Relational and Logical Operators and Control Structures: Part 1

In this Week

- Compound Arithmetic Operators
- Relational (or Comparison) Operators
- Logical Operators
- Boolean Expressions
- Control Structures: Part 1: if, else if, else
- Scope of Variables
- Ternary Operator (Conditional Operator)
- Remainder Operator and Integer Divisibility
- Variable Swapping
- C++ Built-In Functions
  - Mathematical Functions
  - Random Number Generator Function

Compound Arithmetic Operators

- C++ supports the following compound binary operators

Compound Operator	Example	Description
+=	a += b	a = a + b
-=	a -= 5	a = a - 5
*=	b *= c	b = b * c
/=	a /= 4.0	a = a / 4.0
%=	a %= b	a = a % b

Compound Arithmetic Operators

- Analyze the following program and determine its output

```
int main()
{
    int a = 5, b = 3, c = 7;

    cout << "At first the values of a, b, and c are: ";
    cout << a << ", " << b << ", and " << c << endl;

    a -= b;     //a becomes 2
    b *= c;     //b becomes 21
    c += a;     //c becomes 9
    a /= b;     //a becomes 0
    b %= c;     //b becomes 3

    cout << "At last the values of a, b, and c are: ";
    cout << a << ", " << b << ", and " << c << endl;

    system("Pause");
    return 0;
}
```

Relational/Comparison Operators

- C++ supports the following relational (comparison) binary operators

Relational Operator	Description
==	is equal to
!=	is not Equal to
<	is less than
>	is greater than
<=	is less than or equal to
>=	is greater than or equal to

- Relational operators give either a **true** or a **false** answer

Relational/Comparison Operators

- What is the output of the following program?

```
int main()
{
    int a = 5, b = 3;
    bool x1 = a == b;
    bool x2 = a != b;
    bool x3 = a > b;
    bool x4 = a >= b;
    bool x5 = a < b;
    bool x6 = a <= b;

    cout << "Value of x1 = " << x1 << endl;
    cout << "Value of x2 = " << x2 << endl;
    cout << "Value of x3 = " << x3 << endl;
    cout << "Value of x4 = " << x4 << endl;
    cout << "Value of x5 = " << x5 << endl;
    cout << "Value of x6 = " << x6 << endl;

    system("Pause");
    return 0;
}
```

Logical Operators

- C++ supports the following logical operators

Logical Operator	Description	Evaluation
!	Logical NOT	false if operand is true; true if operand is false
&&	Logical AND	true if and only if both operands are true; otherwise false
||	Logical OR	false if and only if both operands are false; otherwise true

- The operands of logical operators must be true or false values; OR some expressions that give true or false values
- The operator ! has one operand (unary operator)
- The operators && and || have two operands (binary operators)
- ! has precedence over && and ||
- && has precedence over || (see example below). Same type logical operators are executed left to right
- Also the Relational/Comparison operators have precedence over the logical operators

Relational and Logical Operators

Analyse the following program manually and determine the output

```
int main()
{
    float a = 3.6, b = 2.5, c = -7.5;
    cout << "a = " << a << ", b = " << b << ", and c = " << c << endl;
    bool var1 = a > b;  // var1 is now true
    cout << "var1 = a > b gives the result var1 = " << var1 << endl;
    bool var2 = b*c >= a;
    cout << "var2 = b*c >= a gives the result var2 = " << var2 << endl;
    bool var3 = var1 && var2;
    cout << "var3 = var1 && var2 gives the result var3 = " <<  var3 << endl;
    bool var4 = var1 || var2;
    cout << "var4 = var1 || var2 gives the result var4 = " <<  var4 << endl;
    cout << "The expression !(a > c) is evaluated to " << !(a > c) << endl;
    cout << "The expression (b > a) is evaluated to " << (b > a) << endl;
    cout << "The expression ((!(a > c)) && var1 || !var3 && (b > a)) is evaluated to " <<
        ((!(a > c)) && var1 || !var3 && (b > a)) << endl;
    system("pause");
    return 0;
}
```

Boolean Expressions

- C++ expressions that give a boolean (true or false) result
- In C++ true is equivalent to any non-zero number and false is equivalent to zero. Example

  bool flag = false; is equivalent to bool flag = 0;
  bool flag = true; is equivalent to bool flag = 1; //or any other non-zero int, float, double or other number

- Boolean expressions are constructed using relational and logical operators. Examples:

```
int a = 5, b = 3, c = 6;
a*c > b                  ← which is true
(a <= c) && (b > a)      ← which is false
(a != c)|| !(c > a)      ← flag is now true
```

Control Structures: Part 1
The if statement

- C++ supports conditional statements using **if** statements, **if ... else** statements, and **if ... else if ... else** statements

- **<if> statement**: **Syntax**

```
if (boolean expression)← The brackets are required!!!
{
        Block of if statement
}
```

- In this case, the **Block of the if** is **executed** only when the **boolean expression** is evaluated to **true**; otherwise it is skipped
- A program can have several if statements each with its own block. In that case each if statement will work independently

Control Structures: Part 1
The if statement

- Consider the following program and
  - Determine its output and
  - Describe what the program does

```
int main()
{
    int x;
    cout << "Enter an integer ";
    cin >> x;
    if (x >= 0)
    {
        cout << x << endl;
    }
    if (x < 0)
    {
        cout << -x << endl;
    }

    system("Pause");
    return 0;
}
```

Control Structures: Part 1
The if-else statement

- **<if ... else> statement**: **Syntax**

```
if (boolean expression)        ← The brackets ( and ) are required!!!
{
      If Block C++ Statements
}
else
{
     Else Block C++ statements
}
```

- Here, the **if block** is **executed** only when the **boolean expression** is evaluated to **true**. **In this case the else block is automatically skipped!!!**
- Similarly, the **else Block** is **executed** only when the **boolean expression** is evaluated to **false.** **In this case the if block is automatically skipped!!!**
- Thus, the if and else blocks are mutually exclusive. **But one of the two blocks is always executed!!!**

Control Structures: Part 1
The if-else statement

- Consider the following program and
  - Determine its output and
  - Describe what the program does

```
int main()
{
    int x;
    cout << "Enter an integer ";
    cin >> x;
    if (x >= 0)
    {
        cout << x << endl;
    }
    else
    {
        cout << -x << endl;
    }

    system("Pause");
    return 0;
}
```

Control Structures: Part 1
The if - else if … else if statement

- **<if … else if … else if …> statement**: **Syntax**

```
if (boolean expression 1)          ← The ( and ) brackets  are required!!!
{
        If Block C++ Statements
}
else if (boolean expression 2)     ← The ( and ) brackets  are required!!!
{
        Else if Block C++ statements
}
else if (boolean expression 3)
{
        Else if Block C++ statements
}
⋮
else if (boolean expression n)
{
        Else if Block C++ Statements
}
```

- Again these <if ... else if ... else if ... > blocks are mutually exclusive. Once a true boolean expression is found, its block is executed and all the remaining else if statements are skipped. That is IF AT ALL, ONLY ONE BLOCK IS EXECUTED
- If none of the boolean expressions is evaluated to true, all the blocks are skipped!

Control Structures: Part 1
The if - else if … else if statement

- Analyze the following program and determine its output. When are all the blocks skipped? What does the program print if you enter 0?

```
int main()
{
    int n;
    cout << "Enter an integer ";
    cin >> n;
    if (n >= 100)
    {
        cout << "You entered a big positive number." << endl;
    }
    else if (n <= -100)
    {
        cout << "You entered a big negative number." << endl;
    }
    else if (n < 100 && n > 0)
    {
        cout << "You entered a small positive number." << endl;
    }
    else if (n > -100 && n < 0)
    {
        cout << "You entered a small negative number." << endl;
    }

    system("Pause");
    return 0;
}
```

Control Structures: Part 1
The if - else if … else statement

- **<if … else if … else> statement**: Syntax

```
if (boolean expression 1)
{
        If Block C++ Statements
}
else if (boolean expression 2)
{
        Else if Block C++ statements
}
else if (boolean expression 3)
{
        Else if Block C++ statements
}
⋮
else
{
        Else Block C++ statements
}
```

- As before all the blocks are mutually exclusive. The else block is executed if none of the boolean expressions is evaluated to true
- **The biggest difference when there is else block is that in this case ONE OF THE BLOCKS WILL ALWAYS NECESSARILY BE EXECUTED!!!**

Control Structures: Part 1

The if - else if … else statement

Analyze the following program manually and determine the output

```
int main()
{
    int n;
    cout << "Enter an integer ";
    cin >> n;
    if (n >= 100)
    {
        cout << "You entered a big positive number." << endl;
    }
    else if (n <= -100)
    {
        cout << "You entered a big negative number." << endl;
    }
    else if (n < 100 && n > 0)
    {
        cout << "You entered a small positive number." << endl;
    }
    else if (n > -100 && n < 0)
    {
        cout << "You entered a small negative number." << endl;
    }
    else
    {
        cout << "You entered zero." << endl;
    }
    system("Pause");
    return 0;
}
```

Control Structures: Part 1
Remarks on Curly Brackets

- If a block has only one statement that belongs to the block, then the curly brackets can be omitted for quick coding purposes

```
int main()
{
    int n;
    cout << "Enter an integer ";
    cin >> n;
    if (n >= 100)
        cout << "You entered a big positive number." << endl;
    else if (n <= -100)
        cout << "You entered a big negative number." << endl;
    else if (n < 100 && n > 0)
        cout << "You entered a small positive number." << endl;
    else if (n > -100 && n < 0)
        cout << "You entered a small negative number." << endl;
    else
        cout << "You entered zero." << endl;
    system("Pause");
    return 0;
}
```

Control Structures: Part 1
Remarks on Curly Brackets

- If the opening and closing curly brackets of a **Block** are omitted, then **only one C++ statement belongs to the block**
- Analyze the following program and determine its output.

```
int main()
{
    int x =-2;
    if (x > 0)
        x = 3 * x;
        cout << x << endl;
    cout << x << endl;

    system("Pause");
    return 0;
}
```

Control Structures: Part 1
Scope of Variables

- The same variable name can not be declared more than once inside the same block even if you use different data types

```
int main()
{
    int x = -5;
    cout << "x has a value " << x << endl;
    //int x;    //Error you can not re-declare
    cout << "Enter an integer ";
    cin >> x;
    cout << "x has value " << x << endl;

    system("Pause");
    return 0;
}
```

Control Structures: Part 1
Scope of Variables

- Any variable declared inside a block remains valid ONLY inside the block.
- We say the scope of any C++ variable is only inside the block it is declared in
- Analyze the following program and determine its output

```
int main()
{
    int x = -5;
    if (x < 0)
    {
        cout << "x has value " << x << endl;
        int y = 2;
        cout << "y has value " << y << endl;
    }
    cout << "x has a value " << x << endl;
    //cout << "y has a value " << y << endl;    //This is error

    system("Pause");
    return 0;
}
```

Control Structures: Part 1
Scope of Variables

- It is allowed to declare a variable inside a block even if the same variable name exists outside the block
- In this case, the inner most block variable shadows an existing same name variable when execution reaches the inner most block

```
int main()
{
    int x = -5;
    if (x < 0)
    {
        cout << "x has value " << x << endl;
        int x = 5;
        cout << "x has value " << x << endl;
        int y = 2;
        cout << "y has value " << y << endl;
    }
    cout << "x has a value " << x << endl;
    //cout << "y has a value " << y << endl;//This is error

    system("Pause");
    return 0;
}
```

Control Structures: Part 1
Block inside Block

- A C++ block can contain other valid blocks inside it. Analyze the following program and determine its output.

```
int main()
{
    int n;
    cout << "Enter an integer ";
    cin >> n;
    if (n >= 100)
        cout << "You entered a big positive number." << endl;
    else if (n <= -100)
        cout << "You entered a big negative number." << endl;
    else
    {
        if (n < 100 && n > 0)
            cout << "You entered a small positive number." << endl;
        else if (n > -100 && n < 0)
            cout << "You entered a small negative number." << endl;
        else
            cout << "You entered zero." << endl;
    }
    system("Pause");
    return 0;
}
```

- Can we modify the code in order to remove unnecessary boolean expressions?

Short Circuit Evaluation

- Consider the logical AND (&&) operator
- For any boolean expression **exp**, we can see that (**false** && **exp**) == (**exp** && **false**) == **false**
- Moreover, for any boolean expression **exp**, we can see that (**true** && **exp**) == (**exp** && **true**) == **exp**
- Similarly, for any boolean expression **exp**, we can see that (**true** || **exp**) == (**exp** || **true**) == **true**
- Moreover, for any boolean expression **exp**, we can see that (**false** || **exp**) == (**exp** || **false**) == **exp**
- We refer to these avoidance of unnecessary redundancy as short circuit evaluations

C++ Ternary (or Conditional) Operator

- C++ provides a ternary operator (three operands) which provides a convenient conditional operation
- **Syntax**

  bool_expression ? Statement1 : Statement2;
- In this case, Statement1 is executed if the boolean expression is true in which case statement2 is skipped; otherwise Statement2 is executed and statement1 is skipped.

Ternary Conditional Operator Example

```
int main()
{
    int x;
    cout << "Enter an integer ";
    cin >> x;
    x >= 0 ? cout << x << endl : cout << -x << endl;
    system("Pause");
    return 0;
}
```

The ternary conditional operator is the same as the <if ... else> statement

```
int main()
{
    int x;
    cout << "Enter an integer ";
    cin >> x;
    if (x >= 0)
        cout << x << endl;
    else
        cout << -x << endl;
    system("Pause");
    return 0;
}
```

Remainder Operator and Integer Divisibility

- Recall that the **%** operator computes remainder of division arithmetic
- Now, we would like to use this operator in order to see if a given integer divides another integer
- Consider integer variables **x** and **y**. When do we say that **y** divides **x**?
- We say **y** divides **x** when the division operation **x/y** gives a remainder of **0**
- Therefore, we notice that **y** divides **x** if and only if **x % y** is evaluated to **0**
- Also we note that the terminologies
  - **y divides x**
  - **y is a factor of x**
  - **x is divisible by y** and
  - **x is a multiple of y**

  all mean the same thing

Remainder Operator and Integer Divisibility

- Analyze the following program and determine its output if
  - The input values for **a** and **b** are respectively 6 and 4
  - The input values for **a** and **b** are respectively 6 and -3
  - The input values for **a** and **b** are respectively 0 and 4
  - The input values for **a** and **b** are respectively 6 and 0

```
int main()
{
    int a, b;
    cout << "Enter two integers ";
    cin >> a >> b;
    if (a % b == 0)
        cout << a << " is divisible by " << b << endl;
    else
        cout << a << " is not divisible by " << b << endl;

    system("Pause");
    return 0;
}
```

Variable Swapping Code

- Consider the program below. Insert the missing code in the box shown in order to swap the values of the variables **a** and **b**
- Swapping means interchanging the values of the variables **a** and **b**
- For example if you enter the integer value **5** for **a** and the integer value **8** for **b**, then the program must have the following output:

```
Originally the values of a and b are 5 and 8
After swapping them the values of a and b are 8 and 5
```

```
int main()
{
    int a, b;
    cout << "Enter two integers: ";
    cin >> a >> b;
    cout << "Originally the values of a and b are " << a << " and " << b << endl;
```

Fill the required code here in order to swap the values of the variables a and b

```
    cout << "After swapping them the values of a and b are " << a << " and " << b << endl;
    system("Pause");
    return 0;
}
```

C++ Built-In Functions

- C++ comes with lots of built-in functions
- The built-in functions come embedded in C++ libraries; therefore we need to include any required libraries in order to use the built-in functions
- The most commonly used built-in C++ functions are mathematical functions and random number generation function

C++ Built-In Mathematical Functions

- In order to access C++ built-in mathematical functions we need to include the math library as follows

  #include <cmath>

- The math library has many functions such as the absolute value function, the power function, the square root function, rounding functions, trigonometric functions, and much more
- See example code below...

C++ Built-In Mathematical Functions

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    int a = -3;
    double b = 6.25, c = -1.097;

    a = abs(a); //absolute value function
    cout << "Absoluter value function: = " << a << endl;

    double answer = pow(a, b);  //power function
    cout << "Power function " << answer << endl;

    //rounding up function
    cout << "Rounding up function " << ceil(b) << endl;

    //rounding down function
    int x = floor(c);
    cout << "Rounding down function " << x << endl;

    //Rounding up and down functions
    floor(c) > ceil(c) ? cout << "Wow!" << endl : cout << "Expected!" << endl;

    answer = sqrt(b);   //Square root function
    cout << "Square root function " << answer << endl;

    //Functions inside an arithemetic
    answer = (0.5 * sqrt(b)) / floor(b);
    cout << answer << endl;

    //Trignometric functions: sin,cos,tan,asin,acos,atan
    answer = sin(3.14/2); //Trignometric functions use radian measure NOT degree
    cout << "Sine function " << answer << endl;
    //Please note that C++ trignometric functions use radian NOT degree

    system("pause");
    return 0;
}
```

C++ Built-In Mathematical Functions Practical Example

- Write a C++ program that solves a quadratic equation $ax^2 + bx + c = 0$
- In your program, you will first read the three coefficients a, b, and c (as floats or doubles) and then use these coefficients to solve for the roots of the quadratic equation given by

$$\frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

C++ Built-In Mathematical Functions Practical Example

```
#include <iostream>
#include <cmath>
using namespace std;
int main()
{
    double a, b, c;
    cout << "Please enter the coefficients a, b and c: ";
    cin >> a >> b >> c;
    double d = pow(b, 2) - 4*a*c;
    if (d < 0)
        cout << "No real roots." << endl;
    else if (d > 0)
    {
        double x1 = (-b + sqrt(d)) / (2*a);
        double x2 = (-b - sqrt(d)) / (2*a);
        cout << "Two real roots: " << x1 << " and " << x2 << endl;
    }
    else
        cout << "One real root: " << -b/(2*a) << endl;
    system("Pause");
    return 0;
}
```

C++ Built-In Random Number Generator Function

- In order to work with random numbers, include the cstdlib library as follows:

  #include <cstdlib>

- Then use the built-in function **rand()** to generate random **integers** in the range **[0, RAND_MAX]**
- **RAND_MAX** is a constant integer defined in cstdlib library whose value is **32,767** in MSVC++ 2010 Express Edition
- See the following example…

```cpp
#include <iostream>
#include <cstdlib>


using namespace std;

int main()
{
    int randomNum = rand();
    cout << randomNum << endl;

    system("pause");
    return 0;
}
```

C++ Built-In Random Number Generator Function

- Careful observation will reveal that every time we run the previous program, the same random integer will be printed
- In order to get different output at each execution of the program, we need to seed the random number generator as follows

```cpp
#include <iostream>
#include <cstdlib>
#include <ctime> //Library to use time


using namespace std;

int main()
{
    srand(time(nullptr)); // Use current time as seed
    int randomNum = rand();
    cout << randomNum << endl;

    system("pause");
    return 0;
}
```

C++ Built-In Random Number Generator Function

- As can be seen in the previous program; the C++ rand function is designed to generate random integers in the range **[0, RAND_MAX]**
- But what if we want to generate random integers in a restricted interval **[a, b]** where **a** and **b** are some given integers with **a ≤ b**
- Then we need to perform some computation on a random number generated in order to map it to one of the elements of **[a, b]**

C++ Built-In Random Number Generator Function

- The following procedure (algorithm) maps a random number generated to one of the elements of **[a, b]**
- Given integers **a** and **b** such that **a ≤ b**, do the folloiwing

  Step 1. **count = b – a + 1**

  Step 2. **r = rand() ……**Now r is in [0, RAND_MAX]

  Step 3. **r = r % count ……** Now r is in [0, count-1]

  Step 4. **r = r + a……** Now r is in [a, count -1 + a]
- Observe that **count -1 + a = (b – a + 1) -1 + a = b**. Therefore the value of **r is now in [a, b]**

C++ Built-In Random Number Generator Function

Example: Generating 30 random integers from [-3, 5] .

```
#include <iostream>
#include <cstdlib>
#include <cmath>
#include <ctime>

using namespace std;

int main(){
    srand(time(nullptr));
    int k = 0;
    while (k < 30)
    {
        int r = rand() % 9 + -3; //[-3, 5]
        cout << r << endl;
        k++;
    }
    system("pause");
    return 0;
}
```

C++ Built-In Functions: Summary

- Obviously, we can not list all C++ built-in functions here
- As such we usually look on a reference material to get additional information
- To get additional information on mathematical functions, see http://www.cplusplus.com/reference/cmath/
- To get additional information on random number generator functions, see http://www.cplusplus.com/reference/cstdlib/