**CAB201 Programming Principles - Semester 2, 2019**

**Report for Assignment: Project – Genomic Sequence Retrieval - Part II**

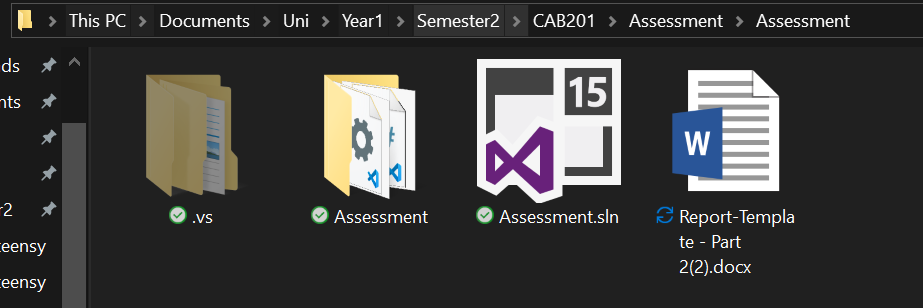
**Student name and number:**

*Rhys De Pater n10478515*

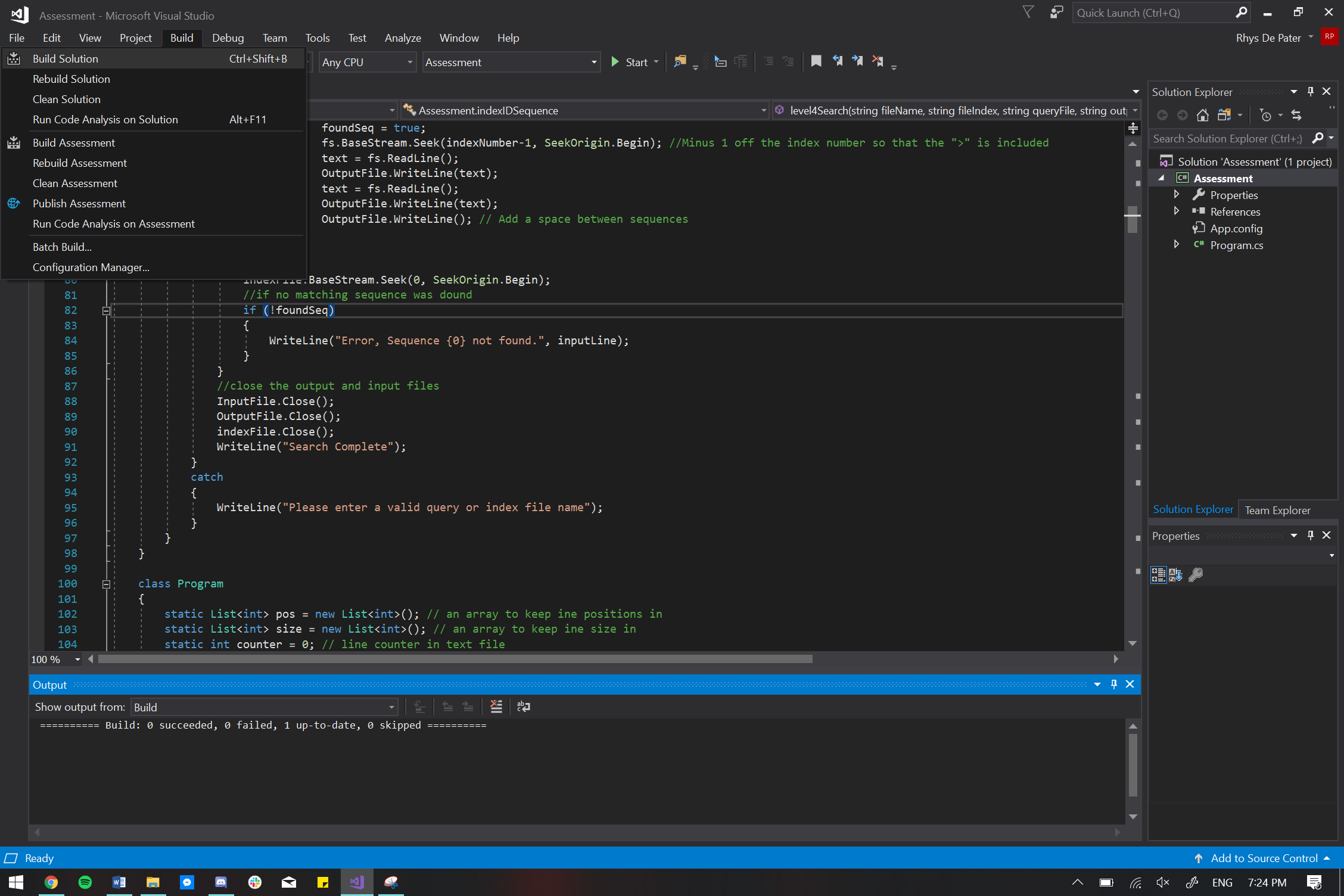
**Build and Run Instructions**

Please provide clear step-by-step instructions here on how to build your program in Visual Studio and run your program in the command line, given your submitted zip folder. For each step, you should include a screenshot. You may expand the box if needed.

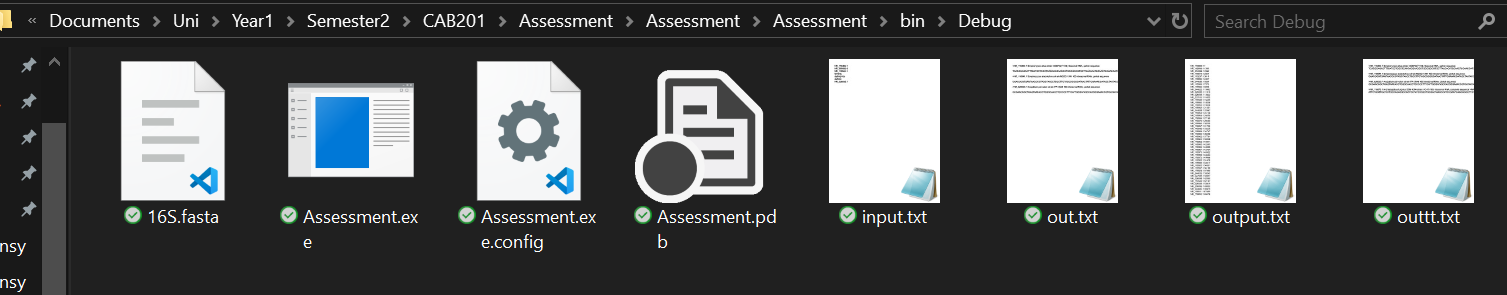
The sln file can be found within the “Assessment” solution folder.



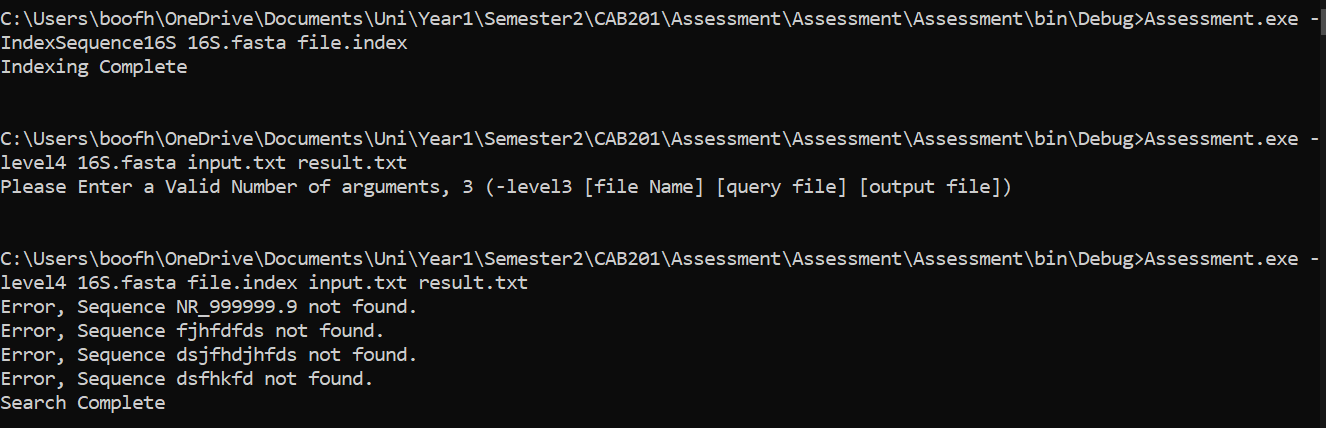
To build the program, within the solution, click build up on the top bar then click build solution, or press CTRL+SHIFT+B.



The exe file can be found in Assessment/Assessment/Bin/Debug.



Here is the exe file being run from command line.



**Statement of Completeness**

This statement of completeness will need to *accurately* state the functionality which has been implemented. There will be a penalty of 3.5 marks (loss of 3.5 marks) for a non-completed or submitted statement of completeness, and a penalty of 1 mark for each inaccurate statement to a maximum of 3 marks.

**In the following section, you are required to mark which functionality you have implemented. In the column on the right please mark ‘Y’ where you have completed this functionality, and ‘N’ where you have not. Please fill in any additional text boxes requested, and please note any limitations or bugs in the box at the end of each section. You may expand the table if you need more room for comments.**

|  |  |  |
| --- | --- | --- |
| **Basic Functionality** | | |
| **Build & Run** | When following the Build & Run instructions, the program successfully builds and runs. This was tested in a QUT SEF lab with a new login session, using the same zipped folder that is submitted. | Y |
| **Basic itinerary output** | The program displays the data from the file | Y |
| The program displays the appropriate line | Y |
| The correct amount of information is displayed, e.g. only the relevant entries | Y |
| The correct level, provided as a command line flag ***-levelN***, is executed | Y |
| The program **does not** store the whole file in memory, instead it accesses the file on disk | Y |
| **Error handling** | A clear error message is displayed when an incorrect number of arguments is provided | Y |
| A clear error message is provided when an incorrect flag is provided (e.g. not -level1, etc.) | Y |
| A clear error message is provided when the input file doesn’t exist, or is incorrectly formatted | Y |
| **Comments** | *Please note any limitations, bugs, logical errors, and/or run-time errors here* | |

|  |  |
| --- | --- |
| **Searching Algorithm – Part II**  Please underline, circle or highlight the levels that were completed. | |
| **Algorithm** | **Level:**  Level 4, level 5, level 6, level 7 |
| **Bonus:**  Sequence matching using wildcards  *Since I have done regex previously, I looked into how to find matches with regex and whether there are any differences to the regex I have done before. Then I looped through every second line in the file starting at line 1 (sequence lines) this is so I don’t have to do any checks to make sure it only prints sequences, because it is now only capable of reading and printing sequences. I then checked every line to see if it matches the regex and if it does print the match. The regex was created through a replace function. It was solved this was because it is very similar to my previous solutions, so I know it works and works fast* |
| **Level 7 Explanation**  *Any resources were used were from Microsoft’ documentation, so first the regex library was included, then the string given had every \* replaced with* (.\*?) *which means any character and 0 to any number of them, every line was then checked to see if it matched this and then has the match printed* |
| **Comments** | *My Indexing command is slightly different to the one provided, it is*  Assessment.exe -IndexSequence16S [fasta file] [index file]  This is because my index function is within the same solution but different class. |

**Screenshots of Functionality**

**In the following section, you are required to provide screenshots that provide evidence of your program working with provided input. You must complete this section.**

1. The 16S.fasta file has been provided with this template. Download them and place them in the same folder as your .exe file. You may have extra files, e.g. a query file, in this folder, and your .exe may be named differently. This is fine.
2. Open the command prompt and go to the above folder. In the command line, type the name of the .exe file and copy and paste following arguments:

Search16s -level4 16S.fasta 16S.index query.txt results.txt

|  |
| --- |
| **Place screenshot of entered command line arguments here:** |

1. Hit enter to run your program.

|  |
| --- |
| **Place screenshot(s) of the full output to console of your program. You may expand the box as necessary, and use as many screenshots as needed:** |

**Self-Assessment:**

1) How do I think I went with this assignment?

I think I went well on this assignment; all error messages are clear and understandable and cover any issue that could arise. I also improved efficiency wherever I could such as incrementing by 2 on level 2 and 3 to skip the sequences and only look at metadata. Additionally, an improvement was made so that on the sequenceID for the metadata was printed instead of every sequenceID in that line because even if they are all on the same line their metadata can be slightly different.

2) What did I find difficult in this assignment?

3) What would I do differently next time?

I found reading the file using the indexed file difficult, as I could easily get to where I wanted using seek but I couldn’t work out how to read the line as I didn’t have the line length in the index file and fileStream does not have a readLine option. This was fixed however by changing to a streamReader which has readLine and then using fs.BaseStream.Seek(indexNumber-1, SeekOrigin.Begin); to access seek.

And at the beginning If found reading and writing to file a little confusing and difficult but got used to it very quickly.

I wouldn’t do anything differently as I am fairly happy with what I have done. As I can’t find any logic errors or an bugs/things to crash it

4) Were there any bugs in my assignment, if so what were they?

The first bug occurred on level 2 where the user could search for any word they wanted to be it the sequence ID, part of a sequence ID, meta data or even the sequence itself, this is because at the time it looped through every line and used the contain function. To fix this firstly the loop incremented by 2 so only meta data was read and then for each line it looped through every letter until it reached a “>” then a 11-character substring is made one character ahead (The length of a sequence ID) this is then compared to the sequence ID provided. This made it so that only a full sequence ID works, any other word or partial sequence ID does not.

Additionally, when writing to the file nothing was present in the file afterwards, this was fixed by making the program close the file so that the data is saved.

The last bug in the program was from the index file, when using it the result line found were incorrect by about a line or two. It was found out this was caused because one wasn’t added to the counter after every line even though it should for “\n” although this seems like a slight issue, with a 40000 line file it adds up quickly**.**

**CRA:**

Please fill out the following CRA, reporting how many marks you believe your project might be awarded. Your assessment should be a considered reflection on what you have achieved. The purpose of this is to advise the marker of what you believe was achieved in order for us to pay attention to discrepancies. **Your self-assessment is NOT attracting marks, but must be provided (penalty applies if missing)**.

|  |  |
| --- | --- |
| **Code Quality**  To score points in this section, the student must follow the code quality guidelines as specified in the C# Coding Style Guide on Blackboard | **30/30** |
| Maintained consistent, clear, and meaningful standard in variable and method naming. No magic numbers. | 3/3 |
| Well structured – consistent and appropriate white spacing, line length, indentation, and separation into files within the project (i.e. one class per file) | 2/2 |
| Well commented – class header comment at beginning of each class, comment before every method, and in-line comments to explain complex or not easily discernible code. In-line comments are not excessive. | 4/4 |
| The DRY principle (Don’t repeat yourself) is followed where appropriate | 3/3 |
| Methods are single purpose and clear | 4/4 |
| Classes are well designed, with high cohesion and low coupling | 8/8 |
| Classes are separated into reusable modules where appropriate | 3/3 |
| Exceptions are thrown and handled appropriately | 3/3 |

|  |  |  |
| --- | --- | --- |
| Basic Functionality  To score marks in this section, your program must be able to be run from the command line with the appropriate arguments. | | **15/15** |
| *Basic Output* | The program displays the data from the file | 1/1 |
| The program displays the appropriate line | 1/1 |
| The correct amount of information is displayed, e.g. only the relevant entries | 1/1 |
| The correct level, provided as a command line flag ***-levelN***, is executed | 1/1 |
| The program **does not** store the whole file in memory, instead it accesses the file on disk | 5/5 |
| **Total:** | **9/9** |
| *Error Handling* | A clear error message is displayed when an incorrect number of arguments is provided | 2/2 |
| A clear error message is provided when an incorrect flag is provided (e.g. not -level1, etc.) | 2/2 |
| A clear error message is provided when the input file doesn’t exist, or is incorrectly formatted | 2/2 |
| **Total:** | **6/6** |

|  |  |  |
| --- | --- | --- |
| Part II  To score marks in this section, your program must be able to run levels 4-7. | | Marks Available: |
| 55/55 |
| *Level 4* | The program creates a file as specified by the command line arguments | 2/2 |
| The index file contains a list of all the sequence ids with the appropriate byte-offset | 5/5 |
| The searching program makes use of the created index file to execute a number of queries | 5/5 |
| A clear error message is provided when the index file does not exist | 1/1 |
| Clear error messages are provided when the query file cannot be found, or when a bad query is given, like in Level 3 | 2/2 |
| **Total:** | **15/15** |
| *Level 5* | The program correctly locates and prints the requested sequence ids | 10/10 |
| A clear error message is provided when the sequence does not exist | 5/5 |
| **Total:** | **15/15** |
| *Level 6* | The program correctly locates and prints the requested sequence ids | 15/15 |
| A clear error message is provided when the keyword does not exist | 5/5 |
| **Total:** | **20/20** |
| *Level 7 (Optional)* | The program correctly decodes the expression given and identifies any matching sequences | 5/5 |
| A clear error message is provided when the sequence does not exist | 5/5 |
| **Total:** | **+10/+10** |