

**C++**

# *1. Triple X Game Design Doc*





# Triple X Game Design Doc (GDD)

- Simple number puzzle game
- Hacking into a computer lock
- Need to find a valid combination of codes
- Starts pretty easy, becomes much harder
- Wrap in your own story.



# Why Triple-X Game?

- Learn basics of C++ “syntax”
- Learn about variables and constants
- Create your own function with an argument
- Use `#include` ready for Unreal
- Learn the `if` and `while` structures



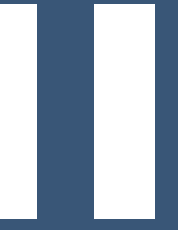
# Why Learn C++ Programming?

- All roads lead to C++ in game development
- You'll be learning a superpower
- It's both high-level, and low-level
- Join a unique tribe of power users
- It's fun and challenging



# What's Your Story

- Try and think of a unique story
- If you're stuck, ask a kid!
- Enjoy sharing





# the structure of c++ by example

- `#include` statements at top, before used
- `PlayGameAtDifficulty()` before used
- Everything inside a function indented
- `while` and `if` statements extra indented
- Colours differentiate types of text.



# Explain Our `main()` Function

- Explain our `main()` function best you can
- Don't worry how much / little you know
- The purpose is twofold
- Firstly: to be brave, it counts for a lot
- Secondly: to record your starting point



***C++***



## ***2. The Structure Of C++ By Example***



```
int main()  
{  
    return 0;  
}
```





# Why `int main()`?

- Every C++ program requires `int main()`
- It is the entry point of a c++ program
- Without it, your program will not build!



# White Space

- Compiler ignores white space!
- Don't be reckless with your code!
- Make the effort to make it beautiful!



# Your first program!

```
int main()  
{  
    return 0;  
}
```

- It runs and then exits immediately!



# White Space

- Write your program onto one line
- Try compiling
- Does it compile?



# int main() & White Space

- int main() required in every c++ program
- Without int main() a c++ program will not run!
- We **return 0** to signal program has run successfully
- Compiler ignores white space
- Code style improves readability!



**C++**

### ***3. Your First Program***





# Preprocessor Directive

(Don't be afraid!)

- Instruction to the compiler
- Used to include a library before compiling
- We want it to be at the start of our file
- # represents preprocessor directive in our code





**#include <iostream>**

**Preprocessor Directive**

**Header File**

- **Instructing the compiler to copy the contents of the iostream header file into our code before the rest of our code is compiled**



# Include the header file!

- Add: `#include <iostream>` to the first line of your file



**C++**

***4. Hello, World!***





```
std::cout << "Hello, World!";
```

Namespace

Scope Operator

String

Cout defined in std namespace





**John Doe**



**John Roe**

**We can tell the difference in the name!**



**namespace::myfunction**

**namespace2::myfunction**

**Same name!**

**Namespace avoids the conflict of code being the same name!**



# TripleX Intro

```
std::cout << "You are a secret agent breaking into a secure server room";
```

```
std::cout << "You need to enter the correct codes to continue...";
```

- Create your own intro messages for your game across two lines of code
- Save, Compile & Run program!



# Hello World!

- `#` represents a preprocessor directive
- `<iostream>` is a header file
- `cout` is defined in `std` namespace
- “Strings are wrapped in quotation marks”
- `std::cout << std::endl;` for new line



**C++**

## ***5. Variables***





# Variables

- Used to store data
- To use variables in c++ you first must declare them in your code
- By declaring a variable you are reserving space in the computer's memory for it



# Variable Declaration

```
int a = 0;
```

Data Type

Variable Name

By giving a value to a variable when it is declared, you are initializing it!



# Variable Declaration

`int a = 0;` ← Initialized

`int a;` ← Uninitialized





# Print your the value of your variable

- Print your variable onto a new line



# TripleX Codes

You're a secret agent breaking into LEVEL 2 server room  
Your SuperHacker 2000 tells you the following information...

- + There are three numbers in the code
- + The codes multiply to give 18
- + The codes add-up to 8

Enter the three code numbers followed by x





# Declare two more variables!

- **Declare an integer named b.**
  - **Initialize it**
- **Declare an integer named c.**
  - **Initialize it**



# Arithmetic Operators

Addition Operator =  $+$

Subtraction Operator =  $-$

Multiplication Operator =  $*$

Division Operator =  $/$



# Multiply your variables!

- Declare the product of a, b & c
- Print your product variable on a new line



# Variables

- You declare variables in code
- A variable with a given value is initialized
- Declaring a variable = Reserved space in memory
- You can add/multiply with arithmetic operators
- Print to the terminal like: `std::cout << a;`



**C++**

## ***6. const & Assigning Values***





# Variables

- Variables can be changed at runtime
- After a variable has been declared, we can assign a new value to it at any point in our code
- *“A value that is subject to change”*



# Assigning values to variables

- We never want to assign values to variables before the variable has been declared
- The compiler will not allow it.
- The compiler will not be aware that the variable exists



# Try this...

- Prefix all your variable declarations with: **const**

```
const int a = 4;
```

- Be brave and post your thoughts in the community!
  - How do you think this affects our code?



# Assigning values to variables

- We cannot assign a value to a variable before the variable has been declared
- The compiler will not be aware that the variable exists if you try assigning to it before it's declaration
- The compiler will not allow this!



# const

(Short for constant)

- With the **const** keyword, we signal our intentions to ourselves and to anybody who reads our code:
- “We never want this variable to change it’s value after it’s declaration!”
- Compiler = best friend! It protects our code!



# **const** & assigning values

(Short for constant)

- Assignment Operator =
- We can assign values to variables like: a = 4;
- Cannot assign before a variable has been declared!
- **const** keyword to mark your variables as constant
- The compiler will protect **const** variables!



**C++**

## ***7. Statements & Comments***





# Expression Statements

```
std::cout << "Hello, World!";
```

```
myvariable = 5;
```

**An expression followed by a semicolon is an expression statement!**



# Explain these! Post in the community forum!

TripleX.cpp

```
1 #include <iostream>
```

Preprocessor Directive

```
3 int main()
```

Main Function

```
4 {  
5     std::cout << "You are a secret agent breaking into a secure server room...";  
6     std::cout << std::endl;  
7     std::cout << "Enter the correct code to continue...";  
8 }
```

Expression Statements

```
10 const int a = 4;  
11 const int b = 3;  
12 const int c = 2;  
  
14 const int sum = a + b + c;  
15 const int product = a * b * c;
```

Declaration Statements

```
18 std::cout << std::endl;  
19 std::cout << sum << std::endl;  
20 std::cout << product << std::endl;
```

Expression Statements

```
22 return 0;
```

Return Statement

```
23 }
```



# Comments

- Commenting code can be used to make your code more easily understood by yourself or others!

// This is a single line comment!



# Comment your code!

- **// We use a double forward slash to comment**
- Add a comment to the start of your main function
- What do the first few lines of your main function do in the vision of your game?
- Comment wisely!



# Statements & Comments

- Expression statements:
  - Expressions that end with a semicolon
- Declaration statements:
  - Statements where we declare 'something'
- Comment like: `// This is a single line comment`



**C++**

## ***8. Naming & Self Documenting Code***





# Your turn!

- Print a message about the product
  - “The codes multiply to give: ”
- Take a screenshot of your terminal to showcase your TripleX game in the community!



# Variable Naming

- Important to give your variables good names!
- Must begin with a letter or an underscore
  - Cannot start with a number!
- You must not use a reserved keyword



# Keywords

- Reserved keywords are to be used by the C++ language only
- Over 90 keywords

## Examples:

<b>auto</b>	<b>bool</b>	<b>break</b>	<b>case</b>
<b>const</b>	<b>int</b>	<b>friend</b>	<b>return</b>



# Self Documenting Code

- Makes our code easier to read and understand
- If we give a variable a good name, we don't have to use a comment to explain what it does!



# Unreal Coding Standard

- Unreal Engine 4 uses an UpperCamelCase naming convention

**myVariable**

**MyVariable**



The start of each word is capitalized!



# Rename your variables!

- Rename your a, b & c variables:
  - CodeA, CodeB, CodeC
- Rename sum and product:
  - CodeSum, CodeProduct
  - Make sure they are initialized correctly!
- Follow Unreal Engine 4's naming convention



# Naming & Self Documenting Code

- Self document code with good naming!
- Must begin with a letter or an underscore
- You must not use a reserved keyword
- Unreal's naming convention UpperCamelCase
- Right click and “change all occurrences” in VS code to rename and replace variable everywhere in file



**C++**

## ***9. Getting User Input***





```
std::cout << "Hello, World!";
```



Character Output



Insertion Operator



```
std::cin >> PlayerGuess;
```

Character Input

Extraction Operator

Extract from input  
stream and assign to  
variable: PlayerGuess



# Store the players guess!

- Make use of `std::cin`
- Store the `PlayersGuess`
- Print a message and the value of `PlayerGuess`
  - Example: “You entered: 512”



# TripleX as a Player

- As a player you have to guess a 3 number code
- 234 will be treated as “two hundred and thirty four”
- What if the code was multi digit?
  - Example: 8 12 24



## Answers

CodeA

CodeB

CodeC

## Guesses

GuessA

GuessB

GuessC



2 3 4

`std::cin >> GuessA;`    `std::cin >> GuessB;`    `std::cin >> GuessC;`

**Your program will only ask for more input when cin is called  
IF the input stream is empty!**



# What's relevant?

- We want the players to enter numbers only
- They can enter numbers on seperate lines
- Or the same line separated by spaces
- We need to add a fix later! Incase anything other than a number gets entered & for replayability



# Declare Guess Sum & Products

- Declare GuessSum
  - Initialize it by adding GuessA, GuessB & GuessC
- Declare GuessProduct
  - Initialize it by multiplying GuessA, GuessB & Guess C



# Getting User Input

- cout = Character Output
  - *Insertion Operator* = <<
- cin = Character Input
  - *Extraction Operator* = >>
- Characters like 'x' get converted to 0 with integers
- Chars like 'x' will halt cin from working until reset



**C++**

## ***10. Using if and else in C++***





# if statements

```
if (condition)
```

```
{
```

```
}
```



# if statements

```
if (condition)
{
    std::cout << “you win!”;
}
```



# if statements

```
if (true)
{
    std::cout << "you win!";
}
```



# if statements

```
if (false)
{
    std::cout << “you win!”;
}
```



# Implement your if statement

- Implement an if statement
- Enter **true** as the condition
- Add a code block that prints a win message
- Play TripleX and see what happens if you win
- Play again and see what happens if you lose



# if statements

```
if (condition)
{
    std::cout << "you win!";
}
```



```
if (GuessSum == CodeSum && GuessProduct == CodeProduct)
{
    std::cout << "You win!";
}
else
{
    std::cout << "You lose!";
}
```



# You win...you lose...

- Print a win message if the player enters the correct code
- Print a lose game message if the player enters an incorrect code
- Share your work so far in the community!



# Using **if** and **else** in C++

- == Equality operator
- && Logical “*and*” operator
- **if** (condition)
  - Executes the code block below if condition is met
- **else**
  - Executes code block below if condition is not met



**C++**

## ***11. Functions***





```
std::cout << "Hello, World!" << std::endl;
```

**Not the only way to end the line!**



# Escape sequences

- Used to represent different characters within a string
- Escape sequences modify the format of a string





He said: “Hello, World!”



```
std::cout << “He said: ”Hello, World!”;
```

```
std::cout << “He said: \“Hello, World!\””;
```

Double quote escape sequence



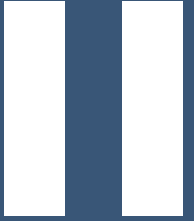
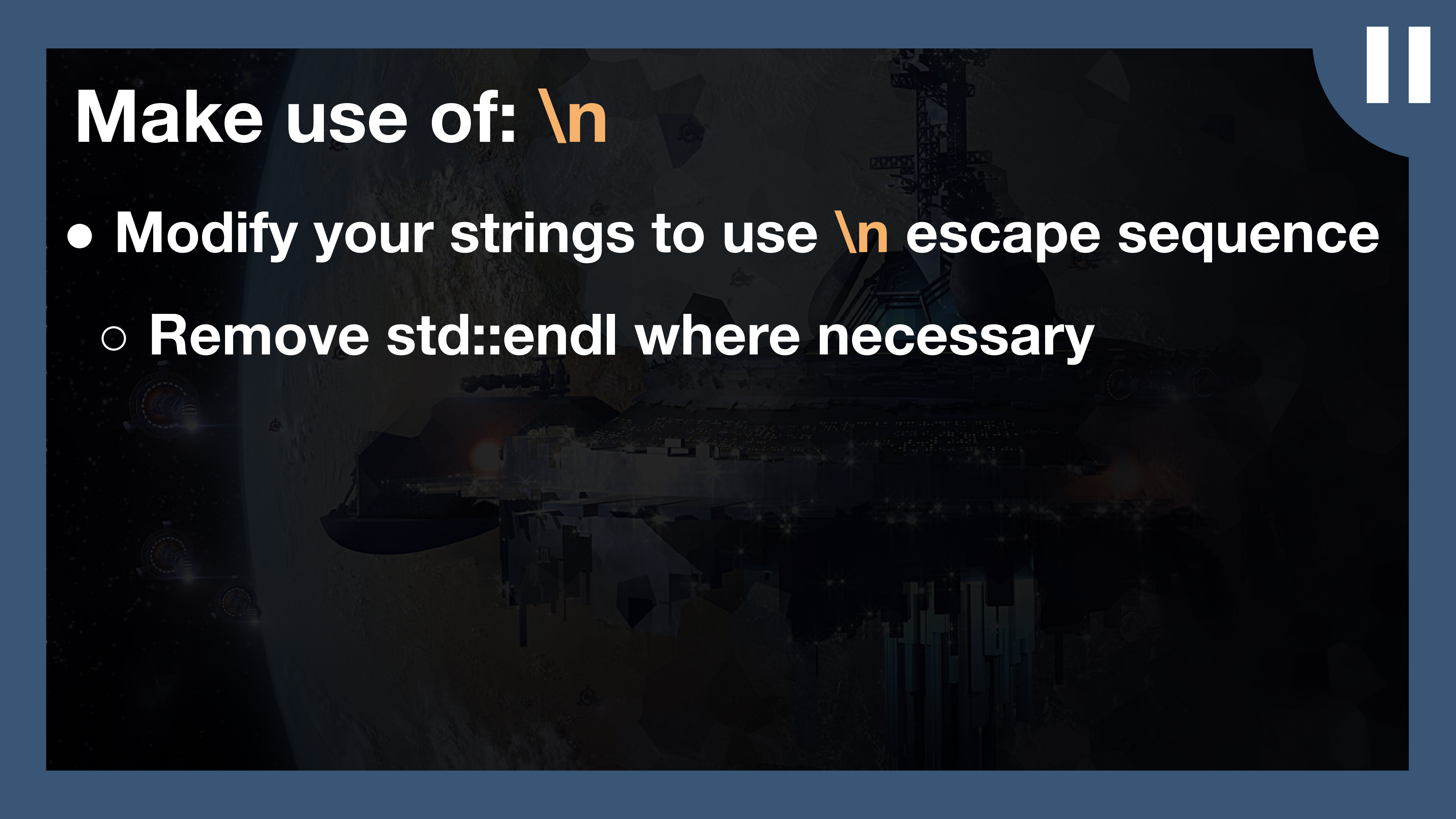
```
std::cout << "Hello, World!" << std::endl;
```

```
std::cout << "Hello, World!\n";
```



**New line  
(Line feed)  
Escape sequence**





# Make use of: `\n`

- Modify your strings to use `\n` escape sequence
  - Remove `std::endl` where necessary



# Function

- A function is a group of statements that together, performs a task and is given a name
- Every C++ program must have at least one function
  - **main()**





```
int main()  
{  
    std::cout << "Hello, World!";  
    return 0;  
}
```



# Let's create a function!

- Our function will return no value
- The name of our function is going to be PlayGame
- The function will contain no parameters
- The function body will contain the code for playing our game



# Your first function!

- Return type: **void**
- Name the function **PrintIntroduction**
- **PrintIntroduction()** at the start of **PlayGame()**
- Create some cool ASCII art to share in the community!



# Functions

- `\n` new line escape sequence in strings
- A function is a group of statements that is named and performs a task
- You must specify the return type of a function
- `void` to return no data
- Must be implemented before it is executed!



**C++**

## *12. Returning Data From Functions*





# while loop

```
while (condition)
```

```
{
```

```
    PlayGame();
```

```
}
```

***Code block executes repeatedly until the condition is not met***





**while** you're here...

- Pause the lesson and implement a **while** loop into your main function
- Hard code the condition as **true**
- Add a code block for the while loop to execute **PlayGame()**



# Fix the issue!

- After `PlayGame()` in your **while** loop, add these lines:
- `std::cin.clear();` **// Clears any errors**
- `std::cin.ignore();` **// Discards the buffer**
- Want to discuss in detail? Post in the community and chat with your fellow students!



# Has the player completed the level?

- Currently PlayGame returns no data
  - It's data type is **void**
- We want to return if the player has completed the current level
- To do this, we are going to work with Booleans



# Boolean

- A boolean is a value that is **true** or **false**





# **bool** PlayGame()

- Change the data type for PlayGame from **void** to **bool**
- Compile your code and see what happens...



# Initializing bLevelComplete

- `bLevelComplete = PlayGame();`



**Initialize bLevelComplete with the  
result of PlayGame**



# Returning Data From Functions

- **while** loop to execute a code block repeatedly
- CTRL + C to exit program
- Functions that are not of type **void** must contain a return statement for all exit paths of the function
- **bool** = true or false value



**C++**

## ***13. Function Parameters***





# **++LevelDifficulty**

**LevelDifficulty + 1**

**4 + 1 = 5**



# Variable Scope

- A variable declared inside a code block has scope
- It is local to where it is declared
- Cannot be accessed or used outside of its 'scope'



# Create a parameter for PlayGame()

- Pause here and create a parameter for PlayGame()
- The data type should be `int`
- The parameter should be named `Difficulty`



# Function Parameters

- Increment variables like: `++MyVariable;`
- Think about where you declare your variables
  - Because of the variables scope!
- Function parameters allow us to pass values into it
- Passing values is known as passing an argument



**C++**

## ***14. Comparing Values***





# Declare a new variable!

- Declare a variable for storing the maximum difficulty/level
  - MaxLevel or MaxDifficulty
- Make it a constant by declaring with the keyword **const**
- Initialize it with the amount of levels you want in your game
- In scope within our **main** function
  - Declare it above the **while** loop



# while loop condition

- **LevelDifficulty** should never become larger than **MaxDifficulty**
- Our **while** loop condition should check to see if the value of the **LevelDifficulty** is lower or equal to **MaxDifficulty**



# Comparison operators

Equal to	<b>==</b>
Not equal to	<b>!=</b>
Greater than	<b>&gt;</b>
Less than	<b>&lt;</b>
Greater than or equal to	<b>&gt;=</b>
Less than or equal to	<b>&lt;=</b>



# Update your strings to reflect the game!

- When our PlayGame function returns **true** or **false**
- Update the strings so they reflect:
  - Moving to the next level if correct code is entered
  - Retrying the level if the player enters the wrong code
- Before we **return 0;** in **main** function:
  - Print a string to the terminal that congratulates the player on completeing the game



# Working with random numbers

- We are going to initialize our variables with the function that is named: **rand()**
  - This function returns a random number



# rand()

- **rand()** returns a value in the range 0 and a value that is guaranteed to be at least **32,767!**
- We need to have more control over the range!
- We need **rand()** to take our LevelDifficulty variable into account



# Comparison Operators

- Compare values with:
- Can be used for conditions
- Like in: **while** and **if**
- Make sure spacing is consistent!
- **rand()** returns a value in the range 0 and 32,767!

Equal to	<b>==</b>
Not equal to	<b>!=</b>
Greater than	<b>&gt;</b>
Less than	<b>&lt;</b>
Greater than or equal to	<b>&gt;=</b>
Less than or equal to	<b>&lt;=</b>



**C++**

## ***15. Generating Random Number Ranges***





# Modulus Operator

- %
- Performs a division but returns the remainder!
- $9 / 4 = 2 \text{ r } 1$
- $9 \% 4 = 1$
- We can use this to map `rand()` to our own range



# There's a pattern!

- $\langle \text{value} \rangle \% \langle \text{modulus} \rangle$
- Maps the  $\langle \text{value} \rangle$  between a range of numbers!
- The range is between 0 and  $\langle \text{modulus} \rangle - 1$



# Modulo Operation

- $2524 \% 10$ 
  - 2524 mapped to a range between 0 - 9
  - Not 0 - 10! The range is 0 to the modulus - 1
- The result is the same as the remainder we would get from dividing 2524 by 10!



# Modulo Operation

- $2524 \% 10 = 4$ 
  - 2524 mapped to a range between 0 - 9
  - Not 0 - 10! The range is 0 to the modulus - 1
- The result is the same as the remainder we would get from dividing 2524 by 10!

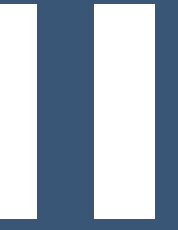


# Mapping rand() to a range

- `rand() % <modulus>`
  - Maps the random number in the range of 0 and the **modulus** value (subtracted by 1)



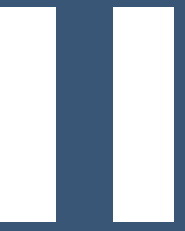
# Challenge





# Challenge

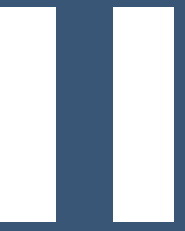
- Limit the range of `rand()` to initialize variables: **CodeA, CodeB & CodeC!**





# Challenge

- Limit the range of `rand()` to initialize variables: CodeA, CodeB & CodeC!
- Play the game!





Modulus Value

rand() % <Modulus>  
(Range)

1

0 - 0



Difficulty

rand() % Difficulty  
(Range)

1

0 - 0



Difficulty	rand() % Difficulty (Range)
1	0 - 0
2	0 - 1
3	0 - 2
4	0 - 3
5	0 - 4



Difficulty	rand() % Difficulty (Range)	rand() % Difficulty + 1 (Range)
1	0	1 - 1
2	0 - 1	1 - 2
3	0 - 2	1 - 3
4	0 - 3	1 - 4
5	0 - 4	1 - 5



# Challenge

- Initialize your code variables with: `rand() % Difficulty`
- Add 1 to `Difficulty` to offset the range
- Play the game!



Difficulty	rand() % Difficulty + Difficulty (Range)
1	1
2	2 - 3
3	3 - 5
4	4 - 7
5	5 - 9



# Not random yet!

- Our game isn't random yet!
- Each time we play it, it produces the same result!
- We have one more step...



# Seeding rand()

- We need to initialize rand() with a different seed
  - This will produce more random results
- The best way to do this is based on your computer's time
- `#include <ctime>`
- At the start of main add this line:
  - `srand(time(NULL));`
  - Creates new random sequence based on the time of day





# How far can you get?

- Play your game!
- What level can you get to?
- What level do you find really hard?
- Share in the community!



# Section 2 Ending

- Well done for completing section 2!
- You've built your first game in C++!
- Play an active part in the community!
  - Help out other students
  - Share your work



# Generating Random Number Ranges

- Modulus Operator: %
- Performs a division but returns remainder
- We can use % to control the range of rand()
- + 1 to offset the range
- Well done for completing section 2!
- Play an active part in the community!



**C++**

***Unreal C++ Projects for VSCode & VS2019***

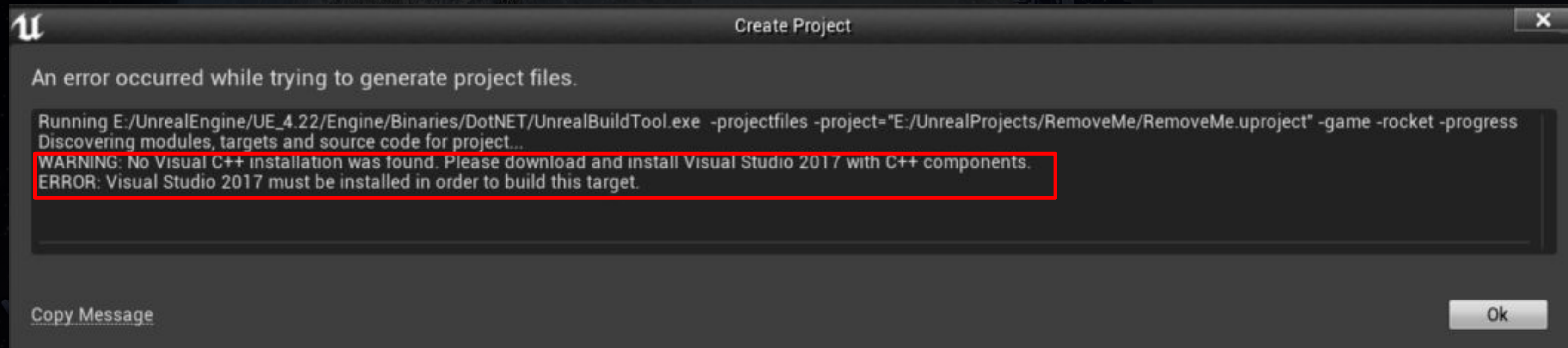




# Creating Unreal C++ Projects

- Unreal Engine 4 expects:
  - Visual Studio 2017 installed if running Windows
  - Xcode installed if running macOS
- By default, Unreal will not allow you to create C++ projects without having these installed





- What if we are running VS Code or Visual Studio 2019?



# Fix for VS Code & Visual Studio 2019

- Launch the Unreal Engine 4 editor
- Create a **Blueprint Project**
- Change the **source control editor** from Unreal's  
*Editor Preferences*



# Blueprint projects

- Blueprint projects can still be C++ projects!
- You just have to add a C++ class to your project



# C++ Projects for VSCode & VS2019

- To use VSCode or VS2019:
- Create a **Blueprint Project**
- Change the **source control editor** from Unreal's *Editor Preferences*
- Blueprint projects can still be C++ projects!
- You just have to add a C++ class to your project