CT4019 PROGRAMMING AND MATHEMATICS

Flow Control and Logical Operators

Attendance PIN

xxxx

A Basic Program

```
#include <iostream>

int main()
{
    int i;  // Create an integer variable
    std::cout << "Enter a whole number and hit enter: " << std::endl;  // Print a string
    std::cin >> i;  // Take in some user input, to n
    std::cout << "You entered the number: " << i << std::endl; // Print out our result
    system("pause");   // Pause so we can see the result
    return 0;   // Exit the program
}
```

Output

```
Enter a whole number and hit enter: 9
You entered the number: 9.
Press any key to continue . . .
```

BORING

A Game Context

```cpp
#include <iostream>

int main()
{
    std::cout << "You are at a crossroads, which way would you like to go?" << std::endl; // Print a question

    std::cout << "1. Go North" << std::endl;
    std::cout << "2. Go East" << std::endl;
    std::cout << "3. Go South" << std::endl; // Print possible answers
    std::cout << "4. Go West" << std::endl;

    int iChosenDirection;
    std::cin >> iChosenDirection; // Create a variable and use it to store the selection

    std::cout << "You went North and were eaten by bears." << std::endl;
    std::cout << "You went East and fell down a bottomless pit." << std::endl;
    std::cout << "You went South and were lost forever in a hedge maze." << std::endl; // Print possible outcomes
    std::cout << "You went West and found hte magic Egg of Mantumbi." << std::endl;

    system("pause"); // Pause to view the output

    return 0; // Exit the program
}
```

Output

```
You are at a crossroads, which way would you like to go

1. Go North
2. Go East
3. Go South
4. Go West

1
You went North and were eaten by bears.
You went East and fell down a bottomless pit.
You went South and were lost forever in a hedge maze.
you went West and found the magic Egg of Mwahahaha.
Press any key to continue . . .
```

RUBBISH

The “if” Statement

- Following the “if” keyword we have a set of normal brackets
- Inside these brackets we add an expression which must evaluate to either true, or false
- This “expression” is essentially our condition
- If the expression evaluates to true then the code inside these curly braces will be executed
- Additionally we can add several other ‘else if’ statements to check for different conditions
- Finally we can add one last “else” statement, this is a catch-all. If none of the other expressions evaluate to true then we are left executing the logic in the else statement.

```
if( expression )
{
    ...logic
}
```

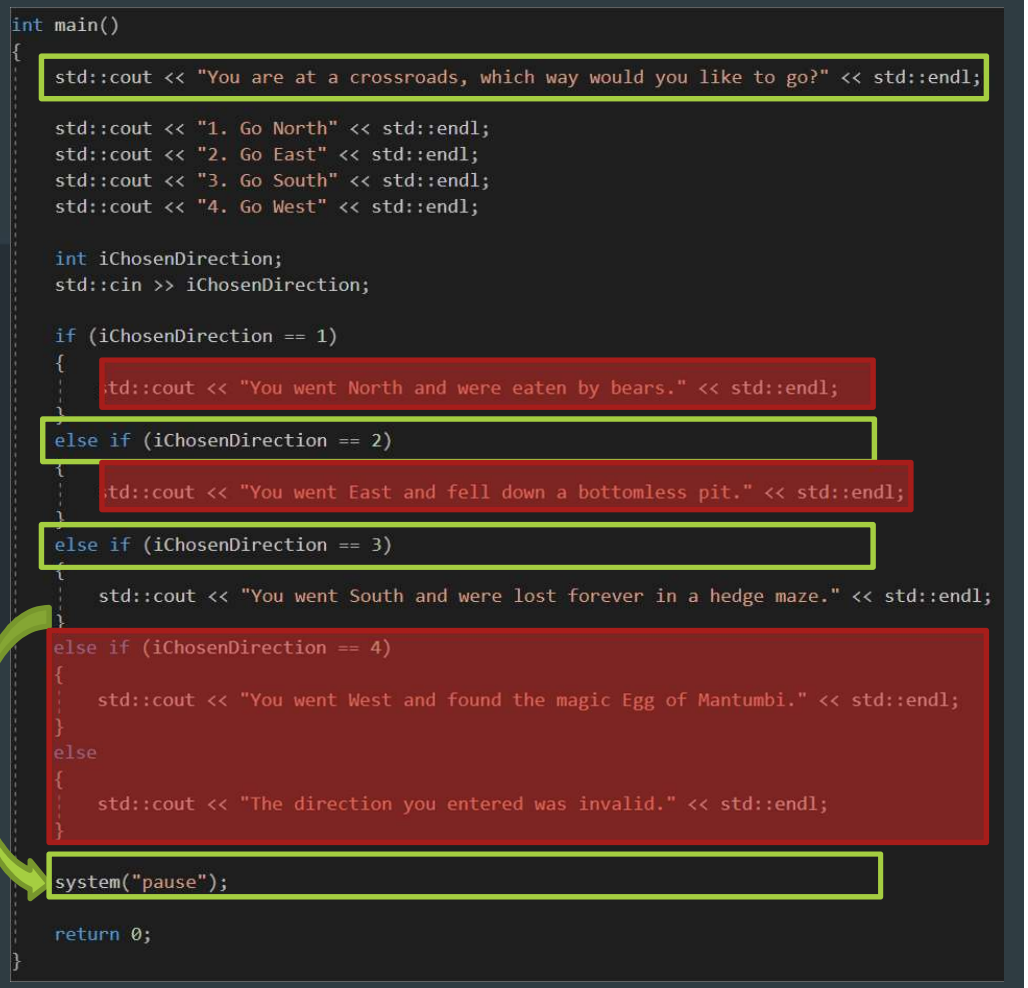
```
if(iNumber == 1)
{
    std::cout << "iNumber equals 1";
}
else if(iNumber == 2)
{
    std::cout << "iNumber equals 2";
}
else
{
    std::cout << "iNumber is not 1 or 2?!";
}
```

Let's put this in to context

```
if(iChosenDirection == 1)
{
    std::cout << "You went North and were eaten by bears." << std::endl;
}
else if(iChosenDirection == 2)
{
    std::cout << "You went East and fell down a bottomless pit." << std::endl;
}
else if(iChosenDirection == 3)
{
    std::cout << "You went South and were lost forever in a hedge maze." << std::endl;
}
else if(iChosenDirection == 4)
{
    std::cout << "You went West and found the magic Egg of Mantumbi!" << std::endl;
}
else
{
    std::cout << "The direction you entered was invalid" << std::endl;
}
```

The Final Product

- So let's start the program
- Print all the initial text and wait for the player's input
- So let's say the player enters "3"
- The flow control we've put in place will check each part of the "if" until it finds a match
- Once we find a match we execute the code within the associated braces
- After that we exit the whole if statement and continue on with the program
- That means, on this particular execution of the program, none of this code was executed



```
int main()
{
    std::cout << "You are at a crossroads, which way would you like to go?" << std::endl;

    std::cout << "1. Go North" << std::endl;
    std::cout << "2. Go East" << std::endl;
    std::cout << "3. Go South" << std::endl;
    std::cout << "4. Go West" << std::endl;

    int iChosenDirection;
    std::cin >> iChosenDirection;

    if (iChosenDirection == 1)
    {
        std::cout << "You went North and were eaten by bears." << std::endl;
    }
    else if (iChosenDirection == 2)
    {
        std::cout << "You went East and fell down a bottomless pit." << std::endl;
    }
    else if (iChosenDirection == 3)
    {
        std::cout << "You went South and were lost forever in a hedge maze." << std::endl;
    }
    else if (iChosenDirection == 4)
    {
        std::cout << "You went West and found the magic Egg of Mantumbi." << std::endl;
    }
    else
    {
        std::cout << "The direction you entered was invalid." << std::endl;
    }

    system("pause");

    return 0;
}
```

Output

Mostly Positive

Logical Operator - Equality

- You will have noticed the previous examples use two equals signs to check for equality.
- This is known as a logical operator.
- Logical operators are used to evaluate two values against each other in some way.
- In this instance we are evaluating our *hitpoints* variable value against that of 0.

```
if (iChosenDirection == 1)
```

```
if( hitpoints == 0 )
```

x == *y*

is the value of *x*

EQUAL TO the value of *y*

Logical Operators – More/Less Than

```
if( ammo_level > 0 )
```

$x > y$

is the value of x
MORE THAN the value of y

```
if( ammo_level < MAX_AMMO )
```

$x < y$

is the value of x
LESS THAN the value of y

- There are other operators which allow us to check if one value is more or less than another.
- Here we are evaluating whether the variable *ammo_level* is **more than (>)** zero.
- In this example we are again checking the *ammo_level* variable, however now we are checking whether it is **less than (<)** the maximum ammo value.

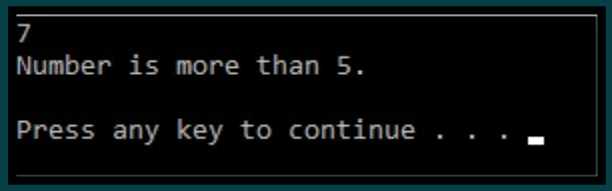
Task #1 – More or Less than 5

Using an **if statement** write a program that allows the user to input a whole number between 0 and 10.

Based on the value input, the program will then output one of the following results:

- *"The number is more than 5."*
- *"The number is less than 5."*
- *"The number is equal to 5."*
- *"The number entered is invalid."*

Example Output



Multiple Operators and Conditions

- if statements can contain multiple conditions.
- To do this we need to use additional logical operators to connect our conditions.
- Here we use a logical **AND operator** (&&) to check if a variable is more than 5 **and** less than 10.
- Similarly we can use an OR operator to check if one condition or the other is true.
- By combining conditions together we can solve fairly complex compound problems.

```
if(numberEntered > 5 && numberEntered < 10)
{
    // This number is between 5 and 10.
}
```

```
if(numberEntered == 0 || numberEntered > 10)
{
    // This number is either 0 or above 10
    // so is invalid
}
```

Task #2 – Grading Program

Using what you have learnt so far write a program that allows the user to enter the grade scored in a programming class (0 to 100).

Based on the score entered notify the user what grade they achieved based on the following:

- Above 80 is an **“A”**
- 70 to 79 is a **“B”**
- 60 to 69 is a **“C”**
- 50 to 59 is a **“D”**
- 40 to 49 is an **“E”**
- 1 to 39 is an **“F”**
- 0 is a **“U”**

Hint

You’ll need to use the logical operators you have learnt so far and multiple if conditions to build this program.

x > *y*

is the value of *x* **MORE THAN** the value of *y*

```
if( ammo_level > 0 )
```

x == *y*

is the value of *x* **EQUAL TO** the value of *y*

```
if( hitpoints == 0 )
```

x < *y*

is the value of *x* **LESS THAN** the value of *y*

```
if( ammo_level < MAX_AMMO )
```

Modulus Operator

$$3 \div 2 = 1.5$$

$$3 \,\%\, 2 = 5$$

$$10 \div 2 = 5$$

$$10 \,\%\, 2 = 0$$

```
if(numberEntered % 2 == 0)
{
    // This number is even
}
```

- The modulus (or 'remainder') operator divides two values and returns us the remainder.
- For instance 3 divided by 2 is 1.5.
- Using modulus 2 on an even number will always produce a value of 0.
- Knowing this, we can write an if statement to check whether a number is even or odd.

Task #3 – Odd/Even/Invalid

Using an **if statement** write a program that allows the user to input a whole number.

Based on the value input, the program will then output one of the following results:

- *"The number entered is even."*
- *"The number entered is odd."*
- *"The number entered is 0."*
- *"The number entered is invalid."*

The “**switch**” statement

- Again after our keyword ‘switch’ we have an expression
- However rather than a condition that evaluates to true/false, we enter a value
- This value is then compared against each ‘case’ in the statement
- Once it finds a matching case it will execute the logic
- We can add as many cases as required
- You should always end a switch with a ‘default’ block, similar to ‘else’ this is a catch-all in case none of the cases are met

```
switch( expression )
{
    case value:
    {
        ..logic
        break;
    }

}
```

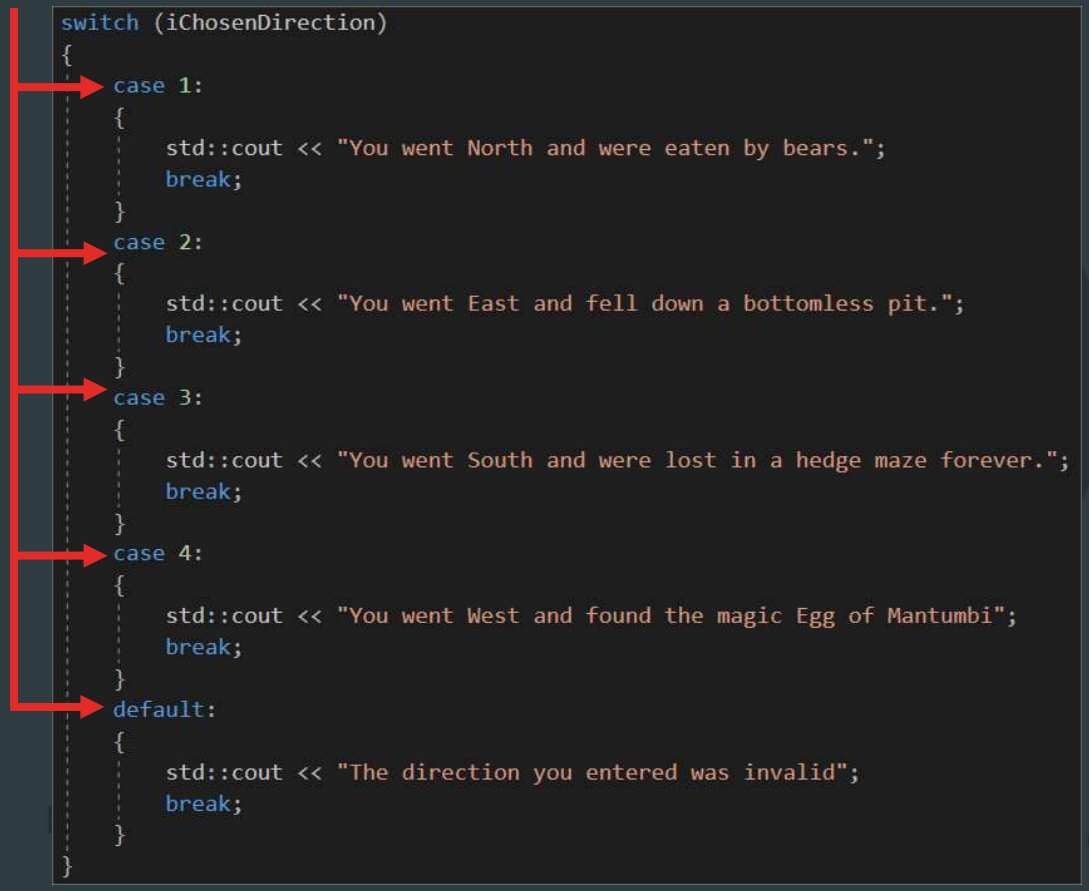
```
switch (iNumber)
{
    case 1:
    {
        std::cout << "iNumber equals 1" ;
        break;
    }
    case 2:
    {
        std::cout << "iNumber equals 2";
        break;
    }
    default:
    {
        std::cout << "iNumber is neither?!";
        break;
    }
}
```

break; break; break; break; break; break; break;

```
switch (iNumber)
{
    case 1:
    {
        std::cout << "iNumber equals 1" ;
        break;
    }
    case 2:
    {
        std::cout << "iNumber equals 2";
        break;
    }
    default:
    {
        std::cout << "iNumber is neither?!";
        break;
    }
}
```

- Make sure you add keyword **break** to the end of every case
- Keyword **break** signals the control flow to exit the whole switch
- Without this keyword we get what is known as **fallthrough**
- **Fallthrough** happens when no **break** is present and results in multiple cases being executed
- Very occasionally we actually want this to happen (known as **intentional fallthrough**) but the rest of the time it will probably create errors

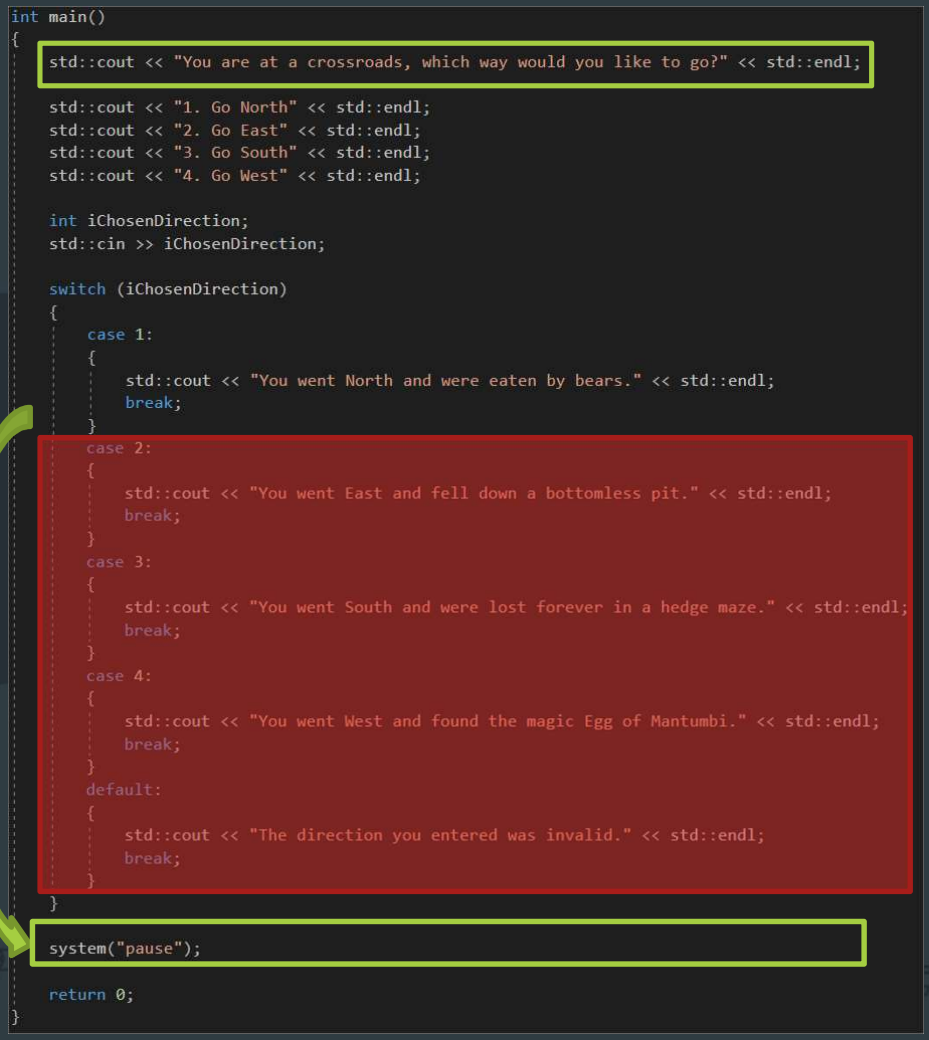
Back in our game context



```
switch (iChosenDirection)
{
    case 1:
    {
        std::cout << "You went North and were eaten by bears.";
        break;
    }
    case 2:
    {
        std::cout << "You went East and fell down a bottomless pit.";
        break;
    }
    case 3:
    {
        std::cout << "You went South and were lost in a hedge maze forever.";
        break;
    }
    case 4:
    {
        std::cout << "You went West and found the magic Egg of Mantumbi";
        break;
    }
    default:
    {
        std::cout << "The direction you entered was invalid";
        break;
    }
}
```

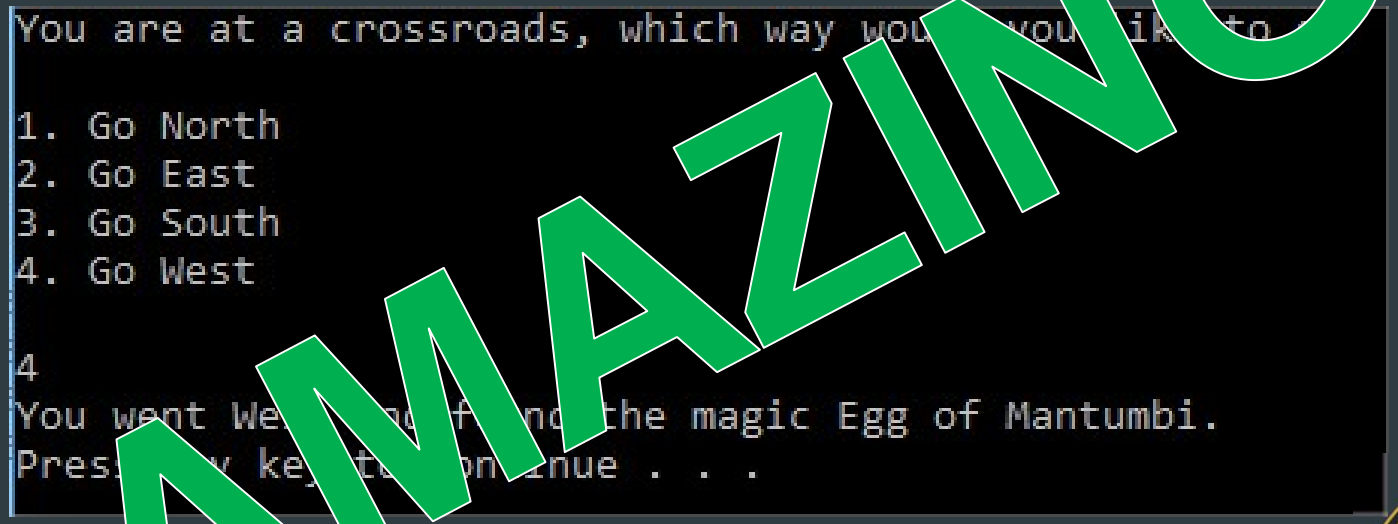
Final Product

- So let's go back to our text adventure
- Print all the initial text and wait for the player's input
- So let's say the player enters "1" this time
- The flow control will check each case until we find one matching the input value
- In this case the very first case matches the value we are looking for so we execute the logic inside the first case
- Once we hit the "break" keyword the flow exits the entire switch statement, so unlike an if statement we have to break out manually
- Again that means none of the code highlighted red was executed this time around



```
int main()
{
    std::cout << "You are at a crossroads, which way would you like to go?" << std::endl;

    std::cout << "1. Go North" << std::endl;
    std::cout << "2. Go East" << std::endl;
    std::cout << "3. Go South" << std::endl;
    std::cout << "4. Go West" << std::endl;

    int iChosenDirection;
    std::cin >> iChosenDirection;

    switch (iChosenDirection)
    {
        case 1:
        {
            std::cout << "You went North and were eaten by bears." << std::endl;
            break;
        }
        case 2:
        {
            std::cout << "You went East and fell down a bottomless pit." << std::endl;
            break;
        }
        case 3:
        {
            std::cout << "You went South and were lost forever in a hedge maze." << std::endl;
            break;
        }
        case 4:
        {
            std::cout << "You went West and found the magic Egg of Mantumbi." << std::endl;
            break;
        }
        default:
        {
            std::cout << "The direction you entered was invalid." << std::endl;
            break;
        }
    }

    system("pause");

    return 0;
}
```

Output



AMAZING

Task #4 – Cola Machine

Using an **switch statement** write a program that presents the user with a choice of 5 beverages (e.g. Coke, Water etc.) then allows the user to make a choice by entering a number 1 to 5.

Output the chosen beverage to the screen:

Example output:

- “You chose: Coke.”
- “You chose : Water.”

When to use **if** and **switch**

if

- Can express complex logic which resolves to a boolean.
- Can get messy and prone to user error.
- Offer much more flexibility but can't be as optimised by the compiler.

switch

- Compares against a single value.
- Enforce a strict structure for checking values.
- Generally faster once compiled than the equivalent **if**.

Syntax and Examples

```
if( ammo_level > 0 )
```

x > *y*

is the value of *x*
MORE THAN the value of *y*

```
if( hitpoints == 0 )
```

x == *y*

is the value of *x*
EQUAL TO the value of *y*

```
if( ammo_level < MAX_AMMO )
```

x < *y*

is the value of *x*
LESS THAN the value of *y*

```
if( num_of_lives >= 1 )
```

x >= *y*

is the value of *x*
MORE THAN OR EQUAL TO
the value of *y*

```
if( temperature <= 0 )
```

x <= *y*

is the value of *x*
LESS THAN OR EQUAL TO
the value of *y*

```
if( bomb_state != defused )
```

x != *y*

is the value of *x*
NOT EQUAL TO the value of *y*

```
if( !game_over )
```

!*x*

inverts the Boolean value
of *x*

```
if( game_over && hardcore_mode_enabled )
```

x && *y*

do both *x* **AND** *y* evaluate
to true

```
if( is_stunned || is_blinded )
```

x || *y*

does either *x* **OR** *y*
evaluate to true

Logical Operators

Operator	Function	Example
==	Equal to	lhs == rhs
!	Logical NOT	!conditional
!=	NOT equal to	lhs != rhs
<	Less than	lhs < rhs
<=	Less than or equal to	lhs <= rhs
>	Greater than	lhs > rhs
>=	Greater than or equal to	lhs >
&&	Logical AND	condition1 && condition2
||	Logical OR	condition1 || condition2

a	b	a && b	a || b	!(a || b)
false	false	false	false	True
false	true	false	true	false
true	false	false	true	false
true	true	true	true	false

Task #5 Source Control (again)

- Just like last week, please upload your work to GitHub.
- Use the GitHub Classroom link under this weeks Moodle topic.
- This will setup a new repo for you to upload this weeks tasks.