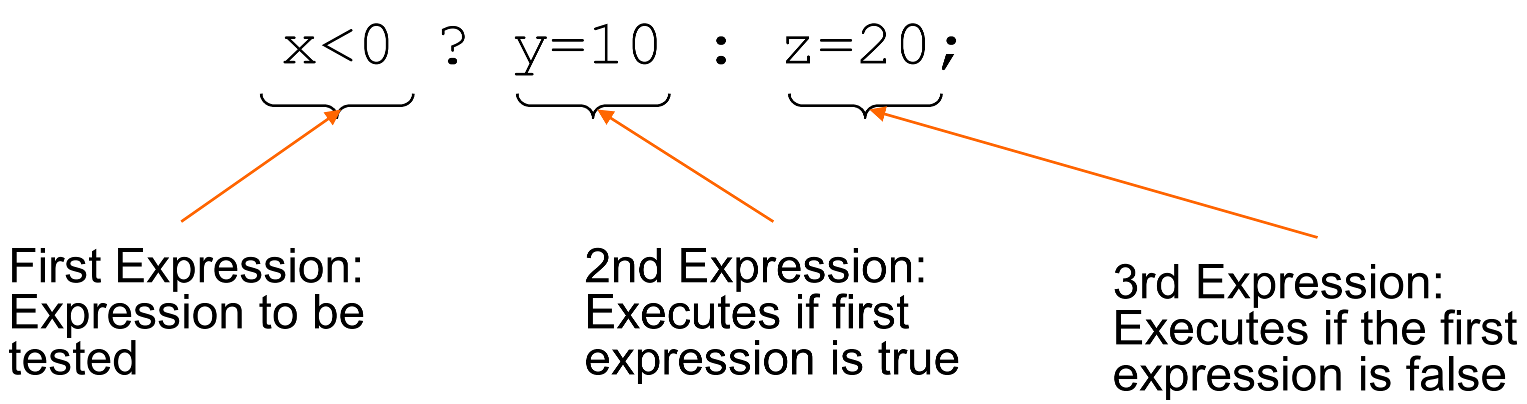
Exercise Week 7_10

- Refer back to Lab 7, Exe. 19, Program 7.5 in pg. 95.
- Change the program that you wrote in Exercise Week 7_7 :
  - – Change variable `choice` to variable `iceCream[20]`
  - – Instead of using menu, use `getline` so the user can enter the flavor and use `strcmp` to `if / else…if` statement.

The Conditional Operator

The Conditional Operator

- Can use to create short `if/else` statements
- Format: `expr ? expr : expr;`



```
x<0 ? y=10 : z=20;
```

First Expression: Expression to be tested

2nd Expression: Executes if first expression is true

3rd Expression: Executes if the first expression is false

The Conditional Operator

- The value of a conditional expression is
  - The value of the second expression if the first expression is true
  - The value of the third expression if the first expression is false
- Parentheses （） may be needed in an expression due to precedence of conditional operator

The Conditional Operator

- Condition operator vs `if/else` statements

```
if (x<0)
  y=10;
else
  z=20;
```

```
(x<0)?(y=10):(z=20);
```

```
if (x>100)
  a=0;
else
  a=1;
```

```
a=x>100?0:1;
```

```
if (score<60)
 cout<<"Your grade is FAIL";
else
 cout<<"Your grade is PASS";
```

```
cout<<"Your grade is "
<< ((score<60)? "FAIL":
"Pass");
```

The Conditional Operator - example

Program 4-27

```
// This program calculates a consultant's charges at $50
// per hour, for a minimum of 5 hours. The ?: operator
// adjusts hours to 5 if less than 5 hours were worked.
#include <iostream>
#include <iomanip>
using namespace std;

int main()
{
   const double PAY_RATE = 50.0;
   double hours, charges;

   cout << "How many hours were worked? ";
   cin >> hours;
   hours = hours < 5 ? 5 : hours; //conditional operator
   charges = PAY_RATE * hours;
   cout << fixed << showpoint << setprecision(2);
   cout << "The charges are $" << charges << endl;
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
How many hours were worked? 10 [Enter]
The charges are $500.00
```

Program Output with Example Input Shown in Bold

```
How many hours were worked? 2 [Enter]
The charges are $250.00
```

Exercise Week 7_11

- Rewrite the following if/else statements as conditional expressions

```
if (x>y)
  z = 1;
else
  z = 20;

if (hours> 40)
  wages *= 1.5;
else
  wages *= 1;

if (result >= 0)
 cout «"The result is +ve";
else
 cout «"The result is -ve";
```

- Rewrite the following conditional expressions as if/else statements

```
j = k > 90 ? 57 : 12;

total += count == 1 ? sales :
count * sales;

cout « (((num % 2) == 0) ?
"Even\n" : "Odd\n");
```

The `switch` Statement

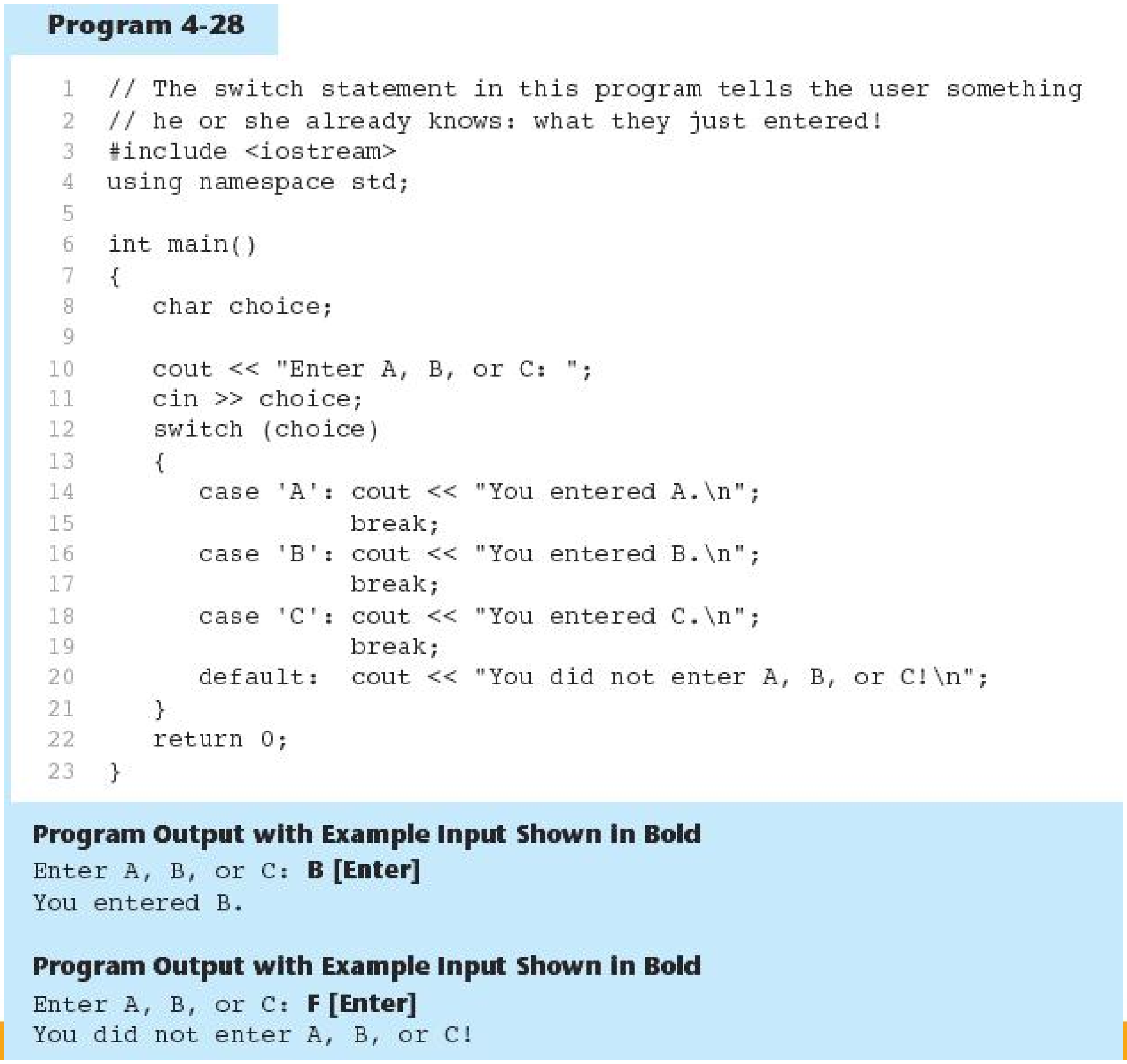
The `switch` Statement

- Used to select among statements from several alternatives
- In some cases, can be used instead of `if/else if` statements

`switch` statement format

```
switch (expression) //integer
{
case exp1: statement1;
  case exp2: statement2;
  ...
  case expn: statementn;
  default:   statementn+1;
}
```

`switch` statement - example



Program 4-28

```
// The switch statement in this program tells the user something
// he or she already knows: what they just entered!
#include <iostream>
using namespace std;

int main()
{
   char choice;

   cout << "Enter A, B, or C: ";
   cin >> choice;
   switch (choice)
   {
      case 'A': cout << "You entered A.\n";
                break;
      case 'B': cout << "You entered B.\n";
                break;
      case 'C': cout << "You entered C.\n";
                break;
      default:  cout << "You did not enter A, B, or C!\n";
   }
   return 0;
}
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: B [Enter]
You entered B.
```

Program Output with Example Input Shown in Bold

```
Enter A, B, or C: F [Enter]
You did not enter A, B, or C!
```

`switch` statement requirements

1) *`expression`* must be an integer variable or an expression that evaluates to an integer value
2) *`exp`**`1`*** through *`exp`**`n`*** must be constant integer expressions or literals, and must be unique in the `switch` statement
3) `default` is optional but recommended

`switch` statement – how it works

1) *`expression`* is evaluated
2) The value of *`expression`* is compared against *`exp`**`1`*** through *`exp`**`n`***.
3) If *`expression`* matches value *`exp`**`i`***, the program branches to the statement following *`exp`**`i`*** and continues to the end of the `switch`
4) If no matching value is found, the program branches to the statement after `default:`

`break` statement

- Used to exit a `switch` statement
- If it is left out, the program "falls through" the remaining statements in the `switch` statement

`break` statement - example

Program 4-30

```cpp
// This program is carefully constructed to use the "fallthrough"
// feature of the switch statement.
#include <iostream>
using namespace std;

int main()
{
   int modelNum;  // Model number

   // Get a model number from the user.
   cout << "Our TVs come in three models:\n";
   cout << "The 100, 200, and 300. Which do you want? ";
   cin >> modelNum;

   // Display the model's features.
   cout << "That model has the following features:\n";
   switch (modelNum)
   {
      case 300: cout << "\tPicture-in-a-picture.\n";
      case 200: cout << "\tStereo sound.\n";
      case 100: cout << "\tRemote control.\n";
                break;
      default:  cout << "You can only choose the 100,";
                cout << "200, or 300.\n";
   }
   return 0;
}
```

break statement - example

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 100 [Enter]
That model has the following features:
    Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 200 [Enter]
That model has the following features:
    Stereo sound.
    Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 300 [Enter]
That model has the following features:
    Picture-in-a-picture.
    Stereo sound.
    Remote control.
```

Program Output with Example Input Shown in Bold

```
Our TVs come in three models:
The 100, 200, and 300. Which do you want? 500 [Enter]
That model has the following features:
You can only choose the 100, 200, or 300.
```

Using `switch` with a menu

- `switch` statement is a natural choice for menu-driven program:
  - display the menu
  - then, get the user's menu selection
  - use user input as `expression` in `switch` statement
  - use menu choices as *`expr`* in `case` statements

From Program 4-32

```
    // Display the menu and get a choice.
    cout << "\t\tHealth Club Membership Menu\n\n";
    cout << "1. Standard Adult Membership\n";
    cout << "2. Child Membership\n";
    cout << "3. Senior Citizen Membership\n";
    cout << "4. Quit the Program\n\n";
    cout << "Enter your choice: ";
    cin >> choice;
```

```
       // Respond to the user's menu selection.
       switch (choice)
       {
          case 1:  charges = months * ADULT;
                   break;
          case 2:  charges = months * CHILD;
                   break;
          case 3:  charges = months * SENIOR;
       }
```

Back to slide 47

Exercise Week 7_12

- Change Program 7.5 in pg. 63 to :
  - Input 1=> Output :Muhammad’s favorite Ismael’s favorite
  - Input 3=> Output :Adibah’s favorite Munirah’s favorite
  - Input 2 => Output :Ismael’s favorite
  - Input 4 => Output :Munirah’s favorite

Testing for File Open Errors

Testing for File Open Errors

- Can test a file stream object to detect if an open operation failed:

```
infile.open("test.txt");
  if (!infile)
  {
    cout << "File open failure!";
  }
```

- Can also use the `fail` member function

Exercise Week 7_13

- Refer to Program 6.4 in pg. 74.
- Modify the program to detect if the open files operation failed.

Thank You

Q & A