Welcome to CT4019

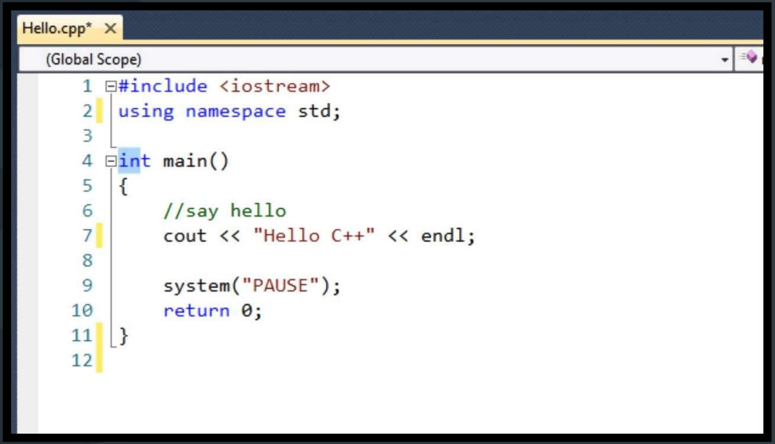
Programming and Mathematics for Games

Please sign the register on Moodle

JISC DataX
Attendance Logging

Programming and Mathematics

- This module assumes no prior experience in programming.
- But it moves, **quickly**.
- We will take you from the basics of C++ programming through to more advanced topics.
- We will also cover mathematical topics that are the cornerstone of 3D simulation and game development.
- The focus here is to give you the foundation you need to cope with the more complex coding modules at Levels 5 and 6.



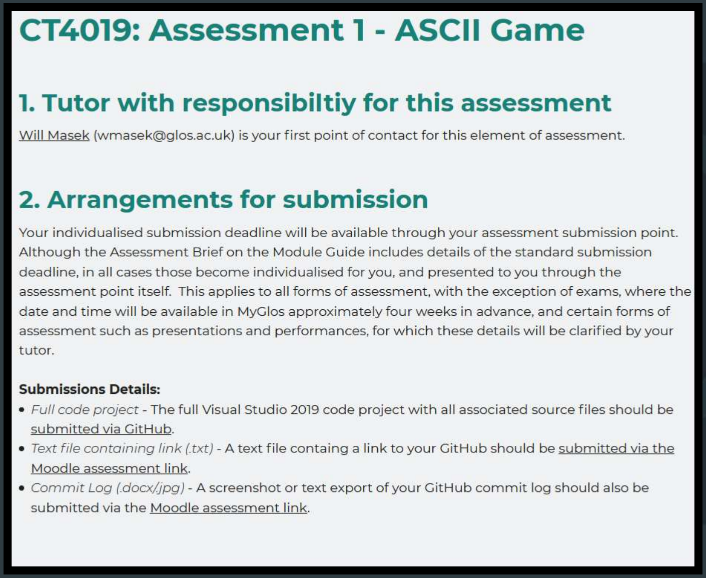
```
#include <iostream>
using namespace std;

int main()
{
    //say hello
    cout << "Hello C++" << endl;

    system("PAUSE");
    return 0;
}

```

Assessment


CT4019: Assessment 1 - ASCII Game

1. Tutor with responsibiltiy for this assessment

Will Masek (wmasek@glos.ac.uk) is your first point of contact for this element of assessment.

2. Arrangements for submission

Your individualised submission deadline will be available through your assessment submission point. Although the Assessment Brief on the Module Guide includes details of the standard submission deadline, in all cases those become individualised for you, and presented to you through the assessment point itself. This applies to all forms of assessment, with the exception of exams, where the date and time will be available in MyGlos approximately four weeks in advance, and certain forms of assessment such as presentations and performances, for which these details will be clarified by your tutor.

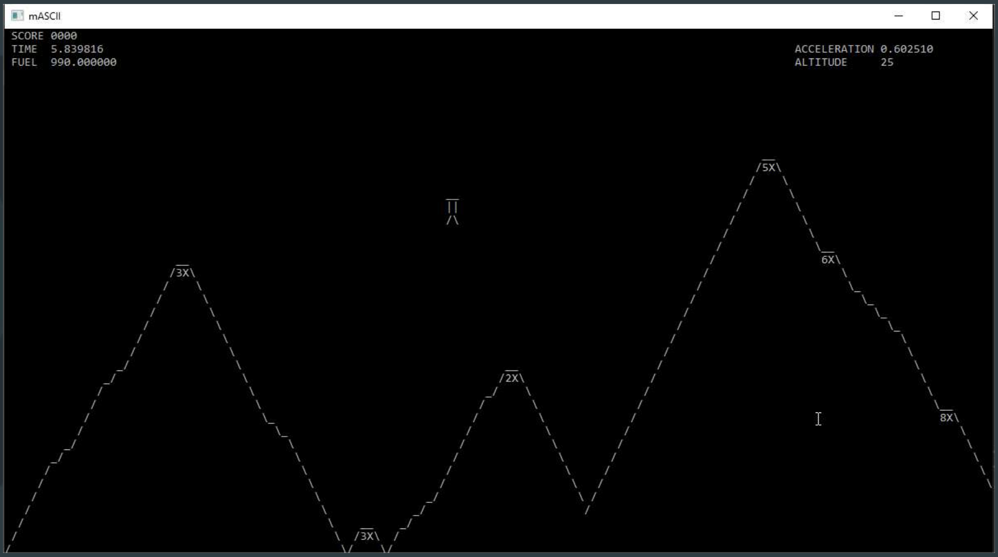
Submissions Details:

- *Full code project* - The full Visual Studio 2019 code project with all associated source files should be submitted via GitHub.
- *Text file containing link (.txt)* - A text file containg a link to your GitHub should be submitted via the Moodle assessment link.
- *Commit Log (docx/jpg)* - A screenshot or text export of your GitHub commit log should also be submitted via the Moodle assessment link.

- This module contains two assessment points:
  - One in Semester 1
  - Another in Semester 2
- Each of these assessments is worth 50% toward your final module grade.
- You must attain a minimum grade of 40% overall (across both assessments) to pass the module.

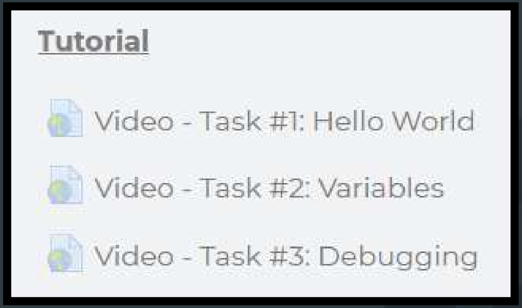
Assessment #1

- Students will be required to build an ASCII-based game engine using the Windows Console.
- Students will use this basic engine to build an approximation of the 1979 Atari Classic *Lunar Lander*.
- More details on this assessment will follow in future classes (and are also available through the assessment link on Moodle).

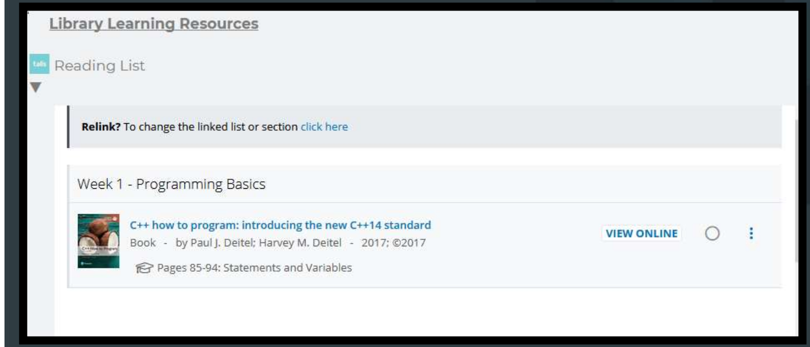


```
SCORE 0000
TIME  5.839816                                                                                  ACCELERATION 0.602510
FUEL  990.000000                                                                                ALTITUDE     25
```

Class Work


Tutorial

- Video - Task #1: Hello World
- Video - Task #2: Variables
- Video - Task #3: Debugging


Library Learning Resources

Reading List

Relink? To change the linked list or section click here

Week 1 - Programming Basics

C++ how to program: introducing the new C++14 standard
Book - by Paul J. Deitel; Harvey M. Deitel - 2017; ©2017
Pages 85-94: Statements and Variables

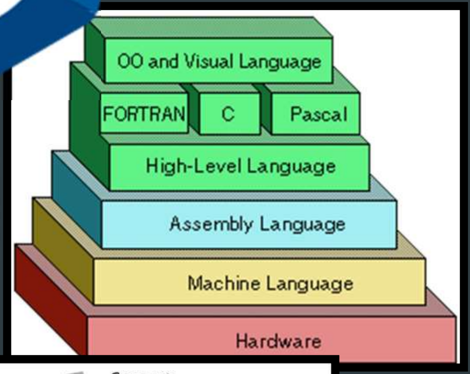
- In addition to the assignment you will be given tasks to complete each week.
- These tasks are not assessed.
- However they are essential to building your understanding so you have the skills to complete the assignment.
- You will also be assigned reading to do each week, these readings can be accessed via Moodle.

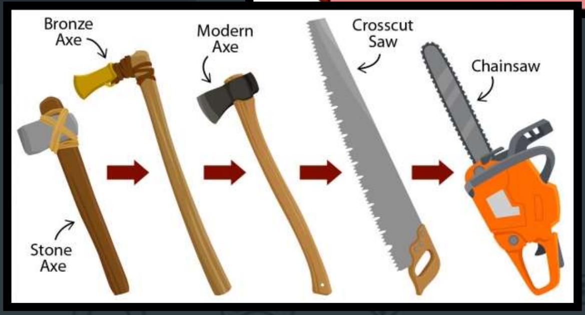
CT4019 PROGRAMMING AND MATHEMATICS FOR GAMES

Programming Basics

What is C++?

- C++ is a *High-Level* programming language.
- C programming goes back as far as 1972.
- C++ (or C with Classes) was an extension developed in 1979.
- Programming languages work in layers.
- Programming languages, like any tool, evolve over time.





Question

Why do you think game developers still use a 40 year old programming language?

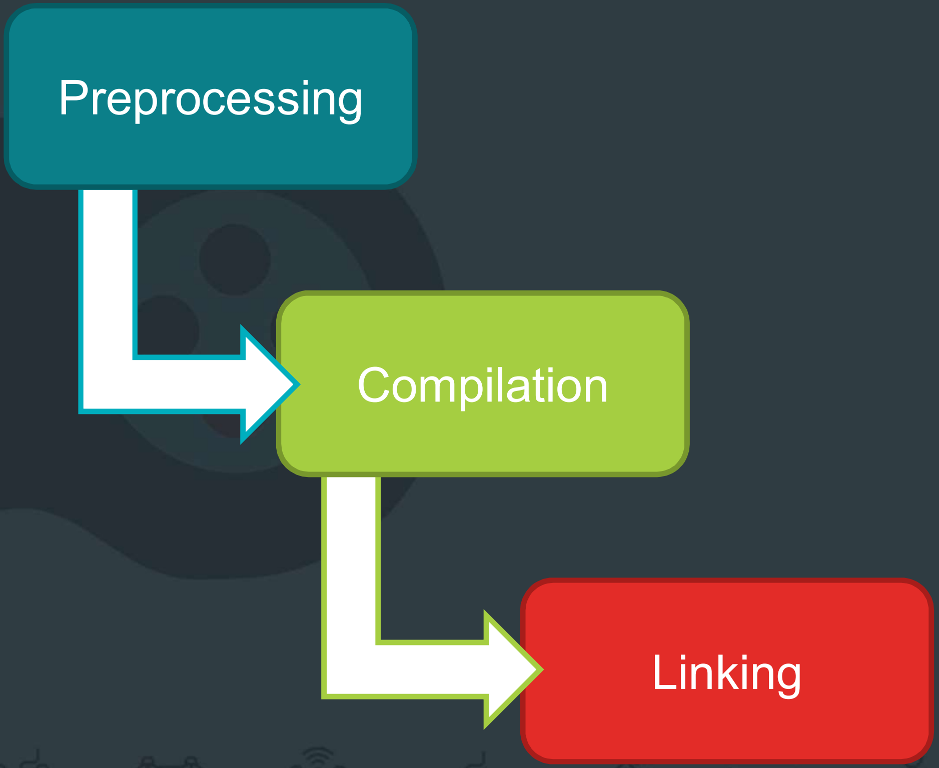
Visual Studio

- On this module, we are going to use Visual Studio 2019 to develop our C++ code
- Visual Studio is an “Integrated Development Environment” or IDE
- It consists of a text editor, with auto completion of code (intellisense), a debugger, and easy integration with compilation tools

C++ Compilation

Compilation of C++ code consists of three main stages:

1. Preprocessing
   - The pre-processor handles all “pre-processor directives” which are lines of code starting with #
   - These are simple tasks, such as including other C++ files (copy-pasting the contents of that file into the including file).
2. Compilation
   - Pre-processed C++ code is compiled to object files
3. Linking
   - All the object files that comprise your program are linked together, and the various functions within connected



Task #1 – Hello World

- Please start Visual Studio 2019, load the first video on Moodle in a browser, and follow along with that video to create a hello world program in C++
- See if you can make the program print “Hello [Your Name]” replacing [Your Name] with your name, instead of “Hello World!”
- When complete, save this project somewhere safe on the system.

Congratulations!

- You're now a C++ programmer!
- How long will it be until you're an expert C++ programmer?
  - Roughly ten years worth of programming regularly, or about 15,000 solid hours of effort is a common estimate
- How long until you know everything about C++?
  - Pretty much impossible – new things are being added fast enough and the existing language is broad enough that it is arguably impossible to remember everything about C++
- So what will we achieve on this module?
  - An introduction to C++, building you to the point where you can understand enough of the syntax and logic, you can produce 2D games by only inputting your time and effort

Breaking down our program

- #include <iostream>
  - This line of code includes the iostream library, from the C++ standard template library (stl), which contains the function we need to print to the console screen

- void main(int argc, char *argv[])
  - This creates a function called “main”, accepting two parameters. This is a standard entry point for a C++ program, so when we run our code, the operating system will look for this function to run it. The contents of the function are enclosed between curly braces ‘{‘ and ‘}’

Breaking down our program continued

- std::cout << “Hello World” << std::endl;
  - This line of code uses the iostream libraries “cout” function, which is within the “std” namespace (used for the standard template library, or stl). It passes it a string, which is enclosed between double-quotes, and the symbol for endl, which is also in the std namespace. “endl” finishes the line we’re printing and moves to the next line.
- system(“pause”);
  - Without this line, our program would complete, then disappear as it’s finished. This line tells C++ to wait until the user presses a key.
- Semi-colons, each “line” of code in C++ will normally end with either a semi-colon ; or a close-curly-brace } . Some keywords are exceptions.

Therefore…

- All we have to do to get a personal greeting, is to alter the string that is being passed to cout, as that is what is being printed to the screen
- std::cout << “Hello Will” << std::endl;

Variables

- With what we've learnt so far of C++, we can only use information that we have when we write our code – we have to hard-code values into our code to use.

- C++ allows us to store values for later use, these are called variables and come in two types, **primitive** and **complex**

- **Primitive** types have two sub-categories:
  1. Integral (whole number) variables
  2. Floating point (fractional number) variables

Integral types

- Can store a finite range of numbers
  - If you exceed this range, the numbers will “overflow”
  - I.e. an unsigned char type can store up to 255, meaning 256 values including 0. If you are at 255 and add one to this variable, it will return to 0
- Signed integers can store values that are both above and below 0
- Unsigned integers are always above zero

Floating point types

- Floating point variables contain both an integral portion (numerical value) and a fractional portion (where to place the decimal point)
- The processing of these numbers is more complex than that of integer types – usually it uses two’s complement
- The two parts of a floating point are called the “mantissa” for the integral component, and the “exponent” for the fractional component.

Value	Mantissa	Exponent
123.456	123456	3

Single vs double precision

- Floating point variables can be single precision (declared by a “float” variable type)
- Or double precision (declared by a “double” variable type)
- A single precision floating point variable is accurate to 7 decimal places.
- A double precision floating point variable is accurate to 15 decimal places.
- Why is this important?
- Most of the time, in games development, we use floats. It’s important to use the most suitable type, as lower precision means operations take less processing power and games are ALL floating point math

Task #2 – Hello World with variables

- Please start Visual Studio 2019, load the second video on Moodle in a browser, and follow along with that video to create a hello world program in C++ using variables
- Experiment with the variables we have created
- When complete, save this project somewhere safe on the system

Variable naming

- Variable names should be descriptive about the data that the variables contain
- Variable names should follow the coding guidelines, including being in camel case
- One-letter variable names are fine when testing something quickly, but should be refactored to something more descriptive if they enter permanent code
- Variable names should not contain any profanity or bad language – be professional
- Variable names cannot begin with a number, or contain spaces or punctuation

Variable naming

- Good variable names:
  - username
  - minimumNameLength
  - CharacterName

- Bad variable names:
  - a
  - data
  - un1
  - word1, word2, word3…
  - myString
  - username, userName, UserName

Complex variables

- Everything we’ve dealt with to now are termed “primitive” variables
- Complex variables are combinations of primitive variables (and sometimes functions)
- They may be classes, or structs, in C++ there is no difference except for the default access level (which we’ll cover later). Structs default to public, classes to private.

Debugging

- Visual studio has a massively powerful debugger built right into it
- There are three main types of error you will encounter in your code (along with hundreds of edge cases)
  - Compilation errors
  - Execution or run-time errors
  - Logical errors

Compilation errors

- These are the best kind of errors!
- The compiler toolchain will try to tell you what it thinks is wrong
  - These will be listed in the *output* or the *error-list* window
- Messages are usually not very accurate (in C++) but should give you some clues at least
- Fix one error first, then re-compile and investigate any remaining errors, even if there are several listed

Common causes of compilation errors

- Missing parenthesis, brackets, or semi-colons, give errors like:
  - `Error C2059: syntax error : ‘}’`
  - `Error C2046: illegal case`
  - `Error C2143: syntax error : missing ‘)’ before break`
- Missing includes:
  - `‘no such file or directory’`
  - `Undeclared identifier`
  - `X is not a member of…`
- Other common errors:
  - Using ‘=‘ when you meant ‘==‘
  - Misspelling variable names
  - Trying to use variables out of scope
  - Include errors

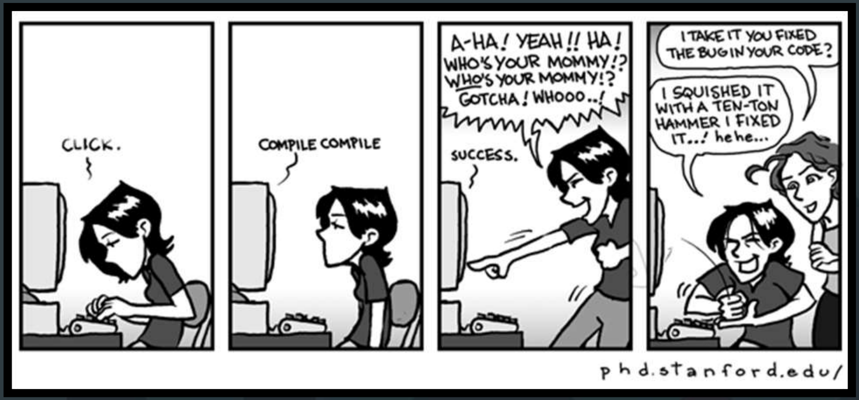
Execution errors

- The second best kind of errors (because at least you're sometimes aware there is an error)
- The code compiles okay, executes, but halts execution when it reaches a problematic section of code
- This may be every time it's run, or just sometimes!
- Commonly caused by memory issues such as:
  - Miscalculation (such as a divide by zero)
  - Accessing unreferenced memory (such as accessing item 100 in a 99 item array)

Logic errors

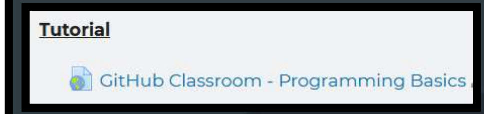
- The worst kind of error – sometimes you don't even know there is an error
- Your code compiles fine, runs fine, doesn't crash – but doesn't do what it's supposed to do in some way
- If you're lucky, it will be an obvious way. If you're unlucky it might be something not obvious at all, but which will cause issues later
- Common errors are:
  - Saving something incorrectly, causing issues on reload
  - Incorrect calculations, that complete successfully and return a number – the wrong one
  - Infinite loops

Task #3 Debugging

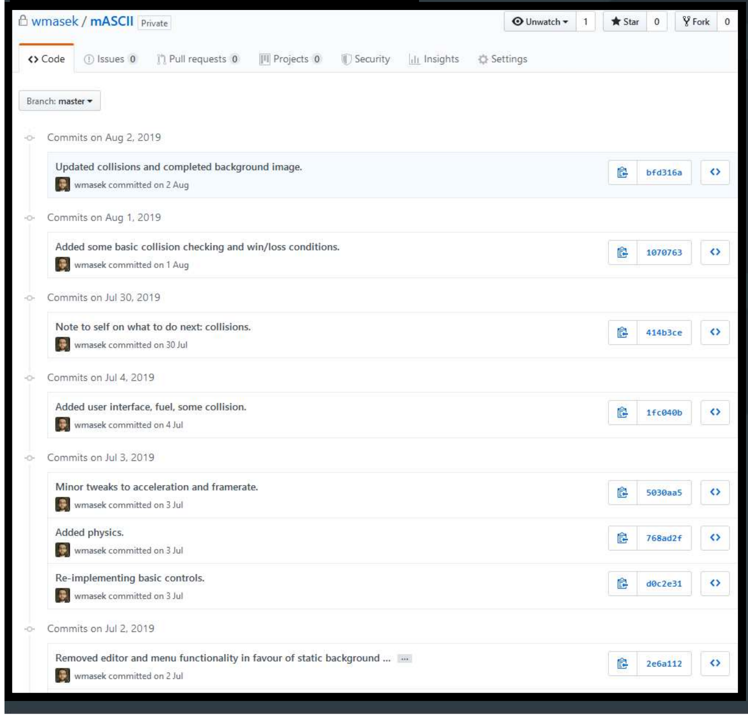

CLICK.
COMPILE COMPILE
A-HA! YEAH!! HA! WHO'S YOUR MOMMY!? WHO'S YOUR MOMMY!? GOTCHA! WHOOO..!
SUCCESS.
I TAKE IT YOU FIXED THE BUG IN YOUR CODE?
I SQUISHED IT WITH A TEN-TON HAMMER I FIXED IT..! hehe...
phd.stanford.edu/

- Please start Visual Studio 2019, load the third video on moodle in a browser, and follow along with that video to practice debugging a program in C++
- When complete, save this project somewhere safe on the system

Task #4 Source Control


Tutorial

GitHub Classroom - Programming Basics


wmasek / mASCII Private

Commits on Aug 2, 2019

Updated collisions and completed background image.
wmasek committed on 2 Aug — bfd316a

Commits on Aug 1, 2019

Added some basic collision checking and win/loss conditions.
wmasek committed on 1 Aug — 1070763

Commits on Jul 30, 2019

Note to self on what to do next: collisions.
wmasek committed on 30 Jul — 414b3ce

Commits on Jul 4, 2019

Added user interface, fuel, some collision.
wmasek committed on 4 Jul — 1fc040b

Commits on Jul 3, 2019

Minor tweaks to acceleration and framerate.
wmasek committed on 3 Jul — 5030aa5

Added physics.
wmasek committed on 3 Jul — 768ad2f

Re-implementing basic controls.
wmasek committed on 3 Jul — d0c2e31

Commits on Jul 2, 2019

Removed editor and menu functionality in favour of static background ...
wmasek committed on 2 Jul — 2e6a112

- Finally I want you to push all of today’s work to GitHub.
- You will **need** a GitHub account (www.github.com).
- Git is a Source Control application:
  - Git will keep every version of your code that you submit to the repository.
- This means if you break your code you can rollback to a previous version.
- It is **essential** you get to grips with using source control.
- Use the GitHub Classroom link on Moodle and upload your work to the Week 1. Tutorial.

Any Questions?