Stream 1: How to do C++ work

Students must complete this stream.

Description:
This stream of spikes is designed to familiarise you with the programing language and IDE you will be using for the rest of the semester.

Spikes:

*Spike 1: Simple Game Loop
*Spike 2: IDE Experience & Comparison
*Spike 3: Debugger Use
Spike 4: Advanced Visual Studio
Spike 5: C++11

Spikes marked with an asterix are core spikes

Spike 1: Simple Game Loop

CORE SPIKE

Context: Games are commonly driven by some form of game loop.

Knowledge/Skill Gap: The developer is not familiar with basic game loops.

Goals/Deliverables:
You need to create the "GridWorld" game, as described in a simple specification document available **below**. The game will demonstrate the use of a simple game loop, separation of update/render code, and game data (both for the world map and players current location).

You will need to deliver the following items:

1. A simple paper-based plan for your code design. (Yes, a simple functional design is fine – just as long as you can demonstrate that you planned first before coding!)
2. Create a simple console program that implements the "GridWorld" game using a simple game loop. The game must demonstrate the separation of:
    a. processing of input (text commands from the player),
    b. updating of a game model (where the player is and their options),
    c. display (output current location and options) of the game to the user.
3. Spike Outcome Report.
   Note: The Spike Outcome Report is always a required "deliverable". It will not be repeated in future spike requirements as it is assumed.

Recommendations:

- Read the "GridWorld" game details and on paper (as required in the deliverables) do a quick sketch/design of how you will organise your code. (No formal standard specified… just use something that would work to help you explain your design to another programmer)
- Look at the Spike Outcome Report template – note what you need to record for later.
- Use an IDE for C/C++ development that you already know. (We'll look at some others later so don't get distracted from the main point of this spike!)
- Begin small, test often. Use simple printouts to check values (or the debugger if you already know that and are comfortable). Don't try fancy debugging yet if you don't know it – that's a later spike if it's something you need to work out. Stay on target!

Tips:

- Consider a DEBUG macro condition if extra code is needed for testing purposes.
- You do not need to show the "map" or the current player location – but it is useful.
- If you find some useful resources that helped you to get your code working, then remember to note them down and include them in your spike outcome report. Google, books, blogs, classmates, etc. all go in the spike report.
- Have a plan for when you present your outcomes to the tutor. (They will probably say something like "Right – show me"… and it's up to you to show-off what you've done.)
- Many of these tips also apply to spikes in general and are not exclusive to this spike.

DO NOT

- Make things any more complicated than you need – just get it working.
- Create complex data-structures. A simple 2D array is fine (and expected).
- Add all the fancy features you can think of! Save ideas for later and perhaps develop them as a portfolio item. (You should still document/note them down though.)
- Load the map data from a file. Simply hard code the map data.

Spike ILOs

Design:	LOW	(1)
Implementation:	MEDIUM	(2)
Maintenance:	–	(0)
Performance:	–	(0)

Spike 2: IDE Experience & Comparison

CORE SPIKE

Context: There are many IDE tools available and developers need to be able to make informed decisions in order to be productive with the tools they select.

Knowledge/Skill Gap: The developer is not familiar with different IDE development tools.

Goals/Deliverables: [SPIKE REPORT] + [COMPARISON REPORT]

1. Using the simple command line program from Spike 1 create a working project using three of the following IDEs:
    a. Microsoft Visual Studio 2010/2012/2013 (any version)
    b. Code::Blocks
    c. Eclipse+CDT
    d. XCode
    e. Netbeans (for C/C++ development)
2. Create a short report to document your findings. (Note: the short report is NOT the spike report - you should have two reports for this spike.) Include the following details:
    a. For each IDE, a point-form list of the steps required to create a new project in the IDE, add new files, compile and run a command line program.
    b. For each IDE, a point-form list of the steps required to create a break-point location in the program, and run the program in using the IDE's integrated debugging system so that program execution stops at the nominated point.
    c. For each IDE, note how to inspect the value of variables during debugging.
    d. A comparison matrix (table) to represent the qualities of each IDE (you must decide what criteria you think is important) and your rating (value).
    e. Clearly state which IDE you will use in future and support your decision.

Recommendations:

- Start with the IDE that you know first (which you probably used for Spike 1). If you don't know any, we suggest Visual Studio or whatever your friends can help you with.
- Assume you are writing steps to help a colleague to create a project in the IDE.
- As a review, show your steps with some other students. Make sure your notes make enough sense! Update any suggestions or omissions so that your notes are valuable.
- Be sure to note the version numbers of software you use in your spike outcome report.
- Move on to the next IDE and repeat the steps, documenting as you go. (Try to complete all the deliverables for each IDE before moving on to the next.)
- Remember that spikes should be about the doing the minimum required to close the gap in knowledge or skill (or technology). However if you do not write enough so that someone else would understand, you have not complete to spike work correctly.

Note:

If you wish compare a different IDE that is not listed you MUST get permission from your tutor first. If you do not, your outcome report will NOT be accepted.

Spike ILOs

Design:	–	(0)
Implementation:	LOW	(1)
Maintenance:	LOW	(1)
Performance:	–	(0)

Spike 3: Debugger Use

CORE SPIKE

Context: Effective use of the debugger is essential for isolating and repairing code errors within a non-trivial project of source code.

Knowledge/Skill Gap: The developer is not familiar with the use of debugging tools.

Goals/Deliverables: [FIXED CODE] + [SPIKE REPORT]
You need to demonstrate that you have done the following:

1. Downloaded, compiled and run the "spike3" program provided on blackboard in the IDE of your choice. The program contains a number of "bugs", including a deliberate memory leak. (Memory is allocated, but not de-allocated.) You must discover and fix errors. (You might not find them all!)
2. Use IDE debugging tools to identify memory leaks and other issues. You must document in your spike report the IDE steps you used to do identify bugs. (You are welcome to use screen shots to supplement the written steps you have used. See the planning notes for suggested steps.)
3. Save the fixed code and included source code comments to document what you changed.

Recommendations:

1. Open the program in your IDE of choice provided it supports debugging and ideally "inspection". The subsequent steps may vary depending on your IDE, but should be similar.
    a. Add a break point at a point in the source code at a point likely to be the cause of the leak. (If you're unsure, place one close to the start of the application and step through all of it. This does mean more stepping, but at least you are less likely to skip over the broken code!)
    b. Compile and run the program in debug mode (or similar in your IDE).
    c. When the program breaks, step through the program, examining the source code and variables for potential causes of the leak. Note them down.
    d. Repair any leaks found. Add comments about your victory!

Note:

- You might not understand all the code provided, but that's okay! You should still be able to step through the program execution to find some bugs, if not all.
- The goal of this spike is not really about fixing the bugs/leak(s), but rather to get you familiar with the process of using the debugger, and to demonstrate its usefulness. Your spike report should not dwell on the nature of the bugs (or your corrections), but rather show that you can effectively use the debugger to solve problems (the spike "gap").

Spike ILOs

Design:	–	(0)
Implementation:	LOW	(1)
Maintenance:	LOW	(1)
Performance:	–	(0)

Spike 4: Advanced Visual Studio

OPTIONAL SPIKE

Context: There are many features of Visual Studio that can aid developers in developing & debugging their projects.

Knowledge/Skill Gap: The developer is not familiar with advanced tools and methods in Visual Studio.

Goals/Deliverables: [SPIKE REPORT] detailing the approach you took to using each of these features, including the resources you looked at

Investigate and document the following advanced features of Visual Studio:

- Multiple configurations (Development vs Release) and how they can be used
- Use of built-in pre-processor macros (ie _DEBUG)
- Adding your own pre-processor macros depending on build configuration
- How to launch the debugger with command-line parameters
- How to add additional include directories for a project
- The various optimization settings for a VS project
- How to link against libraries for compilation (through the IDE and the use of pre-processor macros)

Most of these features are available through Visual Studio's project properties dialogue.

Spike ILOs

Design:	–	(0)
Implementation:	MEDIUM	(2)
Maintenance:	MEDIUM	(2)
Performance:	–	(0)

Spike 5: C++11

OPTIONAL SPIKE

Context: C++11 is a new C++ standard that introduces new features

Knowledge/Skill Gap: The developer is not familiar with the new features C++11 has to offer

Goals/Deliverables:

You need to create a simple C++11 application that explores the new basic features of C++ and how they are beneficial in comparison to the C++03 methods.

You will need to deliver the following items:

1. A C++ application showing how to utilize the basic C++11 features
2. A comparison of these features to the C++03 counterparts (if applicable). This can be done in code and can be as simple as having the two methods in the same function to see the difference in code readability, usability etc.

For the purpose of this spike, the basic features of C++11 are considered to be:

- Nullptr keyword
- In-class member initialization
- Override and final keywords
- Auto keyword
- std::tuple
- Range-based for loop
- Non-member begin and end
- std::array

C++11 also offers many advanced features. You are encouraged to explore one of the advanced features of C++11 and compare it to the C++03 methods.

For the purpose of this spike, the advanced features of C++11 are considered to be:

- Smart Pointers
- Static Asserts
- Constexpr
- Variadic Templates
- Lambdas

Spike ILOs

Design:	–	(0)
Implementation:	HIGH	(3)
Maintenance:	HIGH	(3)
Performance:	HIGH	(3)