

Spike: 03**Title:** Debugger Use**Author:** Ford Killeen, 9731822**Goals / deliverables:***Goals this spike aims to achieve.*

- Download, compile and run the *spike3* program
- Use an IDE debugger to step through the program and identify any bugs, leaks or issues
- Save the fixed code complete with comments detailing the changes

Deliverables required.

- Fixed up source code
- Spike report

Technologies, Tools, and Resources used:

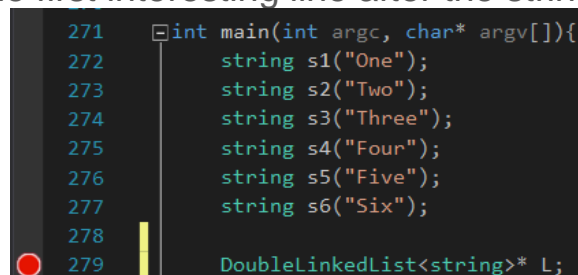
The following is required to complete this spike.

- Visual Studio 2015 (or similar)
- Debugging tool (in my case in-built to Visual Studio)
- Spike3 program source code (download from Blackboard)

Tasks undertaken:

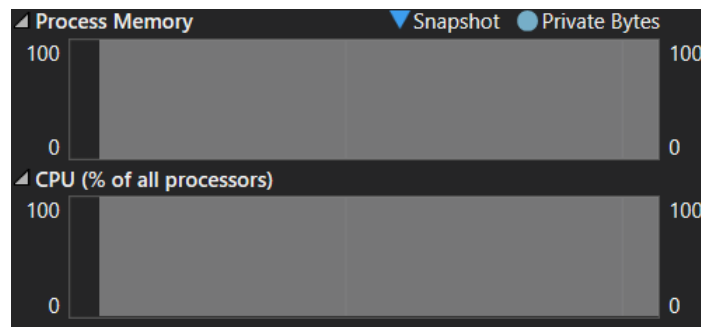
The dot point list below details the steps taken to complete this spike, including a few screenshots.

- I first began by heading straight down to the `main` function, and adding a breakpoint on the first interesting line after the strings setup



```
271 int main(int argc, char* argv){
272     string s1("One");
273     string s2("Two");
274     string s3("Three");
275     string s4("Four");
276     string s5("Five");
277     string s6("Six");
278
279     DoubleLinkedList<string>* L;
```

- Initially I had a suspicion that this line could need adjusting, but thought I would leave it until running the debugger. After starting the debugging and making one or two steps through the code I came across my first bug. The list `L` needed to be initialised.
- After fixing that issue I continued to step through the code searching for any obvious bugs/exceptions as well as taking note of when any memory leaks occurred (using the *Console output* and the *Diagnostic tools*)



- I soon came across my second issue, a read access violation error at line 49. This was due to the `nextnode` and `previousnode` objects not being initialised when the `Node` object was created.
- A few more debug runs later I found my 3rd, 4th and 5th bugs, all in the same method. When I came to the first `print()` in `main`, I made sure to step into it to check out what was happening. At first I saw that the `Node *N` was not being initialised. Next I noticed that `N` was never being set to the next `Node`, as well as the condition for the while loop only checking if `N != last` instead of `N != last->getNext()`
- I kept debugging, over and over again, keeping a keen eye on both the variable inspector and the console itself to take note of when the app got stuck in an infinite loop/memory leak
- As I found more and more bugs, I noticed a similarity with some of them, seeing a couple repeating bugs.
- I thought everything was looking good until I realised the value of `_length` after dropping some `Nodes`

`_length` 6

- Then I realised in the three spots where a `Node` is dropped, it never reduces the length. Thus finding bugs 9 to 11.
- I kept going, repeating the same tasks of debugging the application, checking all my tools and inspectors, and following through what exactly was happening in the code view until I (from what I can tell) found all bugs.

//Done!

What we found out:

The outcome of completing this spike was to familiarise oneself with the debugging tools provided in an IDE, in this case Visual Studio 2015 Community. By using the tools provided, I was successfully able to find all 13 bugs (pending Tutor feedback). It also helped reinforce my pre-existing basic understanding of the debugger within Visual Studio.

L	0xc0000005 {first=??? last=??? _length=??? }
first	<Unable to read memory>
last	<Unable to read memory>
_length	<Unable to read memory>

Recommendations:

I would recommend people to take a look into some of the other cool views available when debugging, such as the Call Stack or Stack Frame, which easily allow the user to jump back and forth between the code being called and the location in which it is being called within the stack. Otherwise this was a fairly straight forward spike and anyone already familiar with debugging should be good to move on.