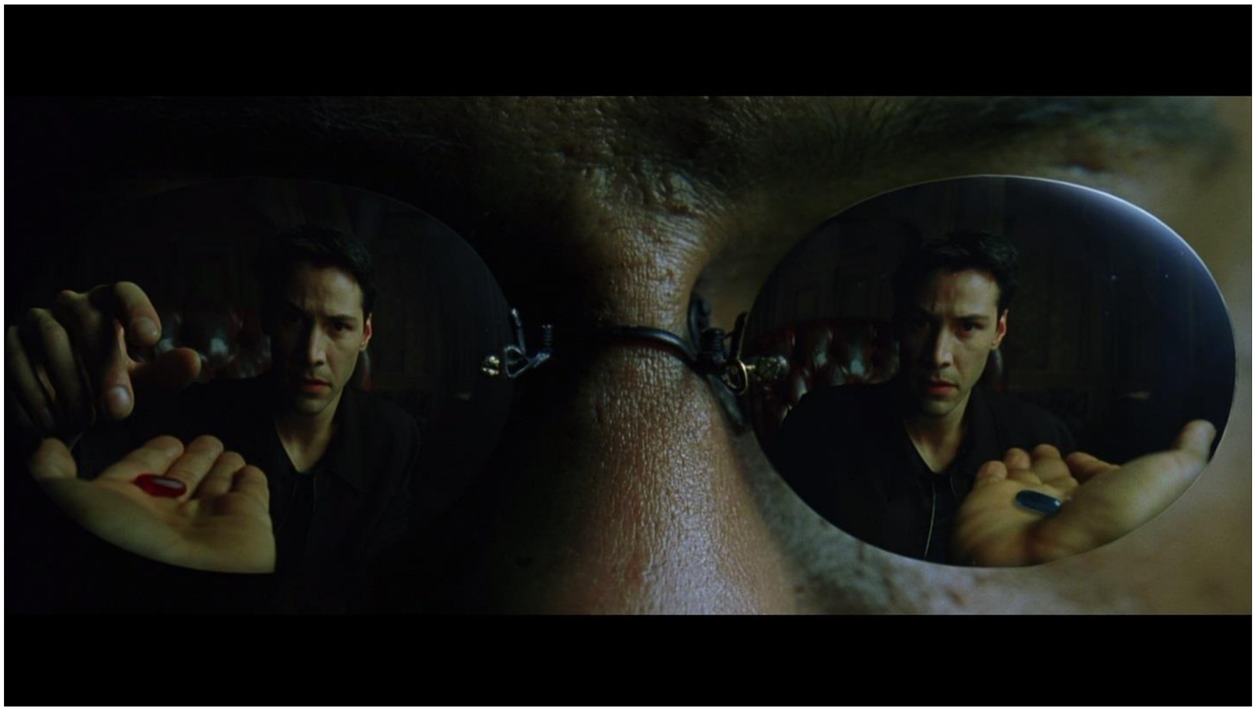
LECTURE #5

Readings: **Chapter 2**: 2.3 to the end of chapter, **Chapter 3:** 3.1



THINGS TO REMEMBER:

- Remember how to create a **Hello World** program in C++
- Remember the **main function**
- **cin** >> and **cout** <<
- Remember **compile vs scripting**

PROGRAM FLOW CONTROL:

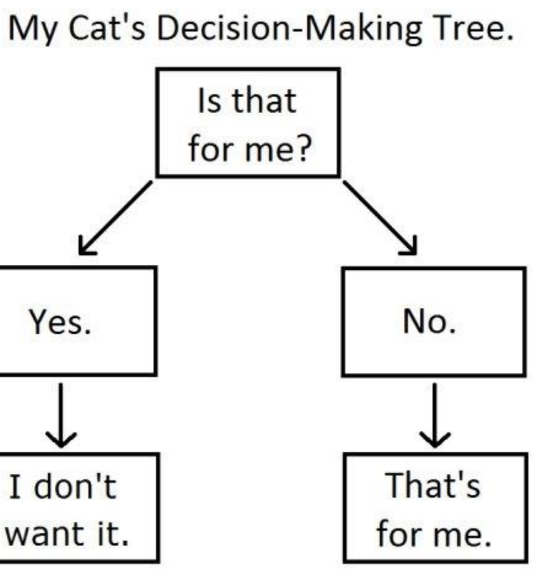
- The hello world program is a **linear program**. This means that the program has only one path to follow. **Each instruction is executed one right after the other. There is no decisions that need to be made**.

- Most programs are not linear, unless they are meant to execute one specific task, but most programs are to be executed providing the user options. Think if you are buying your book on Amazon, do you want it new or used, do you want it to be delivered in 3 days or 7, etc.

- The first decision structure that we are going to learn is the **if-structure**. It is the easiest one to learn but that can be very powerful when combined with logical operators.

IF – STATEMENT:

- If you remember in your introduction class to computer science, flowcharts is something that we use all the time. For example:



- As funny and truthful as the previous image is (you know it is if you have a cat), it shows the principle of if-statements in programming

- **A more formal definition for an if-statement is a structure that allows us to branch in a programming following weather a decision is True or False**.

- In our previous example the **True branch is represented by the Yes decision**, and the **False branch is followed with the No decision**.

- The if-statement structure looks as follows:

```
if ( condition )
    // then perform block of code is expression is True
else
    // perform this block of code if expression is False
```

- In C++, it looks like this:

```
if ( true_or_false condition )
{
        code_if_expression_is_true;
}
else

{

        code_if_expression_is_false;
}
```

WHAT IS A TRUE OR FALSE CONDITION?

- Yes, we are going to explain what a true or false condition is because as you will see in the future, it will cause you all sorts of problems if you are not sure how to construct one.

- There are **three elements you need to create a true or false** condition:
    - **Element #1**
    - **Relational or Logical Condition**
    - **Element #2**

- The elements can be numbers, characters, strings, etc

- The condition can be:
    - **A relational operator:**
        - **>** Greater than
        - **<** Less than
        - **>=** Greater or equal to
        - **<=** Less or equal to
        - **==** Equal to (remember that == is a relational operator, = is an assignment)
        - **!=** Not equal

    - **A logical operator**:
        - **&&** AND
        - **||** OR
        - **!** Not

EXAMPLE:

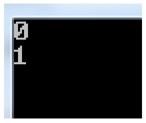
- Start with the following lines of code:
  ```
  int a = 0;
  int b = 1;
  ```

- We can write two simple expressions:
  - a > b
  - b > a

- These are true or false conditions:
  - **a > b** , the result is **False** because 0 is not greater than 1
  - **b > a**, the result is **True** because 1 is not greater than 0
  - We can write two simple expressions:

- In C++ we can write the following lines of code:
  ```
  cout << ( a > b ) << endl;
  cout << ( a < b ) << endl;
  ```

- The result of this is:



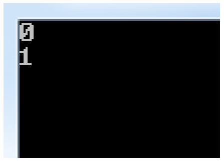
```
0
1
```

- How to interpret these results:
  - Is the result printing the value of a or b? **No**
  - Is the result related to the values of a or b? **No**
  - What is the result if you change the values of a and b?

    ```
    int a = 10;
    int b = 20;

    cout << ( a > b ) << endl;
    cout << ( a < b ) << endl;
    ```

  - Result:



```
0
1
```

- **What does this mean?**
  - **REMEMBER THIS!**
  - **C++ understands 0 as False, and any other number as True. It is not just 1 that is true, you can use any other value as we will see later and we will notice that C++ understands it at True.**

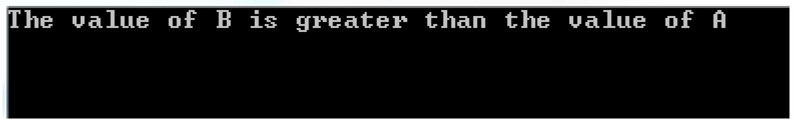
WRITING A C++ IF-STATEMENT:

- Instead of just printing 1 or 0, lets print a simple message :

```
int a = 10;
int b = 20;

if ( a > b )
{
   cout << "The value of A is greater than the value of B" << endl;
}
else
{
   cout << "The value of B is greater than the value of A" << endl;
}
```

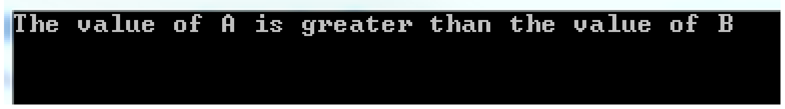
- Result:



```
The value of B is greater than the value of A
```

- This program will always follow the False block of code (or code path) because the values never change. If we change the value of a to 50, the code always follows the True block of code.

```
int a = 50;
```



```
The value of A is greater than the value of B
```

- If we want to make it a more useful program to test whether one number is greater than the other, the only thing that we need to do is to add the following lines of code:

```
   cout << "Enter the value of A: " << endl;
   cin >> a;

   cout << "Enter the value of B: " << endl;
   cin >> b;
```

- With this the user can select the values for both a and b, and the program will test whether A is greater than B, or B greater than A.

SOME PROGRAMMING TIPS:

- If-statements do not require the else section. You can write the previous program to display something to the screen only when a > b:

  ```
  if ( a > b )
  {
      cout << "The value of A is greater than the value of B" << endl;
  }
  ```

- When you do this, nothing will be displayed to the console when a > b is False.

- When you only execute a single line of code in each block of code you can write the if statement as follows:

  ```
  if ( a > b )
          cout << "The value of A is greater than the value of B" << endl
  else
          cout << "The value of B is greater than the value of A" << endl;
  ```

- This will execute the same way as the previous one. If you want to add more lines of code you will need to write them within { }.

- **Since you are just learning C++, please get used to always use { } as in the original example. Once you become more and more familiar with the code you can avoid then when possible,**

- This is a very simple example, so the variable names a and b are ok to use. In more complicated programs use more descriptive variables. For example we could have used value_x and value_y, or value_one and value_two. **Yes, you are going to type more, but trust me, it is easier to figure out your code and your logic than using random variables like a, b, c, x, y, z or random names like a variable called bob. I will take ½ of your project grade if you have a variable with the name bob, so don't try to be funny ☺.**

RELATIONAL VS LOGICAL OPERATIONS:

- In the previous example we used on of the relational operations to determine whether a number if larger than another.

- Anytime we do tests on numbers or any other mathematical combination, we will use these kinds of operators

- These kind of operations can also be used with characters , remember that characters also have a numerical value assigned to them.

- Logical operators are operators that we will use with boolean values

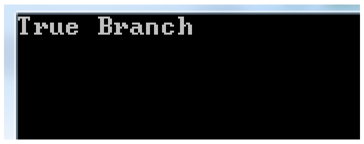
- **A boolean value is a True or False value, in the case of C++ there is a bool type that we can use that uses two values true and false. Also remember how C++ sees True (any other number than 0) and False (0).**

- We can try the following code:

```
bool is_true = true;
bool is_false = false;

if ( is_true )
{
        cout << "True Branch" << endl;
}
else
{
        cout << "False Branch" << endl;
}
```

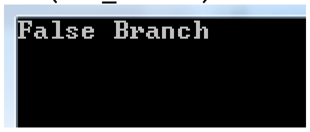
- The result of this code is :



```
True Branch
```

- And if we change the if statement to:

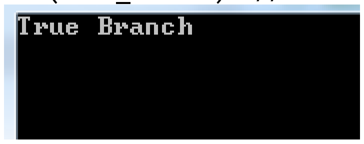
```
if ( is_false )
```



```
False Branch
```

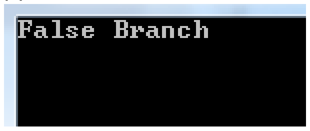
- You can also combine these using the logical operators:

```
if (! is_false )  // read as not (false ) => true
```



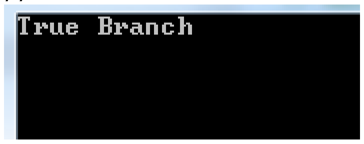
```
True Branch
```

```
if (is_true && is_false )
// read as true AND false, the result is False. In order to be true both
// sides of the AND have to be true, if one side or both are false, the result
// is false
```



```
False Branch
```

```
if (is_true || is_false )
// read as true OR false, the result is True. In order to be true at least
// sides of the OR have to be true. It is false when both are false.
```



```
True Branch
```

BUILDING MORE COMPLEX EXPRESSIONS:

- Now that we know how to use the operational expressions , we can start building more complicated expressions.

EXAMPLE

- How to test if a number is within certain range?
- **Explanation:** We want a program that says whether a number lies within a given range.
- For example, is 5 within the range [2, 10]?, **the answer is Yes.**
- Another example, is 5 within the range [1,2]? **The answer is No.**
- Key elements:
    - In order to test is 5 in the range of [2, 10] we can picture this as follows:
      2 3 4 5 6 7 8 9 10

- Notice two important things:
    - 5 is greater than 2 (the beginning of the range)
    - 5 is less than 10 (the end of the range)

- One more important thing: What if the range is [5, 10]?
    - The answer is still True
    - But greater or less won't work, so for this one we need to use >= and <=

- Now that we understand the key elements of the problem we can think about the solution:
    - We need three variables:
        - One value to test, one for the start of the range, and one for the end of the range

        ```
        int value;
        int start_range;
        int end_range;
        ```

    - We need to test that the value is:
        - Greater or equal than the start of the range:
          `( value >= start_range )`
        - Less or equal than the end of the range:
          `( value <= end_range )`
        - The expression then should be true when the value is greater or equal than the start of the range **AND** less or equal to end of range value.
        - I am not showing you the code because you have to implement this for the project :D