**CAB201 Programming Principles - Semester 1, 2019**

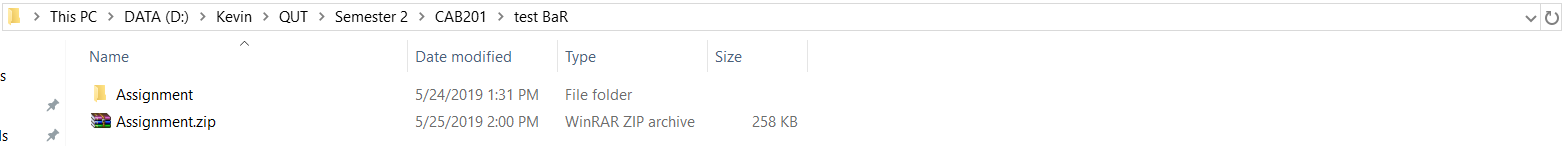
**Report for Assignment: Project – Flying Postman**

**Student name and number:**

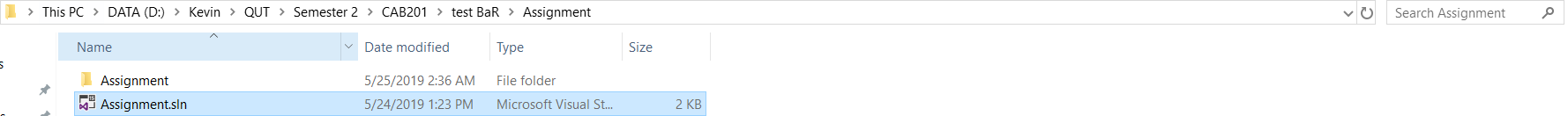
*Nghia Hieu Huynh, n10315071*

**Build and Run Instructions**

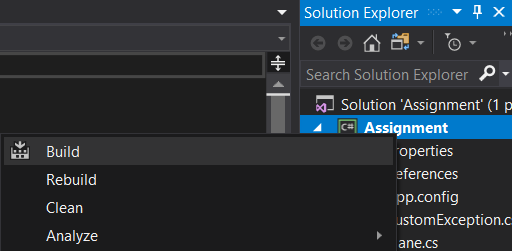
Please provide clear step-by-step instructions here on how to build your program in Visual Studio and run your program in the command line, given your submitted zip folder. For each step, you should include a screenshot. You may expand the box if needed.

*\_ Extract the zip file*

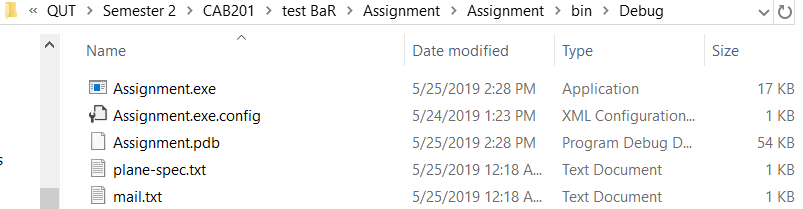
*\_ Go into Assignment folder, there is Assignment.sln file. Open it to see the source code:*



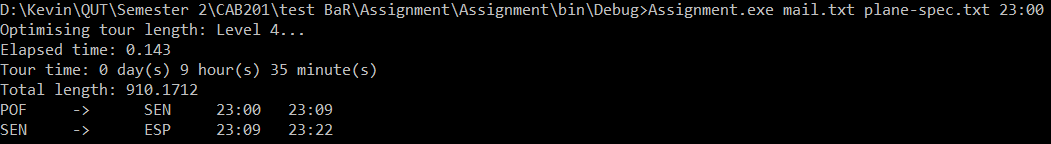
*\_ In Solution Explorer, click on Assignment > Build*



*\_ Go to Assignment/bin/Debug to see the .exe file. Also putting mail.txt and plane-spec.txt there in order to run via cmd.*



*\_ Using cmd, change directory to the Debug folder and run using command in picture:*



**Statement of Completeness**

This statement of completeness will need to *accurately* state the functionality which has been implemented. There will be a penalty of 5 marks (loss of 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 4 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.**

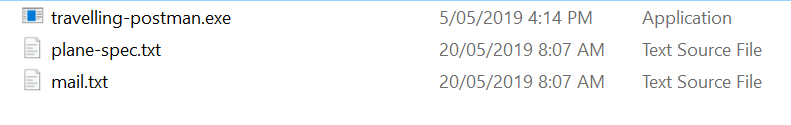
|  |  |  |
| --- | --- | --- |
| **Basic Functionality** | | |
| **Basic itinerary output** | An itinerary including all stations from a provided mail file is output to the console, with no repeats, starting and ending at the post office | Y |
| The time taken to find the itinerary is correctly timed and output in the format: “Elapsed time: \_\_ seconds” | Y |
| Total distance of the itinerary is correctly calculated and output in the format: “Tour length: \_\_” | Y |
| The itinerary is saved to a provided file if the flag “-o” and a valid file location is provided in the command line arguments | Y |
| **Time and plane** | The time taken for each leg of the itinerary is correctly calculated, rounded to the nearest minute | Y |
| The time of the tour starts at the value provided in the command line argument | Y |
| Time ticks over 24 hours correctly | Y |
| The total time of the itinerary is correctly calculated and output in the format: “Tour time: \_ days \_ hours \_ minutes” | Y |
| Plane refuels when needed, and is output to the itinerary | Y |
| **Error handling** | A clear error message and explanation of which command line arguments to use is printed if the incorrect number of arguments is provided | Y |
| A clear error message is provided if the provided plane or mail file does not exist or can’t be read, or if the output file cannot be written to | Y |
| A clear error message is provided if the provided plane or mail file cannot be parsed correctly, for example if a station’s coordinates aren’t listed or aren’t a valid number | Y |
| **Comments** | *Please note any limitations, bugs, logical errors, and/or run-time errors here* | |

|  |  |  |
| --- | --- | --- |
| **Tour Finding Algorithm**  Please underline, circle or highlight the highest level completed. For the bonus and level 4, provide details on your implementation. | | |
| **Algorithm** | **Level:**  Level 1, level 2, level 3, level 4 | |
| **Bonus:**  Implement a tour solver which can account for tours in which not all stations are within the flight range of the plane.  *If completed, insert a brief discussion of your approach, explaining how and why. If the bonus level is not automatically run in your program, provide clear instructions on how to run.* | N |
| **Level 4 Explanation**  Only complete this box if you completed a level 4 implementation.  If you implemented level 4 but it was worse (tour efficiency wise) than level 2 for all tours, you should still complete this section. Have your program run your level 3 solution instead, but provide clear instructions on how to find the code for your level 4 algorithm. Depending on the effort and quality of your implementation, you may still receive additional marks.  *Source of algorithm (e.g. website, paper):*  <http://pedrohfsd.com/2017/08/09/2opt-part1.html>  *Pseudo-code of algorithm:*  *\_ List<Station> 2OptSwap(List, I , K)*  *{ 1. Take the first i-1 stations and add to list A*  *2. Take stations[i] to stations[k], reverse it and add to A*  *3. Take stations[k] to the final station and add to A*  *4. Return A*  }  \_ Repeat until no improvement is made {  Start again  Minimum\_distance = TourLength(currentTour)  For I =1 to Number of Stations – 1:  For k= i+1 to Number of Stations:  {  New\_tour = 2OptSwap(current\_tour,I,k)  New\_length = TourLength(New\_tour)  If New\_Length < Min\_Length:  Min\_Length = New\_Length  Start again  }  }  *Tour efficiency and run time comparison:*  *[Include a comparison of the run time and tour efficiency of this approach vs.* ***your*** *level 2 approach for several tours. Include at least the 51 stops, 101 stops, 237 stops, and 1002 stops examples from Blackboard -> Tour and Itinerary Examples, and any other tours as you wish. You don’t need to include the full tours your program generates, but you should include the total distance and run time. You are welcome to include extra information or graphs to help the comparison, e.g. a graph comparison of the run time vs efficiency in both approaches]*  \_ 51 stops:  + Lvl2:    + Lvl4:    \_ 101 stops:  + Lvl2:    + Lvl4:    \_ 237 stops  + Lvl2:    + Lvl4:    \_ 1001 stops:  + Lvl2:    + Lvl4: The program freeze  *Location of algorithm implementations:*  *[Provide clear instructions on how to find your level 4 and level 2 and 3 algorithm implementations in your code, for example by listing the file and method name]*  *\_ Level 2 is located between /\*Level 2\*/ comment in Solve.cs*  *\_ Level 3 is located between /\*Level 3\*/ comment in Solve.cs*  *\_ Level 4 is located between /\*Level 4\*/ comment in Solve.cs* | |
| **Comments** | *Please note any limitations, bugs, logical errors, differences from provided examples and possible explanations, and/or run-time errors here*  \_ Level 2 1002 Stops takes longer time. | |

**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. A mail.txt file and a plane-spec.txt file have been provided with this template. Download them and place them in the same folder as your .exe file, so that the folder looks something like this:

 You may have extra files in this folder, and your .exe may be named differently. This is fine.

1. 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:

mail.txt plane-spec.txt 23:00 -o itinerary.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:** |

|  |
| --- |
| **Place screenshot of your folder structure after running, which should include the itinerary.txt file:** |

|  |
| --- |
| **Copy and paste the text of the itinerary.txt file here:**  Optimising tour length: Level 2...  Elapsed time: 0.007  Tour time: 0 day(s) 9 hour(s) 33 minute(s) minutes  Total length: 896.2979  POF -> SEN 23:00 23:09  SEN -> ESP 23:09 23:22  ESP -> DJI 23:22 23:31  DJI -> MCO 23:31 23:42  MCO -> ECU 23:42 23:54  ECU -> TUR 23:54 00:03  TUR -> SYC 00:03 00:12  SYC -> MOZ 00:12 00:24  \*\*\* refuel 8 minutes \*\*\*  MOZ -> NRU 00:32 00:43  NRU -> COD 00:43 00:52  COD -> MNG 00:52 01:03  MNG -> OMN 01:03 01:14  OMN -> GUY 01:14 01:23  GUY -> GNQ 01:23 01:33  GNQ -> ISR 01:33 01:42  ISR -> TON 01:42 01:51  TON -> PAK 01:51 02:01  \*\*\* refuel 8 minutes \*\*\*  PAK -> AND 02:09 02:18  AND -> AZE 02:18 02:28  AZE -> WSM 02:28 02:39  WSM -> GEO 02:39 02:52  GEO -> CUB 02:52 03:01  CUB -> BFA 03:01 03:10  BFA -> GRC 03:10 03:24  GRC -> ETH 03:24 03:35  \*\*\* refuel 8 minutes \*\*\*  ETH -> PHL 03:43 03:52  PHL -> BTN 03:52 03:58  BTN -> RUS 03:58 04:09  RUS -> DEU 04:09 04:19  DEU -> HUN 04:19 04:29  HUN -> SWE 04:29 04:39  SWE -> KAZ 04:39 04:47  KAZ -> CRI 04:47 04:56  CRI -> EST 04:56 05:07  \*\*\* refuel 8 minutes \*\*\*  EST -> CHL 05:15 05:25  CHL -> BEL 05:25 05:36  BEL -> URY 05:36 05:48  URY -> TUN 05:48 05:59  TUN -> BEN 05:59 06:12  BEN -> BDI 06:12 06:21  BDI -> HRV 06:21 06:32  HRV -> LAO 06:32 06:44  \*\*\* refuel 8 minutes \*\*\*  LAO -> MDA 06:52 07:05  MDA -> SGP 07:05 07:14  SGP -> MEX 07:14 07:26  MEX -> YEM 07:26 07:35  YEM -> CZE 07:35 07:46  CZE -> PER 07:46 07:56  PER -> JOR 07:56 08:06  JOR -> MNE 08:06 08:16  \*\*\* refuel 8 minutes \*\*\*  MNE -> POF 08:24 08:33 |

1. Open the Tour Visualizer program from Blackboard. Click ‘Load station file’ and select the mail.txt file. Then click ‘Load tour file’ and select the itinerary.txt file. Take a screenshot of the visualized tour from the program. **If the Tour Visualizer program cannot plot your tour, it is incorrectly formatted – make sure it matches the example outputs.**

|  |
| --- |
| **Place screenshot of the visualized tour here:** |