**Unit Test Cases**

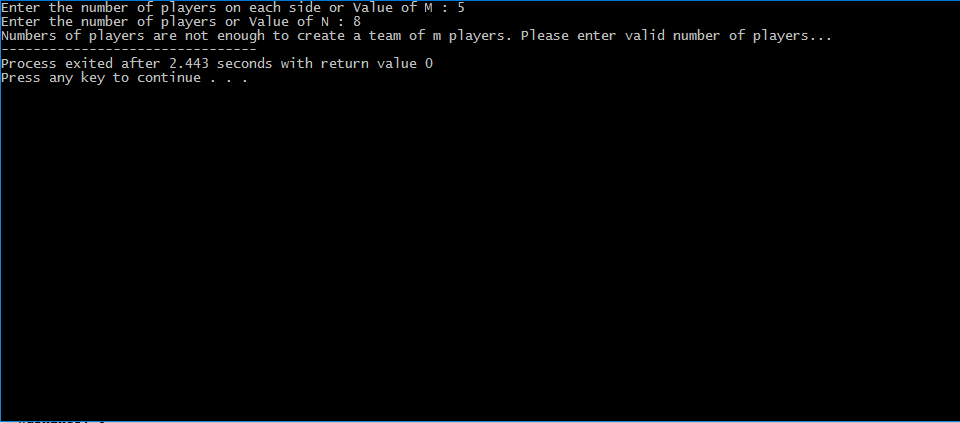
|  |
| --- |
| Test case#: 1.1 Test Name:- Input  System: Solution Code Sub name:- Solution Code  Designed by: Harpreet Singh Design Date:- 8/6/2020  Executed by: Harpreet Singh Execution Date:- 8/6/2020  Short Description: The program must take input correctly on the console. If input can’t make the output a message with invalid input must be shown. |

|  |
| --- |
| Pre-Conditions:-  The user has to provide the proper input from the console. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Action | Expected system Response | Pass/fail | comment |
| 1. | Data types | The program terminates if the input is not of the same data types as a variable is declared | Pass |  |
| 2. | Input Variables | The input from the console is directly stored in the input variables. | Pass |  |
| 3. | Using Map | A map is used in the program and all the elements are also stored on the map. | Pass |  |
| 4. | Invalid Input | If the Input value is invalid a message with invalid input is shown on the console. | Pass |  |

|  |
| --- |
| Post condition:  There is no post condition. For valid output, the input must be proper. |

**#Test Case(1.1) Screenshot**

****

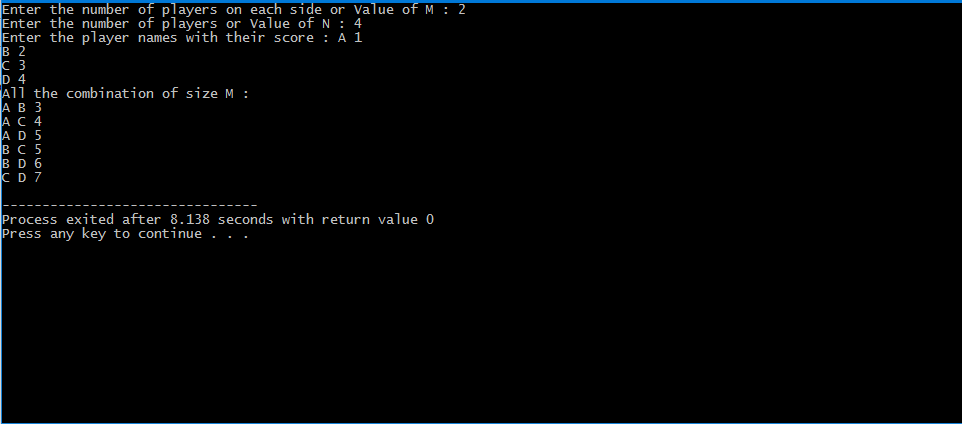
|  |
| --- |
| Test case#: 2.1 Test Name:- Generation of M sized combinations  System: Solution Code Sub name:- Solution Code  Designed by: Harpreet Singh Design Date:- 8/6/2020  Executed by: Execution Date:- 8/6/2020  Short Description: For making the team of m players, the combinations of size M are needed at first. So this test will generate all the combinations of players of size M. |

|  |
| --- |
| Pre-Conditions:-  The user has to provide the Number of players in each team or Value of M  The user has to provide the number of players i.e. a value of N.  The user has to provide all the player names along with their scores. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Action | Expected system Response | Pass/fail | comment |
| 1. | Providing the value of M. | The program gets the value of M and stored it in a variable m. | Pass |  |
| 2. | Providing the value of N | The program gets the value of M and stored it in a variable m. | Pass |  |
| 3. | Getting player names along with a score. | After getting the value of M and N all the N players' names are stored in a string vector array and scores of players in an integer vector array. | Pass |  |
| 4. | Combination generated by a function. | After passing M and N and two vector array to a function named printCombination() all the Combination are generated and shown as output | Pass |  |

|  |
| --- |
| Post condition:  There is no post condition for this once the value of M and N and two vector arrays are passed to a function the printCombination() function will generate all the combination of size M. |

**#Test Case(2.1) Screenshot**

****

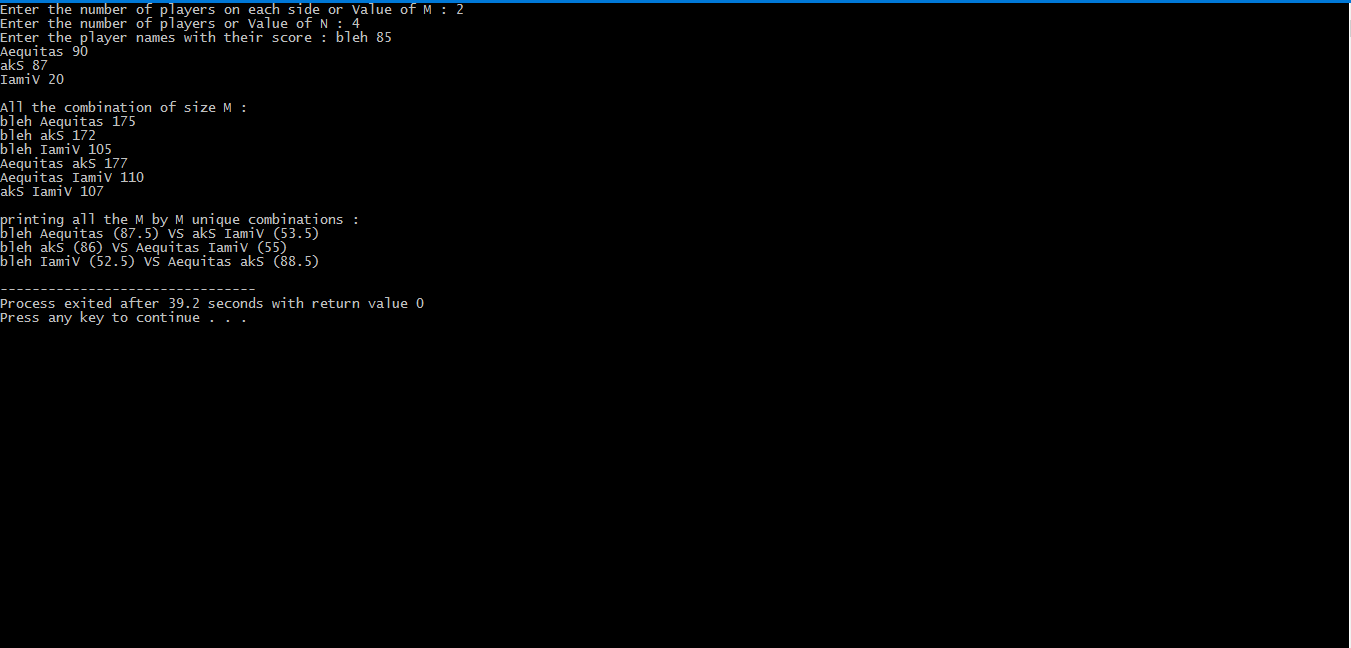
|  |
| --- |
| Test case#: 3.1 Test name: Testing the generation of a unique team of size M  System: Solution Code Subsystem: Solution Code  Design: Harpreet Singh Design date: 9/6/20 Executed by: Harpreet Singh Execution date: 9/6/20  Short description: As in Test Case #2.1 All the combinations of players of size M have been generated, but for M by M match both the teams should have unique players. To do the same the code is tested. |

|  |
| --- |
| Pre-Conditions:-  The user has to provide all the combinations of size M to function named ace(). |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Action | Expected system Response | Pass/fail | comment |
| 1. | Getting the combinations for printCombination() function. | The program stores all the combinations in a global vector array. | Pass. |  |
| 2. | Passing vector array to ace() function. | All the combinations are stored in a global vector array so it can be used directly in ace() function. | Pass |  |
| 3. | Generation | After getting a combination from this vector array all them by m unique combinations are generated. | Pass |  |
| 4. | Base Condition | The ace() function is a recursive function so the base condition is provided and working fine. | Pass. |  |
| 5. | Change in global Array | After every iteration, all the redundant elements from the global vector array are removed. | Pass |  |

|  |
| --- |
| Post condition:  There is not any Post condition, once ace() function global array and it recursively generated all the unique M by M combinations. |

**#Test Case(3.1) Screenshot**

****

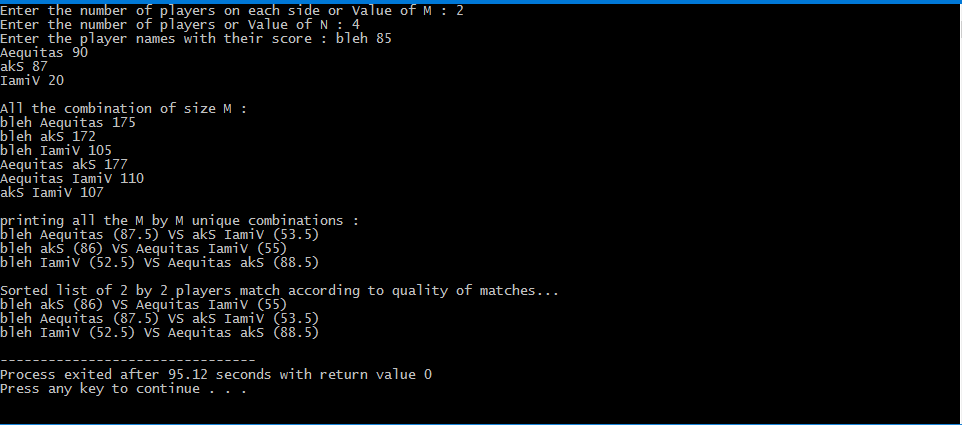
|  |
| --- |
| Test case#: 4.1 Test name: Testing the Output is sorted or not  System: Solution Code Subsystem: Solution Code  Design: Harpreet Singh Design date: 11/6/20 Executed by: Harpreet Singh Execution date: 11/6/20  Short description: As the ace() function only generates M by M combinations but these are not sorted according to quality of matches. So this test is testing the generation of output in sorted order. |

|  |
| --- |
| Pre-Conditions:-  All the combinations generated by ace() function should be stored in a global vector array which takes pair as an element. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Action | Expected system Response | Pass/fail | comment |
| 1. | Storing M by M Combinations | All the Combinations are stored in global vector array which takes pair as an element | Pass. |  |
| 2. | Using a global vector array. | As the vector is global and is accessed in the main() function. | Pass |  |
| 3. | Match quality. | Match quality is the difference between the averages of two teams. This is stored as the second element of the pair in global vector array | Pass |  |
| 4. | Sorting | The match quality is stored in vector array and the whole array is sorted according to the match quality i.e. best to worst. | Pass. |  |
| 5. | Output | After sorting array elements are printed as sorted output. | Pass |  |

|  |
| --- |
| Post condition:  There is not any Post condition, simple sort the global vector array and print all the elements which are in sorted order. |

**#Test Case(4.1) Screenshot**

****

|  |
| --- |
| Test case#: 5.1 Test name: Output for different test cases  System: Solution Code Subsystem: Solution Code  Design: Harpreet Singh Design date: 11/6/20 Executed by: Harpreet Singh Execution date: 11/6/20  Short description: The program is tested for different test cases. The result-set of all the input cases must be according to question. |

|  |
| --- |
| Pre-Conditions:-  The proper Input must be given by the user.  All the combinations os size M must be properly generated.  All the unique M by M combinations must be properly generated.  All the M by M combinations must be sorted according to the quality of matches. |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Step | Action | Expected system Response | Pass/fail | comment |
| 1. | The small value of M and N | The output is generated for a small value of M and N | Pass. |  |
| 2. | Medium value of M and N | The output is generated properly for a medium value of M and N | Pass |  |
| 3. | Large Value | Long long int and long long double is used for large input and output is generated properly | Pass |  |
| 4. | Invalid Input | For Invalid input, no output is generated just an invalid message is shown as output. | Pass. |  |
| 5. | The smallest size of M | The output is properly generated for the smallest value of M I .e.1. | Pass |  |

|  |
| --- |
| Post condition:  If Input is valid the proper output will be generated... |

**#Test Case(5.1) Screenshot**

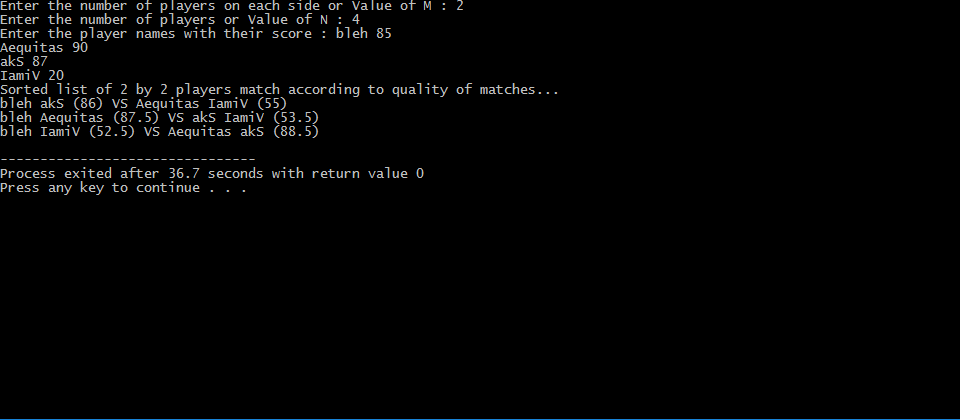


Fig 1 Medium-sized Input.

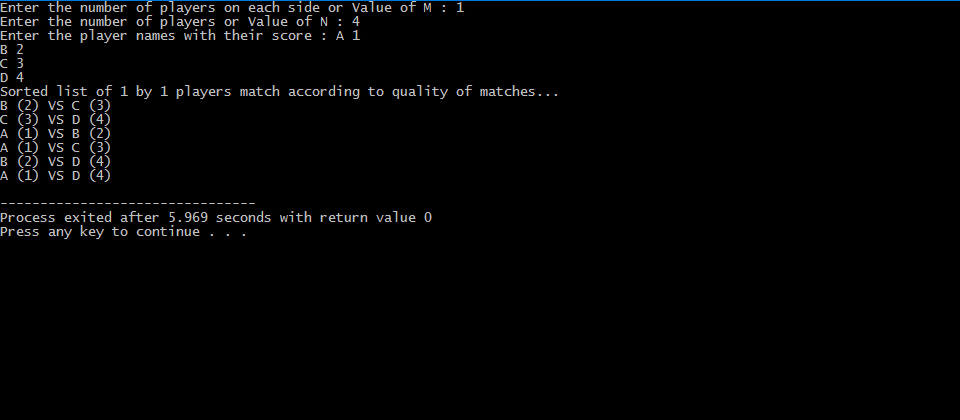


Fig 2 smallest value of M

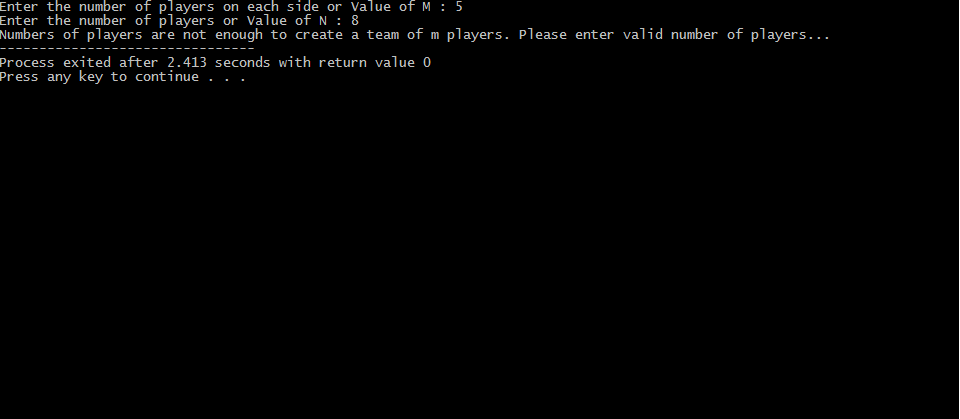


Fig.3 Invalid Input

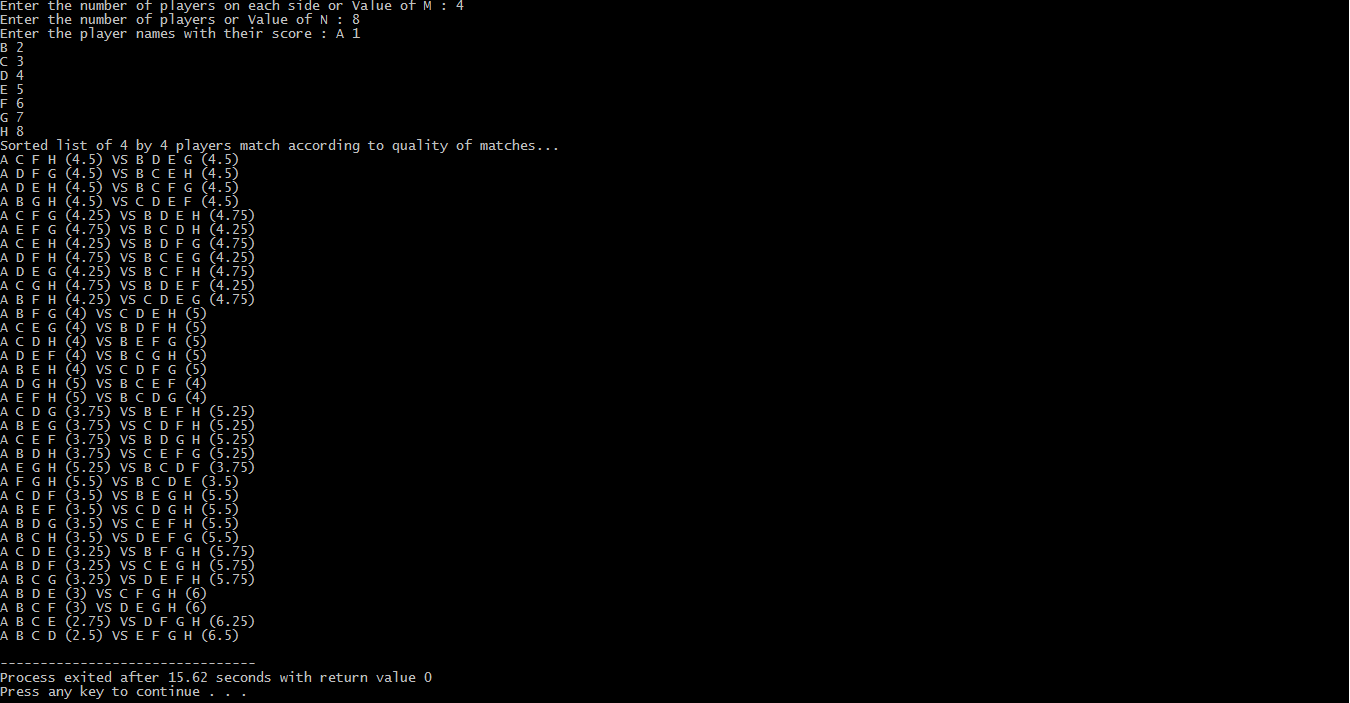


Fig.4 Large Input