

# Day 3

---



# 4.9

## 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!
```

# 4.10

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*
- ❖ 4.18

# 4.11

## 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
- ❖ **BagelValidation.cpp**



# Input Validation in Program 4-19



```
16  int testScore; // To hold a numeric test score
17
18  // Get the numeric test score.
19  cout << "Enter your numeric test score and I will\n"
20       << "tell you the letter grade you earned: ";
21  cin >> testScore;
22
23  // Validate the input and determine the grade.
24  if (testScore >= MIN_SCORE && testScore <= MAX_SCORE)
25  {
26      // Determine the letter grade.
27      if (testScore >= A_SCORE)
28          cout << "Your grade is A.\n";
29      else if (testScore >= B_SCORE)
30          cout << "Your grade is B.\n";
31      else if (testScore >= C_SCORE)
32          cout << "Your grade is C.\n";
33      else if (testScore >= D_SCORE)
34          cout << "Your grade is D.\n";
35      else
36          cout << "Your grade is F.\n";
37  }
38  else
39  {
40      // An invalid score was entered.
41      cout << "That is an invalid score. Run the program\n"
42           << "again and enter a value in the range of\n"
43           << MIN_SCORE << " through " << MAX_SCORE << ".\n";
44  }
```

# 4.12

## Comparing Characters and Strings

# 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 ASCII value of '2' (50)
- ❖ Lowercase letters have higher ASCII codes than uppercase letters, so 'a' > 'Z'

# Relational Operators Compare Characters in Program 4-20



```
10    // Get a character from the user.
11    cout << "Enter a digit or a letter: ";
12    ch = cin.get();
13
14    // Determine what the user entered.
15    if (ch >= '0' && ch <= '9')
16        cout << "You entered a digit.\n";
17    else if (ch >= 'A' && ch <= 'Z')
18        cout << "You entered an uppercase letter.\n";
19    else if (ch >= 'a' && ch <= 'z')
20        cout << "You entered a lowercase letter.\n";
21    else
22        cout << "That is not a digit or a letter.\n";
```

# Comparing string Objects



- ❖ Like characters, strings are compared using their ASCII values

```
string name1 = "Mary";  
string name2 = "Mark";
```

The characters in each string must match before they are equal

```
name1 > name2 // true  
name1 <= name2 // false  
name1 != name2 // true
```

```
name1 < "Mary Jane" // true
```

# Relational Operators Compare Strings in Program 4-21

---



```
26      // Determine and display the correct price
27      if (partNum == "S-29A")
28          cout << "The price is $" << PRICE_A << endl;
29      else if (partNum == "S-29B")
30          cout << "The price is $" << PRICE_B << endl;
31      else
32          cout << partNum << " is not a valid part number.\n";
```

# 4.13

## The Conditional Operator

# The Conditional Operator (1 of 3)

---



- ❖ The *conditional operator* is a ternary (three operand) operator.
- ❖ You can use the conditional operator to write a simple statement that works like an if-else statement.



# The Conditional Operator (2 of 3)

---



- ❖ The format of the operators is:
  - ❖ *BooleanExpression ? Value1 : Value2*
- ❖ This forms a conditional expression.
- ❖ If *BooleanExpression* is true, the value of the conditional expression is *Value1*.
- ❖ If *BooleanExpression* is false, the value of the conditional expression is *Value2*.

# The Conditional Operator (3 of 3)

---



❖ Example:

❖  **$z = x > y ? 10 : 5;$**

❖ This line is functionally equivalent to:

❖ **if( $x > y$ )**

❖  **$z = 10;$**

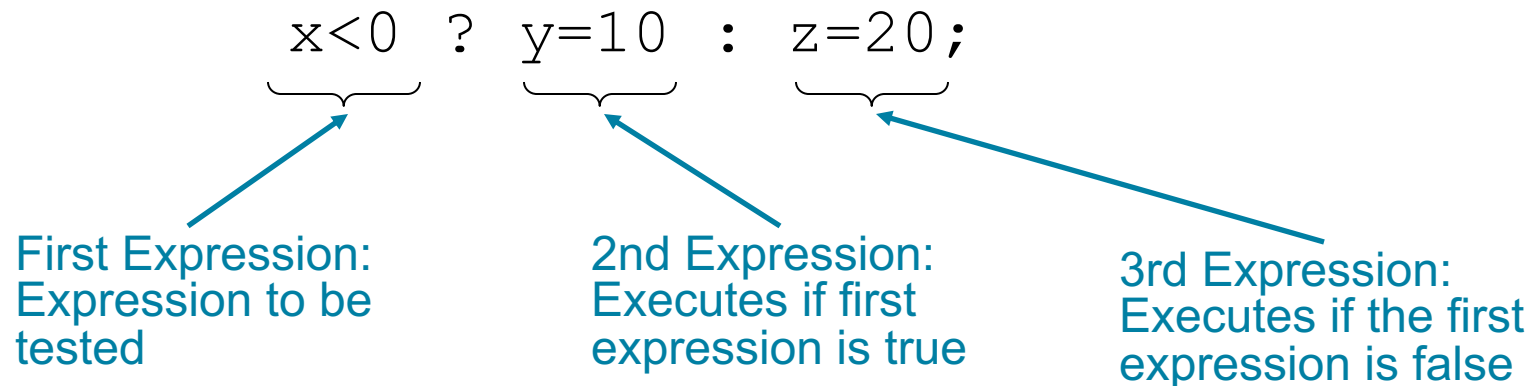
❖ **else**

❖  **$z = 5;$**

# The Conditional Operator



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



# Conditional Operators



```
if (num % 2 == 0)
    cout << num << "is even";
else
    cout << num << "is odd";
```

```
(num % 2 == 0)? cout << num << " is even" : cout << num <<
" is odd";
```

# The Conditional Operator in Program 4-22



```
1 // This program calculates a consultant's charges at $50
2 // per hour, for a minimum of 5 hours. The ?: operator
3 // adjusts hours to 5 if less than 5 hours were worked.
4 #include <iostream>
5 #include <iomanip>
6 using namespace std;
7
8 int main()
9 {
10     const double PAY_RATE = 50.0; // Hourly pay rate
11     const int MIN_HOURS = 5;      // Minimum billable hours
12     double hours,                 // Hours worked
13           charges;                // Total charges
14
15     // Get the hours worked.
16     cout << "How many hours were worked? ";
17     cin >> hours;
18
19     // Determine the hours to charge for.
20     hours = hours < MIN_HOURS ? MIN_HOURS : hours;
21
22     // Calculate and display the charges.
23     charges = PAY_RATE * hours;
24     cout << fixed << showpoint << setprecision(2)
25           << "The charges are $" << charges << endl;
26     return 0;
27 }
```

# 4.14

## 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 Statements

## ❖ Syntax for switch statement

```
switch (switch-expression) {  
    case value1: statement(s) 1;  
        break;  
    case value2: statement(s) 2;  
        break;  
    ....  
    case valueN: statement(s) N;  
        break;  
    default: cout << "Errors: invalid status";  
}
```

See example: SwitchDemo.cpp w5



# switch Statement Rules

The switch-expression must yield a value of char, byte, short, or int type and must always be enclosed in parentheses.

The value1, ..., and valueN must have the same data type as the value of the switch-expression. The resulting statements in the case statement are executed when the value in the case statement matches the value of the switch-expression.

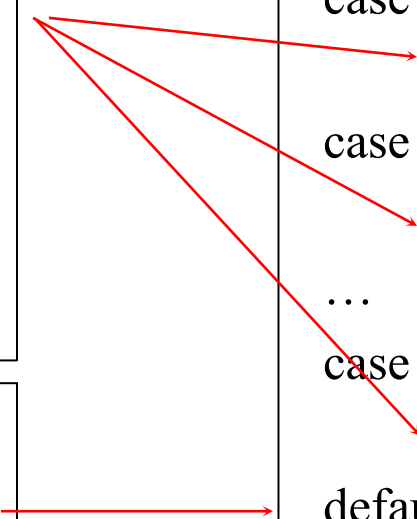
```
switch (switch-expression) {  
    case value1: statement(s)1;  
        break;  
    case value2: statement(s)2;  
        break;  
    ...  
    case valueN: statement(s)N;  
        break;  
    default: statement(s)-for-default;  
}
```

# switch Statement Rules

The keyword break is optional, but it should be used at the end of each case in order to terminate the remainder of the switch statement. **If the break statement is not present, the next case statement will be executed.**

The default case, which is optional, can be used to perform actions when none of the specified cases matches the switch-expression.

```
switch (switch-expression) {  
    case value1: statement(s)1;  
        break;  
    case value2: statement(s)2;  
        break;  
    ...  
    case valueN: statement(s)N;  
        break;  
    default: statement(s)-for-default;  
}
```

Three red arrows originate from the first text box. One points to the first 'break;' statement, another points to the second 'break;' statement, and a third points to the 'default:' line. A single red arrow originates from the second text box and points to the 'default:' line.

When the value in a **case** statement matches the value of the **switch-expression**, the statements *starting from this case* are executed until either a **break** statement or the end of the **switch** statement is reached.

# The switch Statement in Program 4-23



## Program 4-23

```
1 // The switch statement in this program tells the user something
2 // he or she already knows: the data just entered!
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     char choice;
9
10    cout << "Enter A, B, or C: ";
11    cin >> choice;
12    switch (choice)
13    {
14        case 'A': cout << "You entered A.\n";
15                  break;
16        case 'B': cout << "You entered B.\n";
17                  break;
18        case 'C': cout << "You entered C.\n";
19                  break;
20        default:  cout << "You did not enter A, B, or C!\n";
21    }
22    return 0;
23 }
```

### 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

---



- ❖ *Expression* must be an integer variable or an expression that evaluates to an integer value.
- ❖ *exp1* through *expn* must be constant integer expressions or literals, and must be unique in the switch statement
- ❖ default is optional but recommended

# The case Statement

---



- ❖ The break statement ends the case statement.
- ❖ The break statement is optional. If a case does not contain a break, then program execution continues into the next case.
  - ❖ See example: [NoBreaks.cpp w5](#)

# The **break** Statement

---



- ❖ Used to exit a switch statement
- ❖ Omitting the *break* statement for a case will cause the statements within the next case to be executed. Such "falling through" to the next case can be useful when multiple cases, such as cases 0, 1, and 2, should execute the same statements
  - ❖ See example: [PetFood.cpp w5](#)
- ❖ The default section is optional and will be executed if no *CaseExpression* matches the *SwitchExpression*.

# break and default statements in Program 4-25



## Program 4-25

```
1 // This program is carefully constructed to use the "fall through"
2 // feature of the switch statement.
3 #include <iostream>
4 using namespace std;
5
6 int main()
7 {
8     int modelNum; // Model number
9
10    // Get a model number from the user.
11    cout << "Our TVs come in three models:\n";
12    cout << "The 100, 200, and 300. Which do you want? ";
13    cin >> modelNum;
14
15    // Display the model's features.
16    cout << "That model has the following features:\n";
17    switch (modelNum)
18    {
19        case 300: cout << "\tPicture-in-a-picture.\n";
20        case 200: cout << "\tStereo sound.\n";
21        case 100: cout << "\tRemote control.\n";
22                break;
23        default: cout << "You can only choose the 100,";
24                cout << "200, or 300.\n";
25    }
26    return 0;
27 }
```

Continued...

# break and default statements in Program 4-25



## **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.



# Classroom Exercise

---



- ❖ Get a user number from 1 to 10 and output the corresponding Roman numeral using switch statements w5

# 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

- ❖ Do the bagel problem with Menu using switch statements W5

# 4.15

## More About Blocks and Scope

# More About Blocks 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

# Inner Block Variable Definition in Program

## 4-29



```
16     if (income >= MIN_INCOME)
17     {
18         // Get the number of years at the current job.
19         cout << "How many years have you worked at "
20             << "your current job? ";
21         int years;        // Variable definition
22         cin >> years;
23
24         if (years > MIN_YEARS)
25             cout << "You qualify.\n";
26         else
27         {
28             cout << "You must have been employed for\n"
29                 << "more than " << MIN_YEARS
30                 << " years to qualify.\n";
31         }
32     }
```

# Variables with the Same Name

---



- ❖ 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

# Two Variables with the Same Name in Program 4-30



## Program 4-30

```
1 // This program uses two variables with the name number.
2 #include <iostream>
3 using namespace std;
4
5 int main()
6 {
7     // Define a variable named number.
8     int number;
9
10    cout << "Enter a number greater than 0: ";
11    cin >> number;
12    if (number > 0)
13    {
14        int number; // Another variable named number.
15        cout << "Now enter another number: ";
16        cin >> number;
17        cout << "The second number you entered was "
18             << number << endl;
19    }
20    cout << "Your first number was " << number << endl;
21    return 0;
22 }
```

### Program Output with Example Input Shown in Bold

```
Enter a number greater than 0: 2 [Enter]
Now enter another number: 7 [Enter]
The second number you entered was 7
Your first number was 2
```



# Rock-Scissor-Paper

---



- ❖ Write a program that plays the popular rock-scissor-paper game. A scissor can cut a paper, a rock can knock a scissor and a paper can wrap a rock. Use a random generator and generate one number for 0,1,2 representing scissor, rock, paper. Input from user a value as scissor (0), rock (1), paper (2). Determine if User or computer wins. Make sure you validate the input( if the input is invalid send an error message