

# CMPT 1109

## Programming I

Shahriar Khosravi, Ph.D.

Lecture 3

## Plan for Today

- Relational Operators
- The **if** Statement
- The **if-else** Statement
- Nested **if** Statements
- The **if-else if** Statement
- Flags
- Logical Operators
- Validating User Input
- Comparing Characters and Strings
- The Conditional Operator
- The switch Statement
- More about Blocks and Variable Scope

# Relational Operators

# Relational Operators and Expressions

- **Relational operators** are used to compare numbers to determine relative order.
- **Boolean expressions** – statements that are either **true** or **false**.
- **Relational expressions** can be used to assign the value of a Boolean expression to a variable.

`result = x <= y;`

- This statement assigns **0** for **false**, **1** for **true**.
- Do not confuse `=` and `==`.

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

`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**

## Examples

Assume **x** is **10** and **y** is **7**.

Expression	Value
<b>x &lt; y</b>	False, because x is not less than y.
<b>x &gt; y</b>	True, because x is greater than y.
<b>x &gt;= y</b>	True, because x is greater than or equal to y.
<b>x &lt;= y</b>	False, because x is not less than or equal to y.
<b>y != x</b>	True, because y is not equal to x.

Assume **x** is **10** and **y** is **7**, and **z**, **a**, and **b** are of type **int** or **bool**.

Statement	Outcome
<b>z = x &lt; y</b>	z is assigned 0 because x is not less than y.
<b>cout &lt;&lt; (x &gt; y);</b>	Displays 1 because x is greater than y.
<b>a = x &gt;= y;</b>	a is assigned 1 because x is greater than or equal to y.
<b>cout &lt;&lt; (x &lt;= y);</b>	Displays 0 because x is not less than or equal to y.
<b>b = y != x;</b>	b is assigned 1 because y is not equal to x.



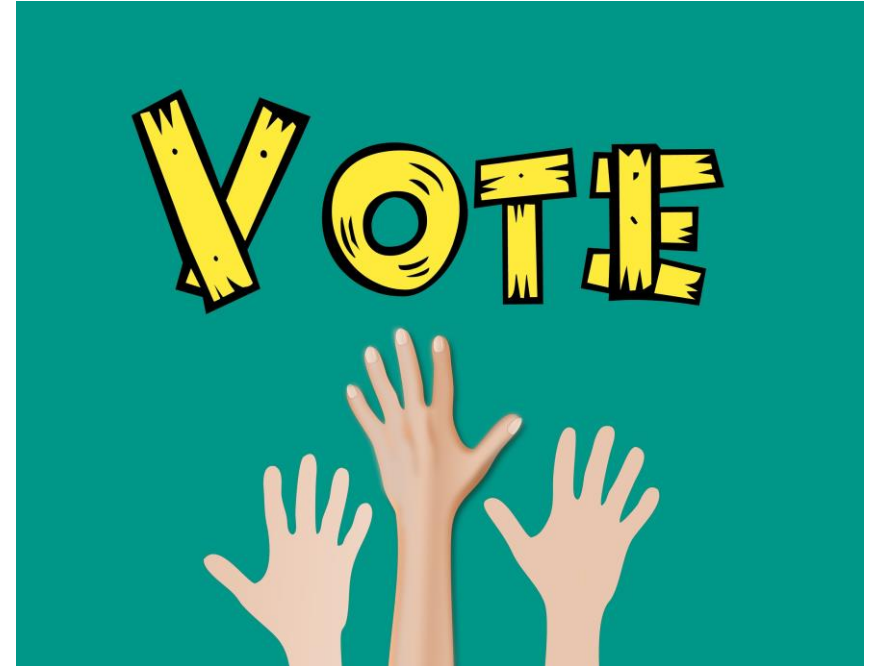
## Poll 1 (Extra Credit)

Relational expressions have a higher precedence than the assignment operator.

`z = x < y;`

- a) True
- b) False
- c) Not sure!

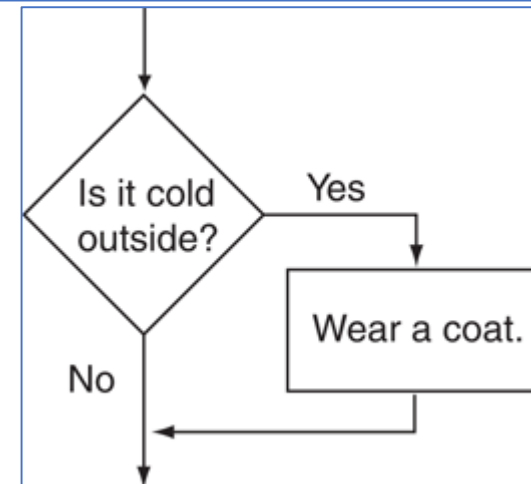
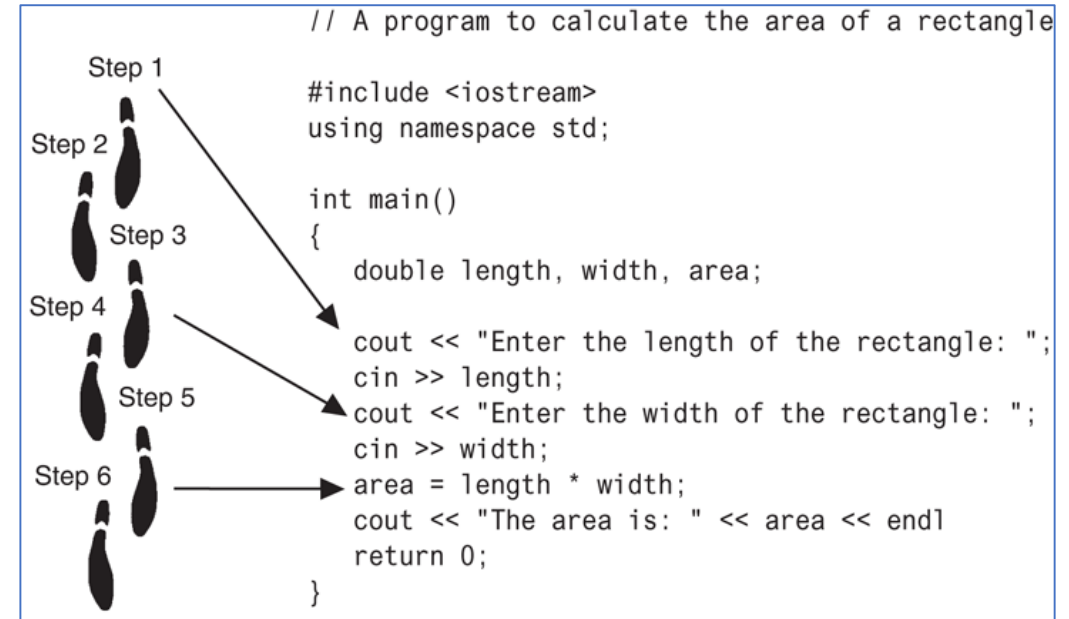
Please use the “Poll” window to participate for extra credit! One answer only please!



# The if Statement

# Sequence Structure vs. Decision Structure

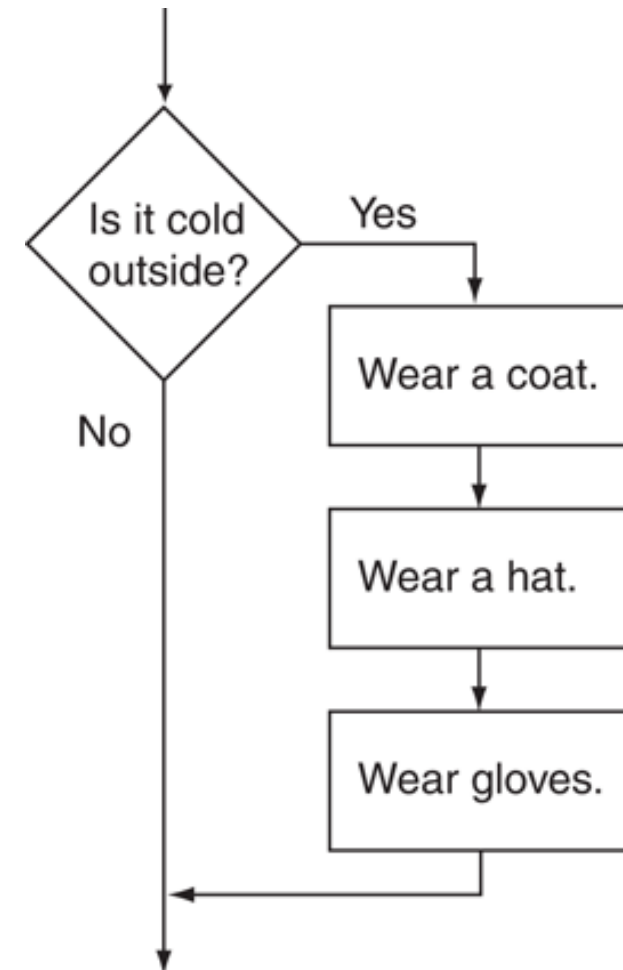
- In a **sequence structure**, the statements are executed in sequence, without branching off in another direction.
- Many algorithms require a program to execute some statements only under certain circumstances.
- This can be accomplished with a **decision structure**.
- In a decision structure's simplest form, a specific action is taken only when a specific condition exists.
- If the condition does not exist, the action is not performed.





## Sequence Structure vs. Decision Structure

- In a **sequence structure**, the statements are executed in sequence, without branching off in another direction.
- Many algorithms require a program to execute some statements only under certain circumstances.
- This can be accomplished with a **decision structure**.
- In a decision structure's simplest form, a specific action is taken only when a specific condition exists.
- If the condition does not exist, the action is not performed.

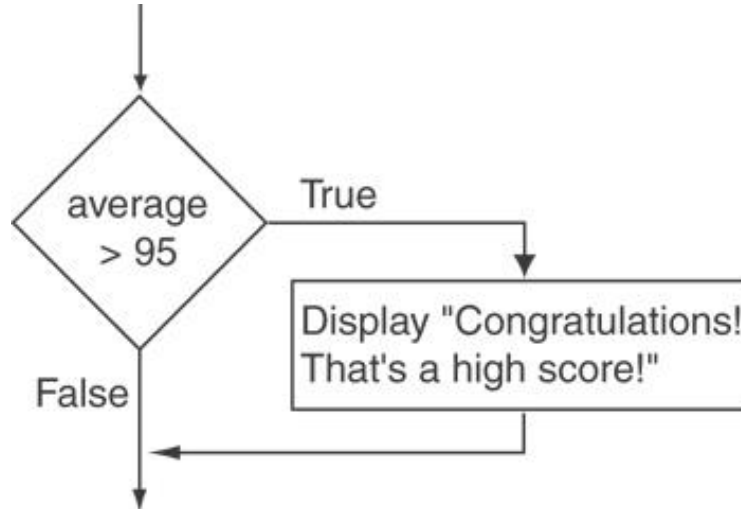


# The if Statement

- General syntax:

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

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



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

int main()
{
    const int HIGH_SCORE = 95;    // A high score is 95 or greater
    int score1, score2, score3;  // To hold three test scores
    double average;              // TO hold the average score

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

# The if Statement

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

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

- Be careful testing **float** and **double** variables for equality.
- 0 is false; any other value is true.**

No semicolon goes here.

```
if (expression)
```

Semicolon goes here.

```
    statement;
```

Statement	Outcome
<pre>if (hours &gt; 40)     overTime = true;</pre>	Assigns true to the bool variable overTime only if hours is greater than 40
<pre>if (value &gt; 32)     cout &lt;&lt; "Invalid number\n";</pre>	Displays the message "Invalid number" only if value is greater than 32
<pre>if (overTime == true)     payRate *= 2;</pre>	Multiplies payRate by 2 only if overTime is equal to true

## Example – Misplaced Semicolon 😞 😞 🙌

```
// This program demonstrates how a misplaced semicolon  
// prematurely terminates an if statement.
```

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

```
int main()
```

```
{
```

```
    int x = 0, y = 10;
```

```
    cout << "x is " << x << " and y is " << y << endl;
```

```
    if (x > y); // Error! Misplaced semicolon
```

```
    cout << "x is greater than y\n"; //This is always  
    executed.
```

```
    return 0;
```

```
}
```

- The compiler will assume you are placing a null statement there. The null statement is an empty statement that does nothing.
- This will prematurely terminate the if statement, which disconnects it from the statement that follows it.

## Example – Numerical Pitfalls ☹️☹️🤖

```
// This program demonstrates how floating-point
// round-off errors can make equality operations unreliable.
#include <iostream>
using namespace std;

int main()
{
    double a = 1.5;           // a is 1.5.
    double b = 1.5;           // b is 1.5.

    a += 0.0000000000000001;  // Add a little to a.
    if (a == b)
        cout << "Both a and b are the same.\n";

    if (a != b)
        cout << "a and b are not the same.\n";

    return 0;
}
```

## Summary of Truth Rules

- Truth is a complicated thing.
- Here is a summary of the rules we have seen so far:
  - When a relational expression is **true**, it has the value **1**.
  - When a relational expression is **false**, it has the value **0**.
  - Any expression that has the value **0** is considered **false** by the **if** statement. This includes the **bool** value **false**, which is equivalent to **0**.
  - Any expression that has any value other than **0** is considered true by the **if** statement. This includes the **bool** value **true**, which is equivalent to **1**.
- The fact that the **if** statement considers **any nonzero** value as **true** opens many possibilities:

```
if (value)
    cout << "It is True!";
```

```
if (x + y)
    cout << "It is True!";
```



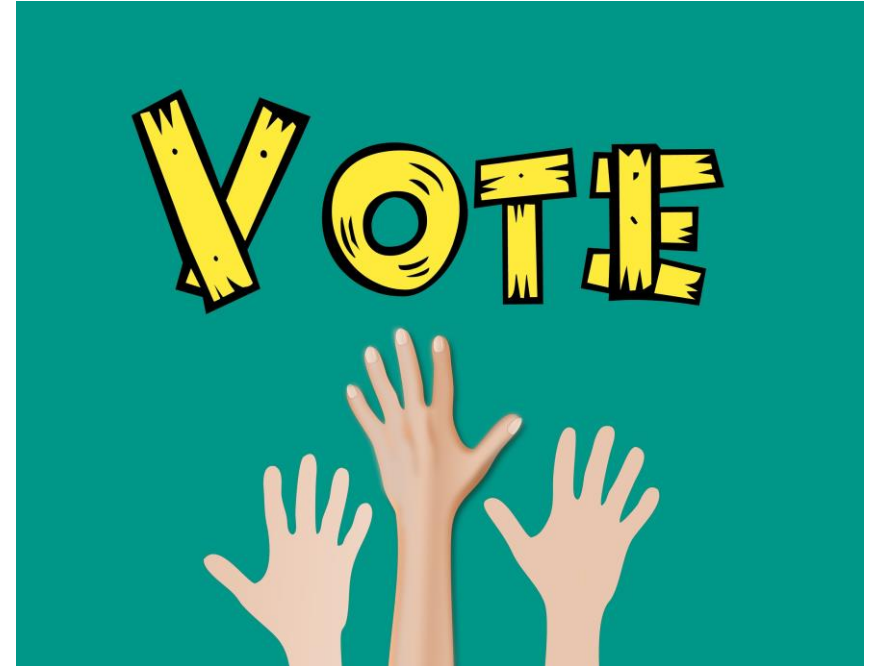
## Poll 2 (Extra Credit)

The following cout statement will execute only if variable x is equal to 2.

```
if (x = 2)  
    cout << "So true!";
```

- a) True
- b) False
- c) Not sure!

Please use the “Poll” window to participate for extra credit! One answer only please!



# 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**.
- It is still okay to include { } when an if statement encloses only one statement.
- Be very careful not to forget braces when more than one statement is to be enclosed in an **if** statement:

```
if (average > HIGH_SCORE)
    cout << "Congratulations!\n";
cout << "That's a high score.\n";
cout << "You deserve a pat on the back!\n";
```

Only this statement is conditionally executed.

These statements are always executed.



DOUGLAS COLLEGE

# The if-else Statement

## The if-else Statement

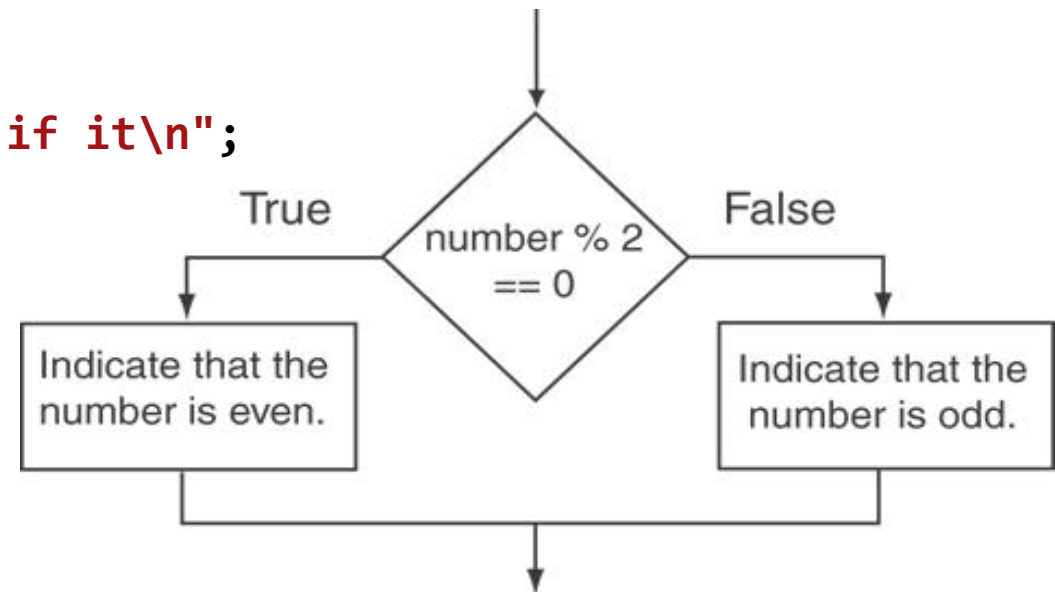
- The **if-else** statement will execute one group of statements if the expression is **true**, or another group of statements if the expression is **false**.
- General Format:

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

- 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.

# The if-else Statement - Example

```
// This program determines if a number is odd or even.  
// If the number is evenly divisible by 2, it is an even number.  
#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;  
}
```



# Example

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

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

    // 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;
}
```



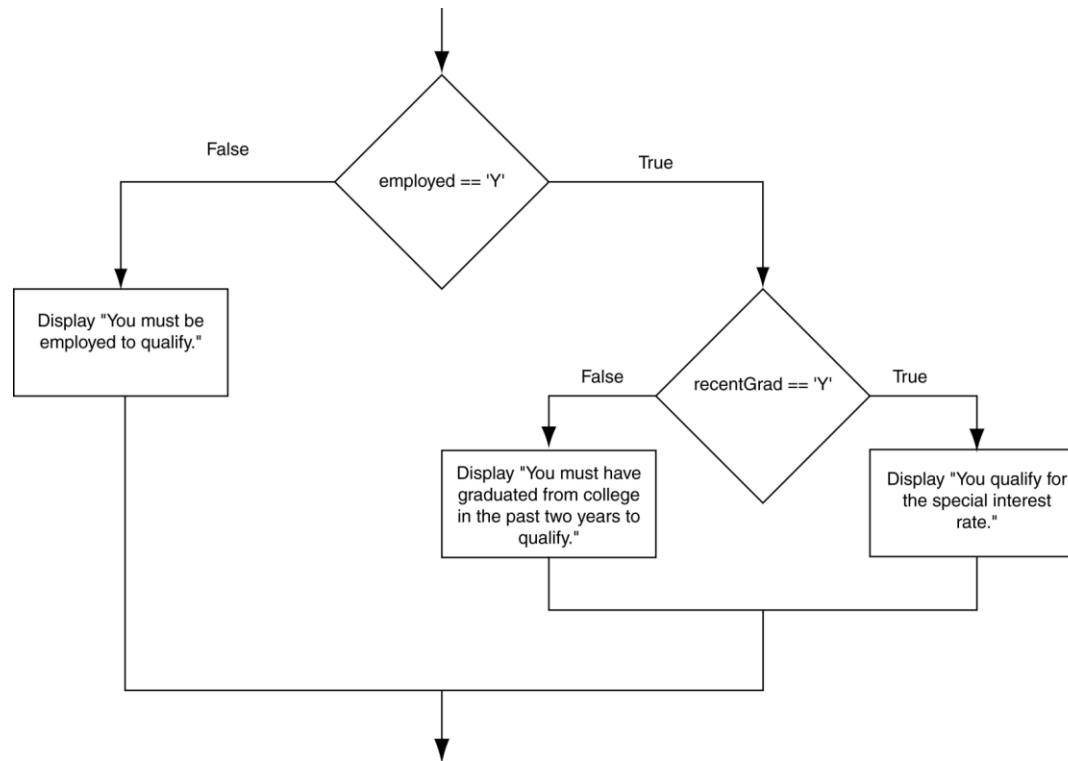


DOUGLAS COLLEGE

# Nested if Statements

# Nested if Statements

- Nested **if** statements can be used to test more than one condition.



```

// This program demonstrates the nested if statement.
#include <iostream>
using namespace std;
int main()
{
    char employed,    // Currently employed, Y or N
        recentGrad;  // Recent graduate, Y or N
    // Is the user employed and a recent graduate?
    cout << "Answer the following questions\n";
    cout << "with either Y for Yes or ";
    cout << "N for No.\n";
    cout << "Are you employed? ";
    cin >> employed;
    cout << "Have you graduated from college ";
    cout << "in the past two years? ";
    cin >> recentGrad;
    // Determine the user's loan qualifications.
    if (employed == 'Y')
    {
        if (recentGrad == 'Y') // Nested if
        {
            cout << "You qualify for the special ";
            cout << "interest rate.\n";
        }
        else // Not a recent grad, but employed
        {
            cout << "You must have graduated from ";
            cout << "college in the past two\n";
            cout << "years to qualify.\n";
        }
    }
    else // Not employed
    {
        cout << "You must be employed to qualify.\n";
    }
    return 0;
}
  
```

# Always Use Proper Indentation

```
if (employed == 'Y')
{
    if (recentGrad == 'Y') // Nested if
    {
        cout << "You qualify for the special ";
        cout << "interest rate.\n";
    }
    else // Not a recent grad, but employed
    {
        cout << "You must have graduated from ";
        cout << "college in the past two\n";
        cout << "years to qualify.\n";
    }
}
else // Not employed
{
    cout << "You must be employed to qualify.\n";
}
```

This if and else go together.

This if and else go together.



DOUGLAS COLLEGE

# The if-else if Statement

## Nested if Statements

- 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."

```
if (expression_1)
{
    statement
    statement
    etc.
}
else if (expression_2)
{
    statement
    statement
    etc.
}
Insert as many else if clauses as necessary
else
{
    statement
    statement
    etc.
}
```

*If expression\_1 is true these statements are executed, and the rest of the structure is ignored.*

*Otherwise, if expression\_2 is true these statements are executed, and the rest of the structure is ignored.*

*These statements are executed if none of the expressions above are true.*

## Example

```
// 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()
{
    // Constants for grade thresholds
    const int A_SCORE = 90,
             B_SCORE = 80,
             C_SCORE = 70,
             D_SCORE = 60;

    int testScore; // To hold a numeric test score

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

    // Determine the letter grade.
    if (testScore >= A_SCORE)
        cout << "Your grade is A.\n";
    else if (testScore >= B_SCORE)
        cout << "Your grade is B.\n";
    else if (testScore >= C_SCORE)
        cout << "Your grade is C.\n";
    else if (testScore >= D_SCORE)
        cout << "Your grade is D.\n";
    else
        cout << "Your grade is F.\n";

    return 0;
}
```





DOUGLAS COLLEGE

# Flags

## Flags

- A **flag** is a variable that signals a condition.
- Usually implemented as a **bool** variable, but it can also be an **int**.
- The value **0** is *usually* considered **false**, and any nonzero value is considered **true**.
- As with other variables, flags must be assigned an initial value before being used.

```
bool salesQuotaMet = false;
```

```
if (sales >= QUOTA_AMOUNT)  
    salesQuotaMet = true;
```

```
else
```

```
    salesQuotaMet = false;
```

```
if (salesQuotaMet)   
    cout << "You have met your sales quota!\n";
```

This is equivalent to:

```
if (salesQuotaMet == true)
```

# Logical Operators

## Logical Operators

- **Logical operators** connect two or more relational expressions into one or reverse the logic of an expression.

Operator	Meaning	Effect
&&	AND	Connects two expressions into one. Both expressions must be true for the overall expression to be true.
	OR	Connects two expressions into one. One or both expressions must be true for the overall expression to be true. It is only necessary for one to be true, and it does not matter which.
!	NOT	The ! operator reverses the “truth” of an expression. It makes a true expression false, and a false expression true.

## Logical Operator Examples

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

<code>(x &gt; y) &amp;&amp; (y &gt; z)</code>	<b>true</b>
<code>(x &gt; y) &amp;&amp; (z &gt; y)</code>	<b>false</b>
<code>(x &lt;= z)    (y == z)</code>	<b>false</b>
<code>(x &lt;= z)    (y != z)</code>	<b>true</b>
<code>!(x &gt;= z)</code>	<b>false</b>

# The Logical && Operator

- If the subexpression on the left side of an **&&** operator is **false**, the expression on the right side will **not** be checked.
- Since the entire expression is **false** if only one of the subexpressions is **false**, it would waste CPU time to check the remaining expression.
- This is called **short-circuit evaluation**.

Truth Table

Expression	Value of Expression
true && false	false (0)
false && true	false (0)
false && false	false (0)
true && true	true (1)

```
// This program demonstrates the && logical operator.
#include <iostream>
using namespace std;

int main()
{
    char employed,    // Currently employed, Y or N
        recentGrad;  // Recent graduate, Y or N

    // Is the user employed and a recent graduate?
    cout << "Answer the following questions\n";
    cout << "with either Y for Yes or N for No.\n";

    cout << "Are you employed? ";
    cin >> employed;

    cout << "Have you graduated from college "
        << "in the past two years? ";
    cin >> recentGrad;

    // Determine the user's loan qualifications.
    if (employed == 'Y' && recentGrad == 'Y')
    {
        cout << "You qualify for the special "
            << "interest rate.\n";
    }
    else
    {
        cout << "You must be employed and have\n"
            << "graduated from college in the\n"
            << "past two years to qualify.\n";
    }
    return 0;
}
```



# The Logical || Operator

- The || operator also performs short-circuit evaluation.
- If the subexpression on the left side of an || operator is **true**, the expression on the right side will **not** be checked.
- Since it's only necessary for one of the subexpressions to be **true**, it would waste CPU time to check the remaining expression.

Truth Table

Expression	Value of Expression
true    false	true (1)
false    true	true (1)
false    false	false (0)
true && true	true (1)

```
// This program demonstrates the logical || operator.
#include <iostream>
using namespace std;

int main()
{
    // Constants for minimum income and years
    const double MIN_INCOME = 35000.0;
    const int MIN_YEARS = 5;

    double income; // Annual income
    int years;      // Years at the current job

    // Get the annual income
    cout << "What is your annual income? ";
    cin >> income;

    // Get the number of years at the current job.
    cout << "How many years have you worked at "
         << "your current job? ";
    cin >> years;

    // Determine the user's loan qualifications.
    if (income >= MIN_INCOME || years > MIN_YEARS)
        cout << "You qualify.\n";
    else
    {
        cout << "You must earn at least $"
              << MIN_INCOME << " or have been "
              << "employed more than " << MIN_YEARS
              << " years.\n";
    }
    return 0;
}
```

# The Logical ! Operator

- The ! operator performs a logical NOT operation.
- It takes an operand and reverses its truth or falsehood

Truth Table

Expression	Value of Expression
!true	false (0)
!false	true (1)

```
// This program demonstrates the logical ! operator.
#include <iostream>
using namespace std;

int main()
{
    // Constants for minimum income and years
    const double MIN_INCOME = 35000.0;
    const int MIN_YEARS = 5;

    double income;    // Annual income
    int years;        // Years at the current job

    // Get the annual income
    cout << "What is your annual income? ";
    cin >> income;

    // Get the number of years at the current job.
    cout << "How many years have you worked at "
          << "your current job? ";
    cin >> years;

    // Determine the user's loan qualifications.
    if (!(income >= MIN_INCOME || years > MIN_YEARS))
    {
        cout << "You must earn at least $"
              << MIN_INCOME << " or have been "
              << "employed more than " << MIN_YEARS
              << "years.\n";
    }
    else
        cout << "You qualify.\n";
    return 0;
}
```

## Precedence and Associativity of Logical Operators

- The **!** operator has a higher precedence than many of the C++ operators.
- To avoid errors, **always enclose its operand in parentheses** unless you intend to apply it to a variable or a simple expression with no other operators.
- The **&&** and **||** operators **rank lower in precedence than the relational operators**, so precedence problems are less likely to occur.
- It does not hurt to use parenthesis anyway!
- Logical operators have **left-to-right associativity**.

Logical Operators in Order of Precedence
!
&&

## Precedence and Associativity Examples

- $!(x > 2)$  is not the same thing as  $!x < 2$ .
- $(a > b) \ \&\& \ (x < y)$  is the same as  $a > b \ \&\& \ x < y$ .
- $(x == y) \ || \ (b > a)$  is the same as  $x == y \ || \ b > a$ .
- In the following expression,  $a < b$  is evaluated before  $y == z$ .

$a < b \ || \ y == z$

- $a < b \ || \ y == z \ \&\& \ m > j$  is equivalent to  
 $(a < b) \ || \ ((y == z) \ \&\& \ (m > j))$   
because  $\&\&$  has a higher precedence than  $||$ .

## 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!
```



DOUGLAS COLLEGE

# 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

# Validating User Input

```
#include <iostream>
using namespace std;
int main()
{
    // Constants for grade thresholds
    const int A_SCORE = 90,
            B_SCORE = 80,
            C_SCORE = 70,
            D_SCORE = 60,
            MIN_SCORE = 0,    // Minimum valid score
            MAX_SCORE = 100;  // Maximum valid score

    int testScore; // To hold a numeric test score
    // Get the numeric test score.
    cout << "Enter your numeric test score and I will\n"
          << "tell you the letter grade you earned: ";
    cin >> testScore;
    // Validate the input and determine the grade.
    if (testScore >= MIN_SCORE && testScore <= MAX_SCORE)
    {
        // Determine the letter grade.
        if (testScore >= A_SCORE)
            cout << "Your grade is A.\n";
        else if (testScore >= B_SCORE)
            cout << "Your grade is B.\n";
        else if (testScore >= C_SCORE)
            cout << "Your grade is C.\n";
        else if (testScore >= D_SCORE)
            cout << "Your grade is D.\n";
        else
            cout << "Your grade is F.\n";
    }
    else
    {
        // An invalid score was entered.
        cout << "That is an invalid score. Run the program\n"
              << "again and enter a value in the range of\n"
              << MIN_SCORE << " through " << MAX_SCORE << ".\n";
    }
    return 0;
}
```



# Comparing Characters and Strings

## Comparing Characters and Strings

- 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'.

Character	ASCII Value
'0'–'9'	48–57
'A'–'Z'	65–90
'a'–'z'	97–122
Blank	32
Period	46

## Comparing Characters – Example

```
// This program demonstrates how characters can be
// compared with the relational operators.
#include <iostream>
using namespace std;

int main()
{
    char ch;

    // Get a character from the user.
    cout << "Enter a digit or a letter: ";
    ch = cin.get();

    // Determine what the user entered.
    if (ch >= '0' && ch <= '9')
        cout << "You entered a digit.\n";
    else if (ch >= 'A' && ch <= 'Z')
        cout << "You entered an uppercase letter.\n";
    else if (ch >= 'a' && ch <= 'z')
        cout << "You entered a lowercase letter.\n";
    else
        cout << "That is not a digit or a letter.\n";

    return 0;
}
```

## Comparing Strings – Example

```
// This program uses relational operators to compare a string
// entered by the user with valid part numbers.
#include <iostream>
#include <iomanip>
#include <string>
using namespace std;

int main()
{
    const double PRICE_A = 249.0, // Price for part A
                PRICE_B = 199.0; // Price for part B

    string partNum;                // Holds a part number

    // Display available parts and get the user's selection
    cout << "The headphone part numbers are:\n"
         << "Noise canceling: part number S-29A \n"
         << "Wireless: part number S-29B \n"
         << "Enter the part number of the headphones you\n"
         << "wish to purchase: ";
    cin >> partNum;

    // Set the numeric output formatting
    cout << fixed << showpoint << setprecision(2);

    // Determine and display the correct price
    if (partNum == "S-29A")
        cout << "The price is $" << PRICE_A << endl;
    else if (partNum == "S-29B")
        cout << "The price is $" << PRICE_B << endl;
    else
        cout << partNum << " is not a valid part number.\n";
    return 0;
}
```

# The switch Statement

# The switch Statement

- The **switch** statement lets the value of a variable or an expression determine where the program will branch.
- The **if-else if** statement allows your program to branch into one of several possible paths.
- The **switch** statement is a similar mechanism. However, **it tests the value of an integer expression and then uses that value to determine to which set of statements to branch.**

```
switch (IntegerExpression)
{
    case ConstantExpression:
        // place one or more
        // statements here
    case ConstantExpression:
        // place one or more
        // statements here
    // case statements may be repeated as many
    // times as necessary
    default:
        // place one or more
        // statements here
}
```

## The switch Statement Rules

- **IntegerExpression** must be an **int** an expression that evaluates to an **int**.
- **exp1** through **expn** must be constant **int** expressions or literals, and must be unique in the **switch** statement.
- **default** is optional but recommended.
- **IntegerExpression** is evaluated first, and compared against **exp1** through **expn**.
- If it matches value **expi**, the program branches to the statement following **expi** and continues to the end of the **switch**.
- If no matching value is found, the program branches to the statement after **default**:

```
switch (IntegerExpression)
{
    case exp1:
        // place one or more
        // statements here
    case exp2:
        // place one or more
        // statements here
    // case statements may be repeated as many
    // times as necessary
    default:
        // place one or more
        // statements here
}
```

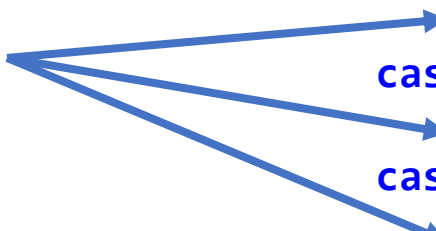
## Example

Without the **break** statements, the program would execute all of the lines from the matching **case** statement **to the end of the block (i.e., “falls through”)**.

```
// The switch statement tells the user something
// they already know: 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;
}
```

A diagram consisting of three blue arrows originating from a single point on the left. Each arrow points to the 'break;' statement at the end of one of the three 'case' statements in the switch block, illustrating that without these breaks, the program would continue to execute the subsequent cases.



## Example

```
// 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;
}
```

## Example

```
// 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;
}
```

## Validating Using switch in Menu Systems Input

- The **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.
- Let us see an example (i.e., `example_01.cpp`).

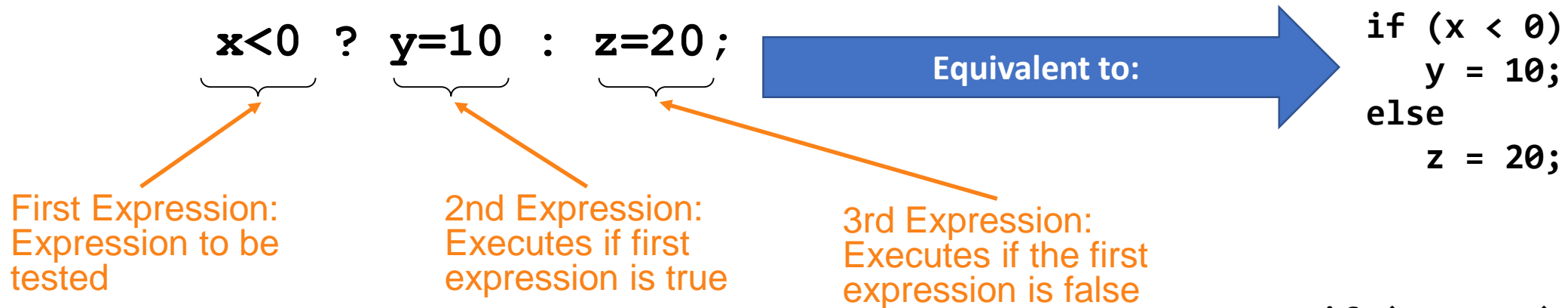


DOUGLAS COLLEGE

# The Conditional Operator

# The Conditional Operator

- The conditional operator `?` can be used to create short **if-else** statements.
- Syntax: `expr ? expr : expr;`



- Remember, in C++ all expressions have a value, and this includes the conditional expression.
- If the first subexpression is true, the value of the conditional expression is the value of the second subexpression. Otherwise, it is the value of the third subexpression.

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

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



DOUGLAS COLLEGE

## 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.
- The scope of a variable is limited to the block in which it is defined. **The variable does not exist outside of that block!**
- Variables defined inside { } have **local** or **block** scope. **They do not exist outside of the block!**
- As a result, when inside a block within another block, we can define variables with the same name as in the outer block.
  - When in inner block, outer definition is not available.
  - This is usually not a good programming practice.

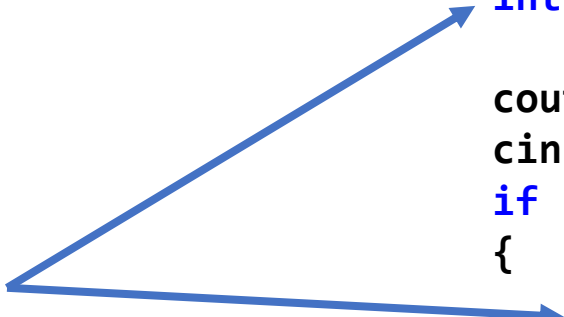
## Example

**NOT** the same variable even though they have the same name. The second **number** only exists inside the if statement's block.

```
// This program uses two variables with the name number.
#include <iostream>
using namespace std;

int main()
{
    // Define a variable named number.
    int number;

    cout << "Enter a number greater than 0: ";
    cin >> number;
    if (number > 0)
    {
        int number; // Another variable named number.
        cout << "Now enter another number: ";
        cin >> number;
        cout << "The second number you entered was "
             << number << endl;
    }
    cout << "Your first number was " << number << endl;
    return 0;
}
```







**Thank you.**  
**DOUGLAS**COLLEGE

**DOUGLAS**COLLEGE