1

2 **4.1**

Relational Operators

3 **Relational Operators**

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

4 **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

5 **Relational Expressions**

- Can be assigned to a variable:

  result = x <= y;
- Assigns 0 for false, 1 for true
- Do not confuse = and ==
-

6 **4.2**

The if Statement

7 **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."
-

8 **Flowchart for Evaluating a Decision**

9 **Flowchart for Evaluating a Decision**

10 **The if Statement**

- General Format:

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

11 **The 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.

12 **if Statement in Program 4-2**

13 **if Statement in Program 4-2**

14 **Flowchart for Program 4-2 Lines 21 and 22**

15 **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 floats and doubles for equality
- 0 is false; any other value is true

16 **4.3**

Expanding the if Statement

17 **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

18 **4.4**

The if/else Statement

19 **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.

20 **The if/else statement**

- General Format:

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

21 **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.

22 **The if/else statement and Modulus Operator in Program 4-8**

23 **Flowchart for Program 4-8 Lines 14 through 18**

24 **Testing the Divisor in Program 4-9**

25 **Testing the Divisor in Program 4-9**

26 **4.5**

Nested if Statements

27 **Nested if Statements**

- An if statement that is nested inside another if statement
- Nested if statements can be used to test more than one condition

28 **Flowchart for a Nested if Statement**

29 **Nested if Statements**

- From Program 4-10

30 **Nested if Statements**

- Another example, from Program 4-1

31 **Use Proper Indentation!**

32 **4.6**

The if/else if Statement

33 **The if/else if Statement**

- Tests a series of conditions until one is found to be true
- Often simpler than using nested if/else statements
- Can be used to model thought processes such as:

"If it is raining, take an umbrella,
else, if it is windy, take a hat,
else, take sunglasses"

34 **if/else if Format**

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

35 **The if/else if Statement in Program 4-13**

36 **Using a Trailing else to Catch Errors in Program 4-14**

- The trailing else clause is optional, but it is best used to catch errors.

37 **4.7**

Flags

38 **Flags**

- Variable that signals a condition
- Usually implemented as a bool variable
- Can also be an integer
  - The value 0 is considered false
  - Any nonzero value is considered true
- As with other variables in functions, must be assigned an initial value before it is used

39 **4.8**

Logical Operators

40 **Logical Operators**

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

41 **Logical Operators-Examples**

42 **The logical && operator in Program 4-15**

43 **The logical || Operator in Program 4-16**

44 **The logical ! Operator in Program 4-17**

45 **Logical Operator-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*)
-

46 **4.9**

Checking Numeric Ranges with Logical Operators

47 **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!
```

-

48 **4.10**

Menus

49 **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

50 **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*

51 **4.11**

Validating User Input

52 **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

53 **Input Validation in Program 4-19**

54 **4.12**

Comparing Characters and Strings

55 **Comparing Characters**

- Characters are compared using their ASCII values
- 'A' < 'B'
  - The ASCII value of 'A' (65) is less than the ASCII value of 'B'(66)
- '1' < '2'
  - The ASCII value of '1' (49) is less than the ASCI value of '2' (50)
- Lowercase letters have higher ASCII codes than uppercase letters, so 'a' > 'Z'

56 **Relational Operators Compare Characters in Program 4-20**

57 **Comparing string Objects**

- Like characters, strings are compared using their ASCII values

58 **Relational Operators Compare Strings in Program 4-21**

59 **4.13**

The Conditional Operator

60 **The Conditional Operator**

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

61 **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

62 **The Conditional Operator in Program 4-22**

63 **4.14**

The switch Statement

64 **The switch Statement**

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

65 **switch Statement Format**

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

66 **The switch Statement in Program 4-23**

67 **switch Statement Requirements**

1) *expression* must be an integer variable or an expression that evaluates to an integer value
2) *exp1* through *expn* must be constant integer expressions or literals, and must be unique in the switch statement
3) default is optional but recommended

68 **switch Statement-How it Works**

1) *expression* is evaluated
2) The value of *expression* is compared against *exp1* through *expn*.
3) If *expression* matches value *expi*, the program branches to the statement following *expi* and continues to the end of the switch
4) If no matching value is found, the program branches to the statement after default:

69 **break Statement**

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

70 **break and default statements in Program 4-25**

71 **break and default statements in Program 4-25**

72 **Using switch in Menu Systems**

- 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