Type casting, logical operators and if/else statement

Explicit Type Conversion

- A type cast expression let's you manually change the data type of a value
- The syntax for type casting is

```
static_cast<DataType>(Value)
```

  - Value is a variable or literal value
  - DataType is the data type that you are converting Value into

Example of Type Casting

```
double number = 3.7;
int val;
val = static_cast<int>(number);
```

What is saved into val?

If Statement

- We may want to execute some code if an expression is `true`, and execute *some other code* when the expression is `false`.
- This can be done with two if statements…

```
if (value >= LIMIT)
{
   // do something
}
if (value < LIMIT)
{
   // do something else
}
```


If/Else

- C++ provides a shortcut to combine two `if` statements:
- The statements in the else clause are executed only when the expression is false.

```
if (expression)
{
  // do stuff
}
else
{
  // do other stuff
}
```


Example

```
int number;
cout << "Enter a number, I'll tell you";
cout << " if it is odd or even: ";
cin >> number;

// use an if/else statement here
```

if/else/if statements

- What if there are more than two alternatives?

```
cout << "Enter two numbers: ";
cin >> num1 >> num2;

if(num1 > num2)
{
  cout << num1 << "is greater" << endl;
}
else if(num2 > num1)
{
  cout << num2 << "is greater" << endl;
}
else
{
  cout << "Numbers are equal" << endl;
}
```


Logical Operators

- There are three logical operators

&&	And
||	Or
!	Not

Evaluating Expressions: And &&

- **`expr1 && expr2`**
- For the complete expression to be true, both expr1 and expr2 have to be true
- Example:

```
(temp > HOT) && (humidity > STICKY)
```

  - These are unbearable heat and humidity conditions
  - Both must be true for the entire expression to be true

Evaluating Expressions: Or ||

- **`expr1 || expr2`**
- The complete expression is true if either expr1 or expr2 is true
- Examples:

```
(salary < MIN_SALARY) || (MARRIED == status)
```

  - To qualify for financial aid, salary has to be less than some minimum salary or you must be married
  - Only one condition has to be true

Evaluating Expressions: Not !

- **`!expr`**
- Unary operator: Negation
- Examples:

```
!(salary < MIN_SALARY)
```

  - What makes this true? False?

Operator Precedence (highest to lowest)

Unary plus & minus	+ - !	Left associative
Multiplication, division, and modulus	* / %	Left associative
Addition & subtraction	+ -	Left associative
Relational operators	< <= > >=	Left associative
Relational operators	== !=	Left associative
Logical AND	&&	Left associative
Logical OR	||	Left associative
Assignment	=	Right associative

Expression Evaluation

- According to the operator precedence and associativity rules given on the previous slide, how will the following expressions be evaluated?

```
x < min + max
min <= x && x <= max
!x == y + 2
x = a + b % 7 * 2
```


Practice

- Are these two code snippets equivalent?

```
int x, y;
cin >> x >> y;
if(x > y)
{
     cout << x;
}
if(x < y)
{
     cout << y;
}
```

```
int x, y;
cin >> x >> y;
if(x > y)
{
     cout << x;
}
else
{
     cout << y;
}
```


Problem

- Write a C++ program segment that allows the user the ability to input an integer from the keyboard.
- If the integer is positive, increment a variable **posCount** by 1. If the integer is **negative**, increment a variable **negCount** by 1. If neither, increment **zeroCount** by 1

```
int posCount = 0, negCount = 0,
    zeroCount = 0;
```