

Brian Morris Part C Final

by Brian Morris

FILE	11831_BRIAN_MORRIS_BRIAN_MORRIS_PART_C_FINAL_173932_1709996085.DOCX (1.16M)		
TIME SUBMITTED	08-APR-2019 04:22PM (UTC+0100)	WORD COUNT	2991
SUBMISSION ID	104384841	CHARACTER COUNT	15931

Programs in C and documentation

PART C

BRIAN MORRIS (1809130)

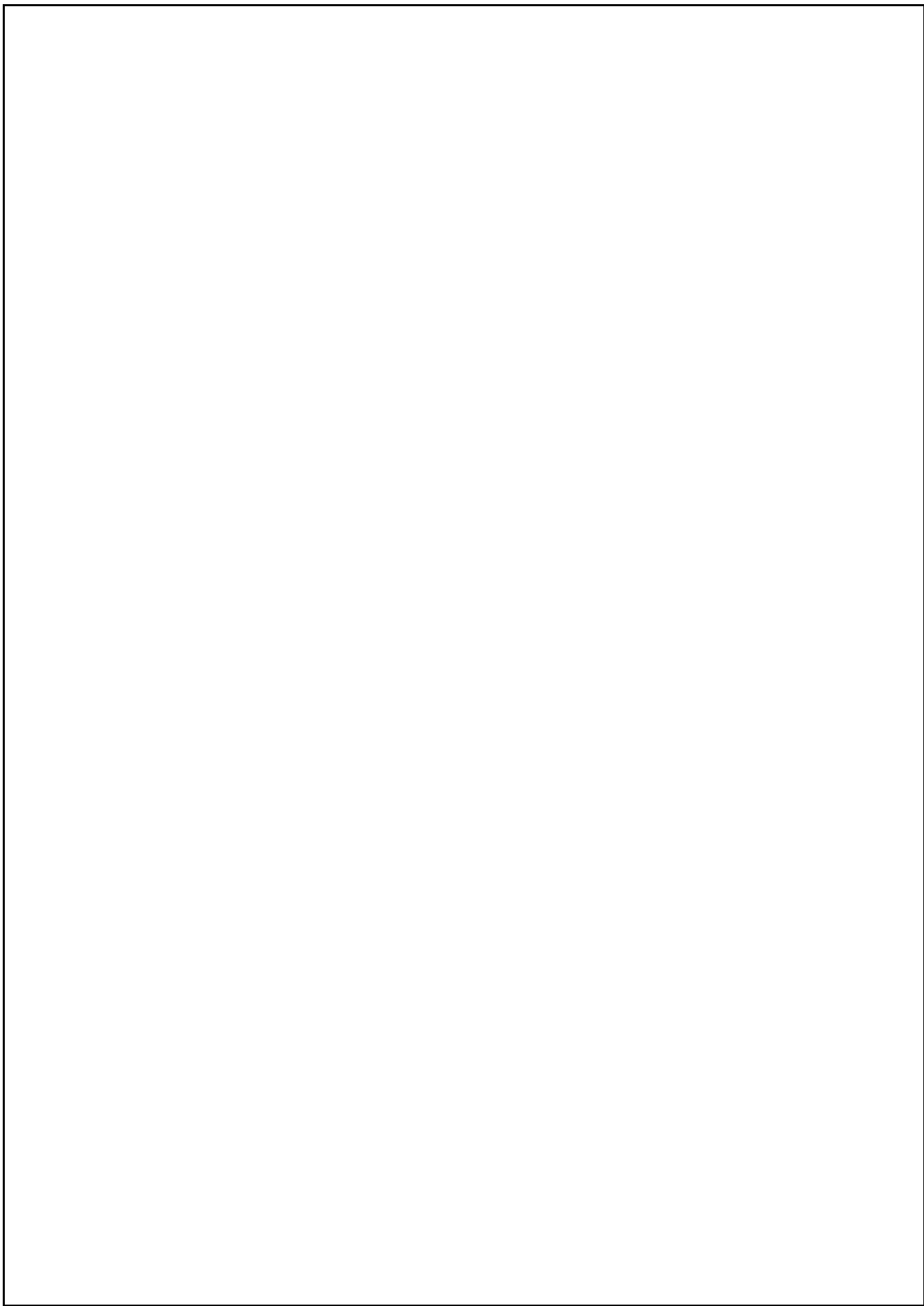
Abstract

This is the last part of a report that documents a C programming solution for a 'pairing game' with annotated code listing, additional screen shoots of output produced during testing and screen shoots of errors outside of the running application.

Table of contents

Contents

Abstract.....	1
Part C Specification	4
Part C Specific problems identified to assist in completing the exercise.	5
Part C Pseudo Code.....	6
Part C Annotated Code	7
Part C Debug testing with normal values.....	15
Part C Testing with negative numbers	24
Part C Testing with boundary values	24
Part C Testing with impossible values.....	27
Improvements.....	30
Bibliography	31

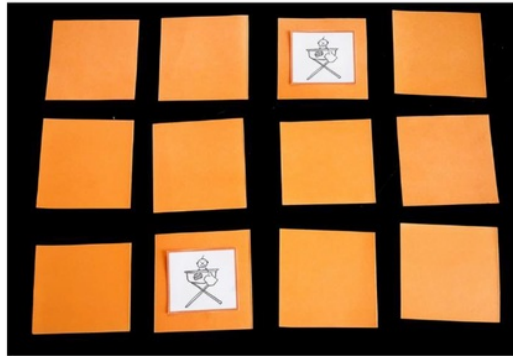


Part C Specification

PART C – (IMPLEMENTATION – FUNCTION BASED) (due date May 2nd 2019 3pm)

Pairs

Develop a C program to allow the user to play a version of a popular card matching game, such as that shown below:



(Note: the game need not be visually complex, and can be implemented with numbers or single letters rather than pictures)

- Identify specific problems in completing the exercise.
 - You may also constrain a problem (simplify it) to produce a simpler solution that may act as a stepping stone to solving the real problem
- Write a clear analysis of the solutions (the thought process behind them) – what logical stages are required. You could use a flowchart or pseudocode
- Show a code listing with annotations
- Show testing (screenshots) that clearly demonstrate that the solutions work under all circumstances

The matching game has various rules which you must identify. For example the player may turn over two “cards” at a time and see their content. Then they must turn the cards back to the blank side before their next turn. The player must attempt to identify matching cards. The goal of the game is to turn over the largest number of matching pairs in the lowest number of moves.

You may implement the rules simply or in a more complex way. You should use at least one function other than the main function. (It is OK to make any arrays used global, to simplify the program. Any variables should be passed as a parameter to a function with a return value if required) Extra credit will be given for more complex implementations.

Note: Remember, there are many marks available for design etc. It is better to have a version of the game that is partially functional, even if it does not incorporate the more complex requirements.

Part C Specific problems identified to assist in completing the exercise.

The limitation of not being able to pass array values to a function limits the use of functions. The use of pointers would have made some of the procedures easier to achieve as its not necessarily good practice to use global variables. For example as programming teams get larger ensuring one programmer is not changing the variable values becomes more difficult to track so the use of a global variable should be kept to a minimum as a good working practice

Ensuring that the numbers are generated randomly each time and that each random number is unique within its range ensuring there are even number of matches so the game can be completed.

How each grid slot is to be identified would it be a using a character or numerical value. Graphical options would be beyond this requirement at the time.

Creating a routine to keep track of pairs

Ensuring the routine used to draw out the grid is able to accommodate the items that have been paired are identified when the grid is being drawn

Keep a count of the users turns as they cycle trough the game and output the count once they have completed the game

Part C Pseudo Code

Call libraries

Setup variables for integers, characters, array and open file for score

Setup arrays for grid slots

Ask user to play a game while 'Y'

- Generate random numbers 1..12

- Ensure each number is unique

- Generate another set of random number between 1..12

- Ensure each number is unique

- Display current winning score

 - Use functions and loops to draw out grid and pair indicators

 - Ask user to select first grid value

 - Call function to collect value

 - Validate number to be within parameters

 - Check value hasn't been selected previously

 - Ask user to select from grid if the number has been paired before

 - Ask user to select second grid value

 - Call function to collect value

 - Validate number to be with parameters

 - Check value hasn't been selected previously

 - Ask user to select from the grid if the number has been paired before

 - Check value of selections to see if they are pairs

 - If matching pair set value of grid numbers to Zero

 - Add one to game counter loop

 - If the value of total grid== 0 all cards have been paired

Inform user they have beaten the game and output how many times it took to complete the game.

Part C Annotated Code

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include <string.h>
#include <conio.h>

// Initialize function prototypes
int Selection();
int DrawRow(int, int);
char keepPlaying();
char HighscoreTable();
char Instructions();

void main()
{
    FILE *fp;
    int GridNums[24]; //create arrays to store random numbers
    int i, j, r = 0, c = 0, low = 1, high = 12, flag = 1, row1, row2, highscore =
100, total = 0, gamego = 0, runningscore=0; // initialize variables for counters etc
    srand(time(NULL)); //used to seed randomness
    char playing;
    char name[5][10], tempname[10];
    int score[6], tempscore, currenthigh;

    fp = fopen("Highscore.txt", "r"); //open highscore file read in values and
store highest score for use within the game
    for (i=0; i<5; i++)
    {
        fscanf(fp, "%s", &name[i]);
        fscanf(fp, "%d", &score[i]);
    }
    for (i = 0; i < 4; i++)
    {
        for (j = i + 1; j < 5; j++)
        {
            if (score[j] > score[i])
            {
                tempscore = score[j];
                score[j] = score[i];
                score[i] = tempscore;
                strcpy(tempname, name[j]);
                strcpy(name[j], name[i]);
                strcpy(name[i], tempname);
            }
        }
    }
    fclose(fp);

    //Call function to display highscore table
    HighscoreTable();
    //Call Function to display instructions
    Instructions();
    //Generate numbers for First Batch
    printf("Print first run of numbers from First batch \n");
    for (i = 0; i < 12; i++)
    {
        GridNums[i] = rand() % (high - low + 1) + low;
        printf("%d ", GridNums[i]);
    }
}
```

```

    }
    //Check for duplicates in First batch of numbers
    while (flag == 1)
    {
        flag = 0;
        for (i = 0; i < 12; i++)
        {
            for (j = i + 1; j < 12; j++)
            {
                if (GridNums[i] == GridNums[j])//replace if duplicate
                {
                    GridNums[j] = rand() % (high - low + 1) + low;
                    flag = 1;
                }
            }
        }
    }
    printf("\nUnique numbers of First batch \n"); //debug check to output unique
    numbers
    for (i = 0; i < 12; i++)
    {
        printf("%d ", GridNums[i]);
    }
    flag = 1;
    printf("\nValue of flag after after first batch %d:", flag);
    //Generate second batch of numbers
    printf("\nGenerate first run of numbers from second batch \n");
    for (i = 12; i < 24; i++)
    {
        GridNums[i] = rand() % (high - low + 1) + low;
        printf("%d ", GridNums[i]);
    }
    printf("The flag equals: %d", flag);
    //Check for duplicates in second batch
    while (flag == 1)
    {
        flag = 0;
        for (i = 12; i < 24; i++)
        {
            for (j = i + 1; j < 24; j++)
            {
                if (GridNums[i] == GridNums[j])//replace if duplicate
                {
                    GridNums[j] = rand() % (high - low + 1) + low;
                    flag = 1;
                }
            }
        }
    }
    printf("\nUnique numbers of Second batch \n"); //debug check //Output unique
    numbers
    for (i = 12; i < 24; i++)
    {
        printf("%d ", GridNums[i]);
    }
    printf("\n\nThe flag is%d", flag);

```

```

printf("\nOutput the numbers in a Row!!!\n"); //debug check //output all
numbers
for (i = 0; i < 24; i++)
{
    printf("%d ", GridNums[i]);
}
//system("cls"); //I would comment this out for production
//print all numbers
printf("\nOutput the numbers in a Grid!!!\n");
for (i = 0; i < 6; i++)
{
    printf("%d ", GridNums[i]);
}
printf("\n");
for (i = 6; i < 12; i++)
{
    printf("%d ", GridNums[i]);
}
printf("\n");
for (i = 12; i < 18; i++)
{
    printf("%d ", GridNums[i]);
}
printf("\n");
for (i = 18; i < 24; i++)
{
    printf("%d ", GridNums[i]);
}
printf("\n\n");

system("cls");
printf("Name: %s           Highscore: %d\n\n", name[0], score[0]);

DrawRow(0, 6); //call function and pass two values
printf("\n");
for (i = 0; i < 6; i++)
{
    if (GridNums[i] == 0)
    {

        printf("Pair ");
    }
    else
        printf("**** ");
}

DrawRow(6, 12); //call function and pass two values
printf("\n");
for (i = 6; i < 12; i++)
{
    if (GridNums[i] == 0)
    {
        printf("Pair ");
    }
    else
        printf("**** ");
}

DrawRow(12, 18); //call function and pass two values
printf("\n");
for (i = 12; i < 18; i++)

```

```

{
    if (GridNums[i] == 0)
    {
        printf("Pair ");
    }
    else
        printf("**** ");
}

DrawRow(18, 24); //call function and pass two values
printf("\n");
for (i = 18; i < 24; i++)
{
    if (GridNums[i] == 0)
    {
        printf("Pair ");
    }
    else
        printf("**** ");
}

playing = keepPlaying();
//Generate the total
for (i = 0; i < 24; i++)
{
    total = total + GridNums[i];
}
//printf("\nTotal %d", total); //used to debug
if (total == 0)
{
    printf("\n You completed the game. Well Done");
    playing = 'n';
}
while (playing == 'y' || playing == 'Y')
{
    row1 = Selection();
    while (GridNums[row1] == 0)
    {
        printf("You have already selected this slot");
        row1 = Selection();
    }

    row2 = Selection();
    while (GridNums[row2] == 0)
    {
        printf("You have already selected this slot");
        row2 = Selection();
    }

    printf("You selected %d and %d \n", GridNums[row1], GridNums[row2]);

    while (row1 == row2) //check the same grid slot has not been selected
    twice
    {
        printf("You selected the same Gridslot twice please select two
different Gridslots ");
        row1 = Selection();
        row2 = Selection();
        printf("You selected %d and ", GridNums[row1]);
    }
}

```

```

        printf("%d \n", GridNums[row2]);
    }
    if (GridNums[row1] == GridNums[row2]) //check the numbers are pairs
    {
        system("cls");
        printf("Name: %s          Highscore:   %d\n\n", name[0],
score[0]);
        runningscore = runningscore + 100;
        printf("\nScore: %d\n", runningscore);
        printf("\nWell done matching pair\n");
        GridNums[row1] = 0;
        value to 0 used to track game progress //set grid num
        GridNums[row2] = 0;
    }
    else
    {
        system("cls");
        printf("Name: %s          Highscore:   %d\n\n", name[0],
score[0]);
        runningscore = runningscore - 15;
        printf("\nScore: %d\n", runningscore);
        printf("You selected %d and ", GridNums[row1]);
        printf("%d \n", GridNums[row2]);
        printf("\nSorry Try again\n");
    }

    DrawRow(0, 6);
    printf("\n");
    for (i = 0; i < 6; i++)
    {
        if (GridNums[i] == 0)
        {
            printf("Pair ");
        }
        else
            printf("**** ");
    }

    DrawRow(6, 12);
    printf("\n");
    for (i = 6; i < 12; i++)
    {
        if (GridNums[i] == 0)
        {
            printf("Pair ");
        }
        else
            printf("**** ");
    }

    DrawRow(12, 18);
    printf("\n");
    for (i = 12; i < 18; i++)

```

```

        {
            if (GridNums[i] == 0)
            {
                printf("Pair ");
            }
            else
                printf("**** ");
        }
        DrawRow(18, 24);
        printf("\n");
        for (i = 18; i < 24; i++)
        {
            if (GridNums[i] == 0)
            {
                printf("Pair ");
            }
            else
                printf("**** ");
        }
        //Add one to game loop counter and take a mark off as game not completed

        gamego = gamego + 1;
        runningscore = runningscore - 10;

        //Generate the total to see if the game is complete
        total = 0;
        for (i = 0; i < 24; i++)
        {
            total = total + GridNums[i];
        }
        // printf("\nTotal %d", total); was used as debug and testing
        if (total == 0) //0*24=0
        {
            //Output the game loop counter score

            printf("\n\nYou completed the game in %d turns.\nWell Done!\n\n",
gamego);

            if (runningscore > score[4])
            {
                printf("Please enter your initials: ");
                scanf(" %s", &name[4]);
                score[4] = runningscore;
                fp = fopen("Highscore.txt", "w");
                for (i = 0; i < 5; i++)
                {
                    fprintf(fp, " %s" " %d", name[i], score[i]);
                }
                fclose(fp);
            }

            HighscoreTable();
            playing = 'n';
        }
    }

}

//function to select a number in the grid
int Selection()

```

```

{
    int choice;
    printf("\n\nPlease select a Grid Slot between 1 and 24\n");
    scanf("%d", &choice);
    while (choice > 24 || choice < 1) //check number is within range and loop until
valid
    {
        printf("\nPlease enter a choice between 1 and 24\n");
        scanf("%d", &choice);
    }
    choice = choice - 1; //array count starts at 0 but we are asking the user to
start at 1
    return choice;
}

// function to keep playing
char keepPlaying()
{
    char response;
    printf("\n\nWould you like to play a game? ");
    printf("\n\nY/N: ");
    scanf(" %c", &response);
    return response;
}

//function to draw grid numbers
int DrawRow(int a, int b)
{
    printf("\n\n");
    int i;
    for (i = a; i < b; i++)
    {
        printf("%d    ", i + 1);
    }
    printf("\n");
    return NULL;
    //printf("\n");
}

char Instructions()
{
    system("COLOR 12");
    system("cls");
    printf("In this game you are attempting to");
    printf("\nmatch pairs of numbers in the least ");
    printf("\namount of turns possible.");
    printf("\n\n");
    printf("Select two different numbers between 1 and 24");
    printf("\nthe less times it takes the higher your score will be.");
    printf("\n\n");
    printf("Good Luck!!!\n\n");
    printf("Press any key to start....");
    getch();
    system("cls");
    system("COLOR 07");
    return NULL;
}

//function to read in high score record sort into decending values and print out
results

```

```

char HighscoreTable()
{
    FILE *fp;
    char name[5][10];
    int score[5], i, j;
    int tempScore;
    char tempname[10];
    fp = fopen("Highscore.txt", "r");
    system("COLOR 12");
    //change screen colours
    for (i = 0; i < 5; i++)
    {
        fscanf(fp, "%s", &name[i]);
        fscanf(fp, "%d", &score[i]);
    }
    for (i = 0; i < 4; i++)
    {
        for (j = i + 1; j < 5; j++)
        {
            if (score[j] > score[i])
            {
                tempScore = score[j];
                score[j] = score[i];
                score[i] = tempScore;
                strcpy(tempname, name[j]);
                strcpy(name[j], name[i]);
                strcpy(name[i], tempname);
            }
        }
    }
    printf("    HighScore Table!\n");
    printf("Name        Score\n\n");

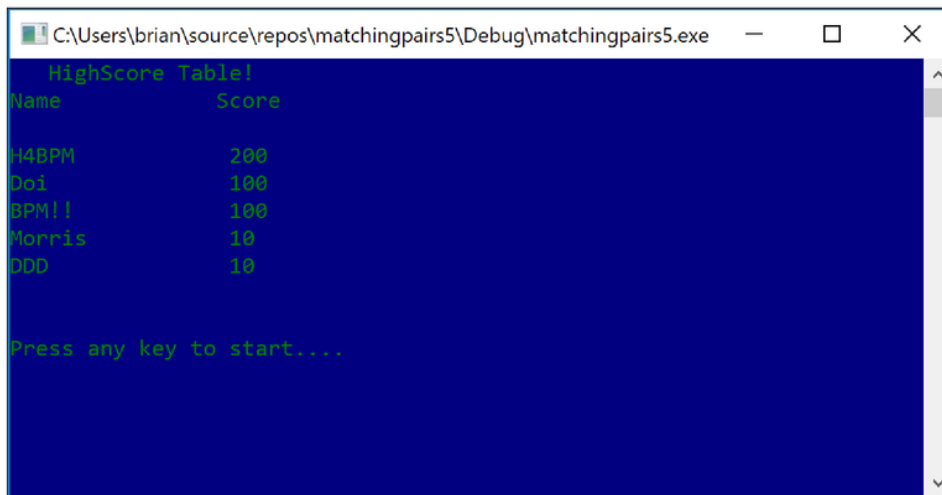
    for (i = 0; i < 5; i++)
    {
        printf("%s        %d\n", name[i], score[i]);
    }

    printf("\n\n");
    printf("Press any key to start....");
    getch();
    system("COLOR 07");
    return NULL;
}

```

Part C Debug testing with normal values

On start up a high score table is presented. The background screen colour and text colour are changed [1]. The table values are loaded from a file and sorted into descending order. The highest score and players name from the table is stored as a variable. Throughout the game these values are presented to the user. Once the user presses any key [2] to continue the screen changes these colours back to black background with white text.

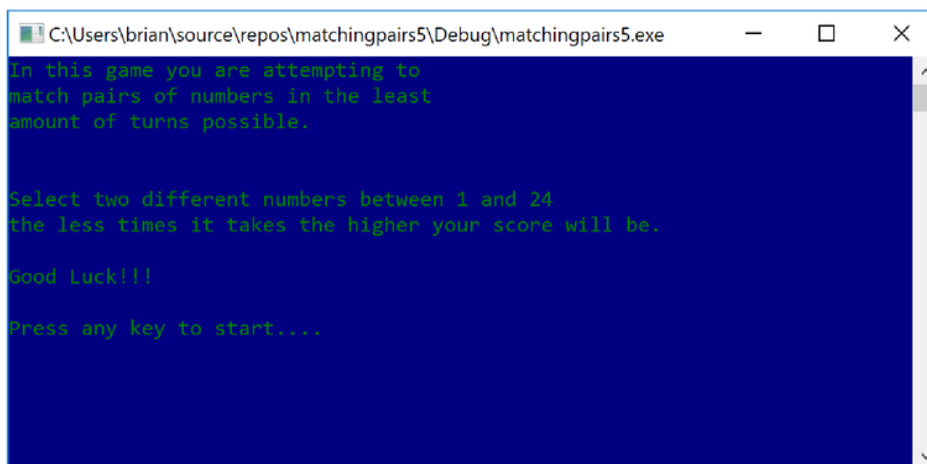


The screenshot shows a Windows command prompt window titled "C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe". The window has a blue background and green text. It displays a high score table with the following data:

Name	Score
H4BPM	200
Doi	100
BPM!!	100
Morris	10
DDD	10

Below the table, the text "Press any key to start...." is displayed.

Reviewing the system, it was decided to add an additional function to present the end users with some simple instructions. Using similar colours as used in the previous function not to make the output too garish.



The screenshot shows a Windows command prompt window titled "C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe". The window has a blue background and green text. It displays the following instructions:

```
In this game you are attempting to
match pairs of numbers in the least
amount of turns possible.

Select two different numbers between 1 and 24
the less times it takes the higher your score will be.

Good Luck!!!

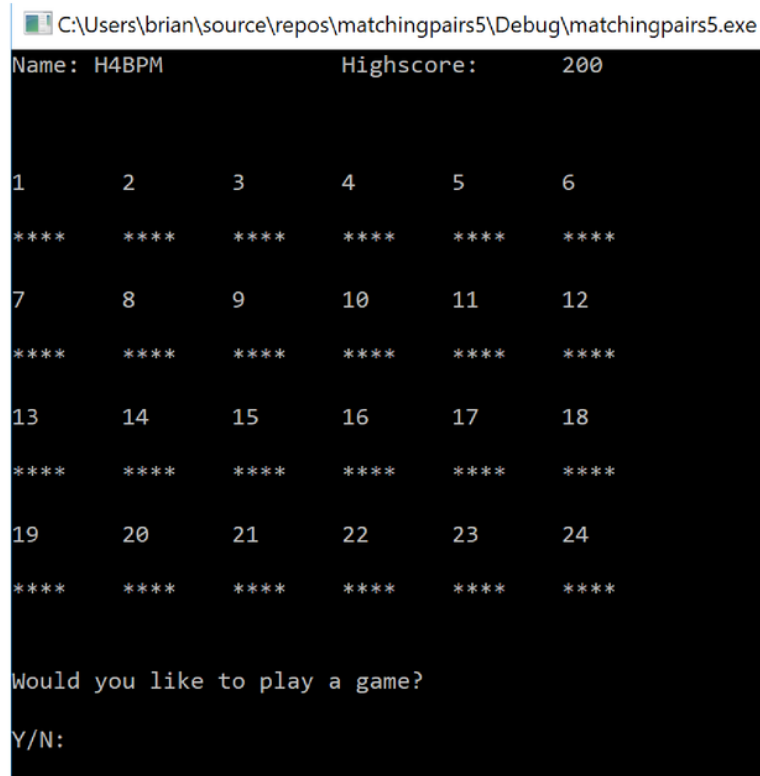
Press any key to start....
```

For testing line 144 "system("cls")" has been commented out to show the output results of the table. If not testing would take too long. It is possible the number of pairs could have been reduced to make the game less complex.

A screen shot of the game grid is taken at the beginning of the game and used to validate testing output.

```
Output the numbers in a Grid!!!
2      7      6      8      1      3
9      11     10     12     4      5
1      8      11     5      4      9
7      6      3      12     2     10
```

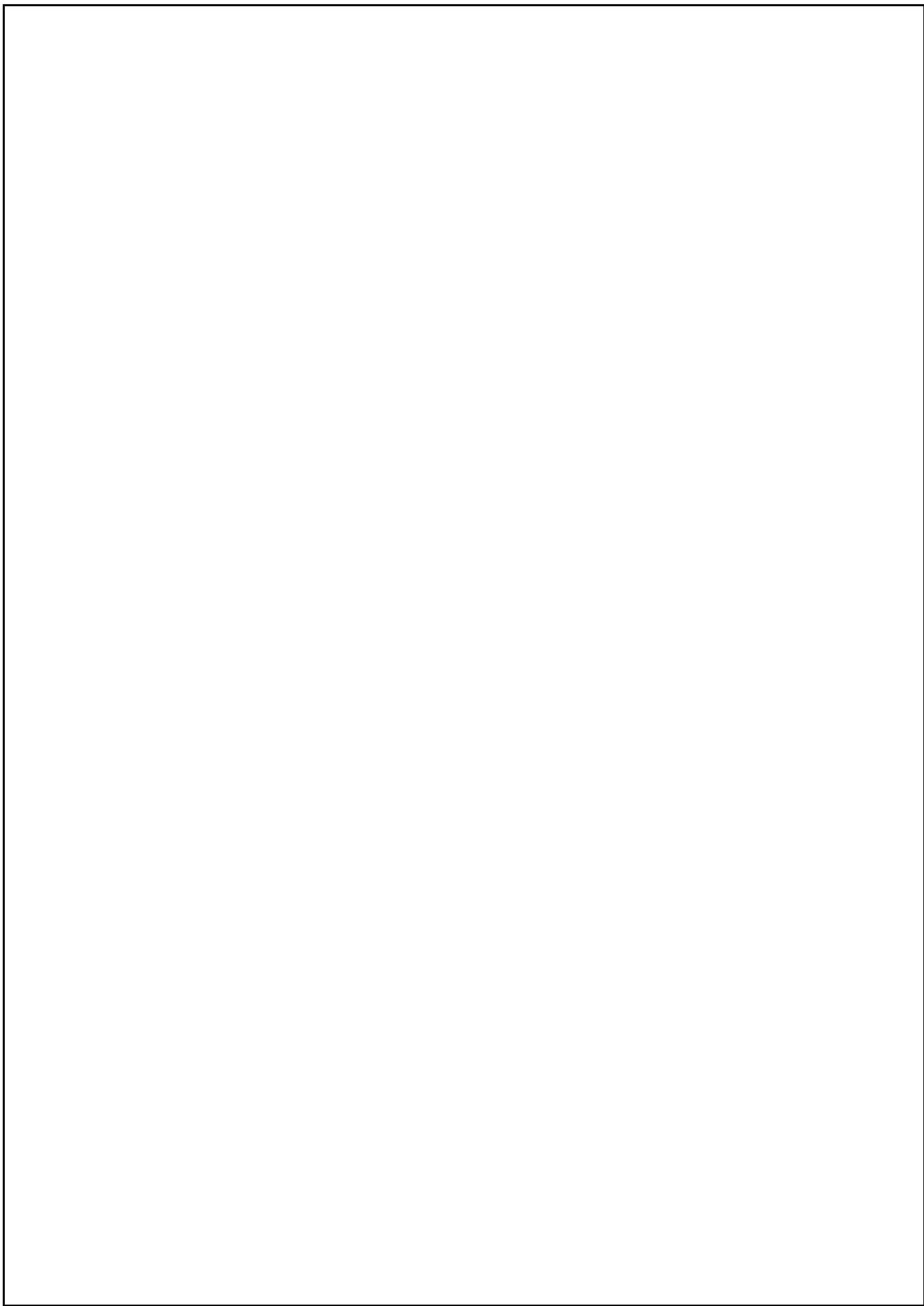
The screen layout shows the name of the player with the highest score their high score a layout of the game grid and prompts the user if they would like to play a game any response other than a 'y' or 'Y' will end the game,



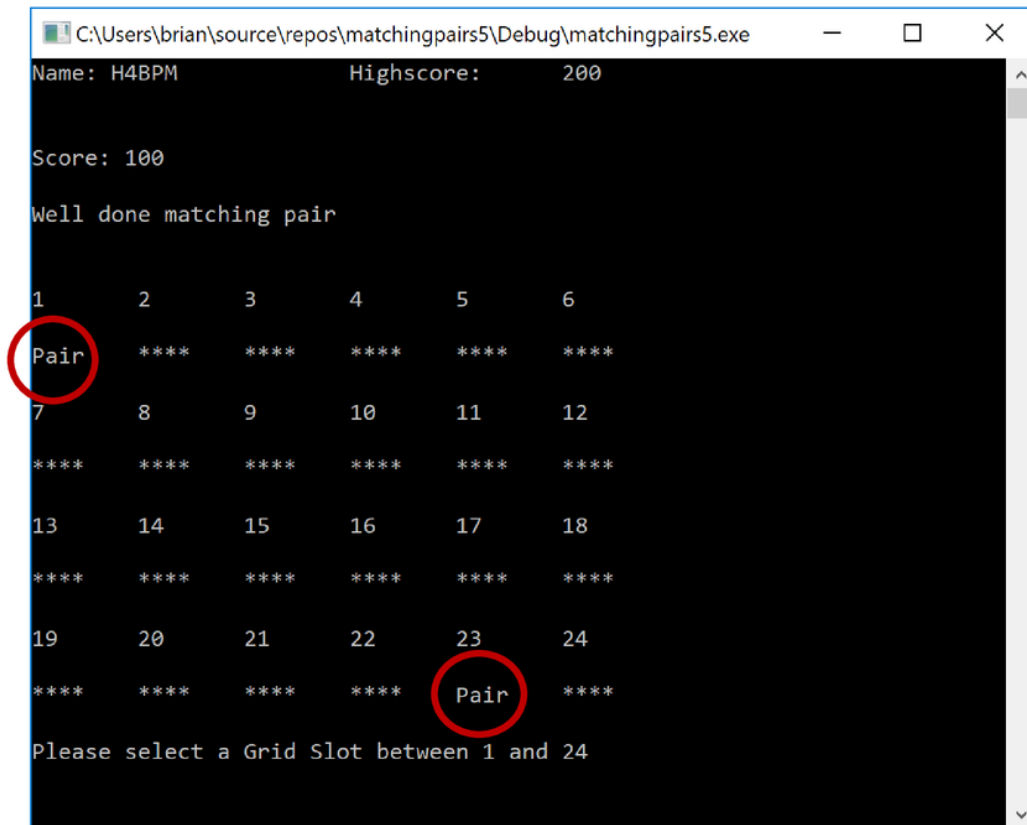
```
C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe
Name: H4BPM      Highscore: 200

1      2      3      4      5      6
****   ****   ****   ****   ****   ****
7      8      9     10     11     12
****   ****   ****   ****   ****   ****
13     14     15     16     17     18
****   ****   ****   ****   ****   ****
19     20     21     22     23     24
****   ****   ****   ****   ****   ****

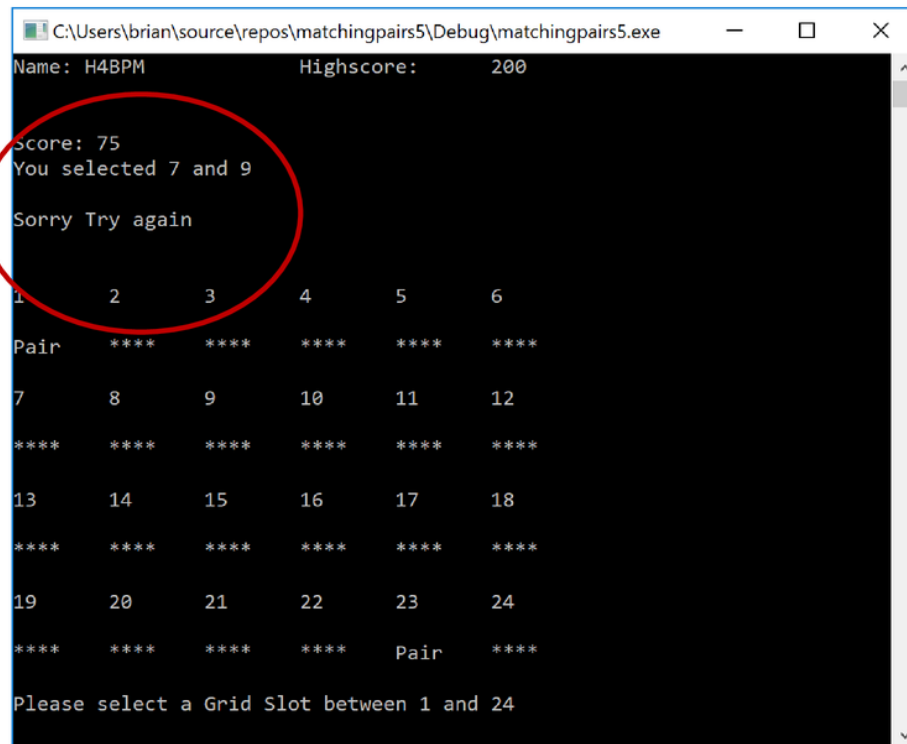
Would you like to play a game?
Y/N:
```



Grid slot 1 and 23 are pairs. The user informed that they selected a matching pair and the indicators show that items in that grid slot have been paired.



Selecting grid options that don't pair redraws the screen with the high score details, it updates the running score subtracting from the total, informs the user of the values unhidden but also informs them that they have been unsuccessful the routine adds one more to the game counter and prompts the user to select new numbers.



The screenshot shows a Windows application window titled "C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe". The window displays the following text:

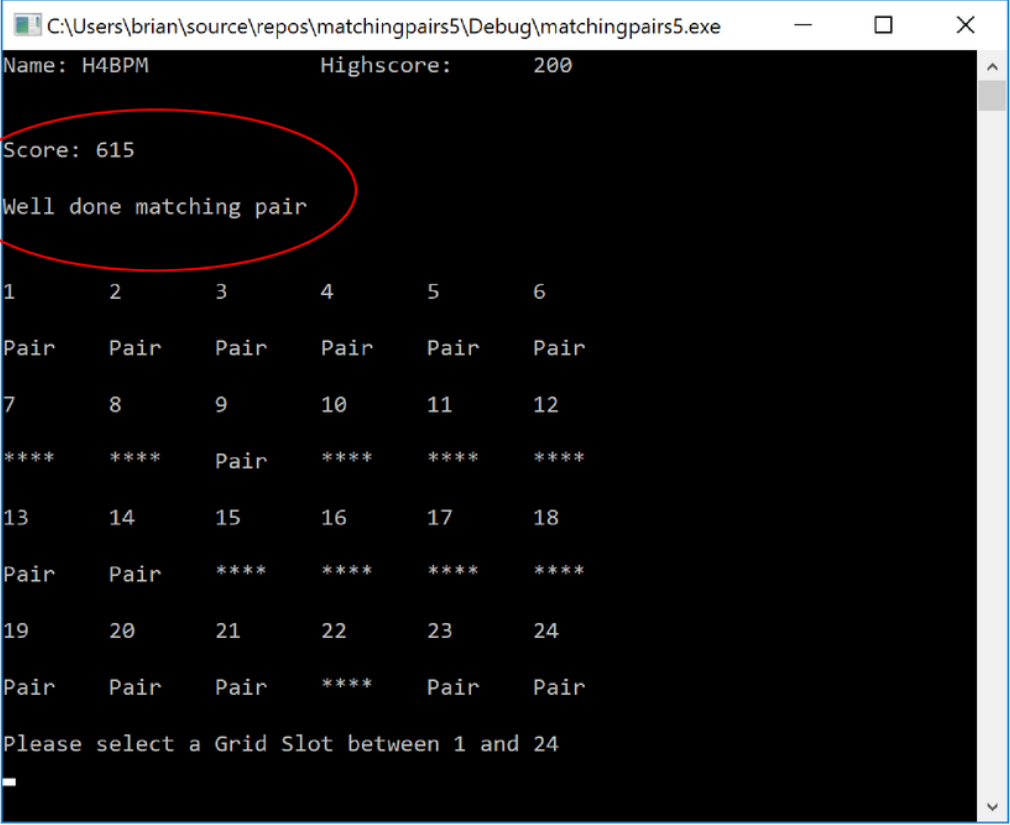
```
Name: H4BPM           Highscore: 200
Score: 75
You selected 7 and 9
Sorry Try again

1      2      3      4      5      6
Pair   ****   ****   ****   ****   ****
7      8      9      10     11     12
****   ****   ****   ****   ****   ****
13     14     15     16     17     18
****   ****   ****   ****   ****   ****
19     20     21     22     23     24
****   ****   ****   ****   Pair   ****

Please select a Grid Slot between 1 and 24
```

A red circle highlights the text "You selected 7 and 9" and "Sorry Try again".

As indicated the players score is update throughout the game.

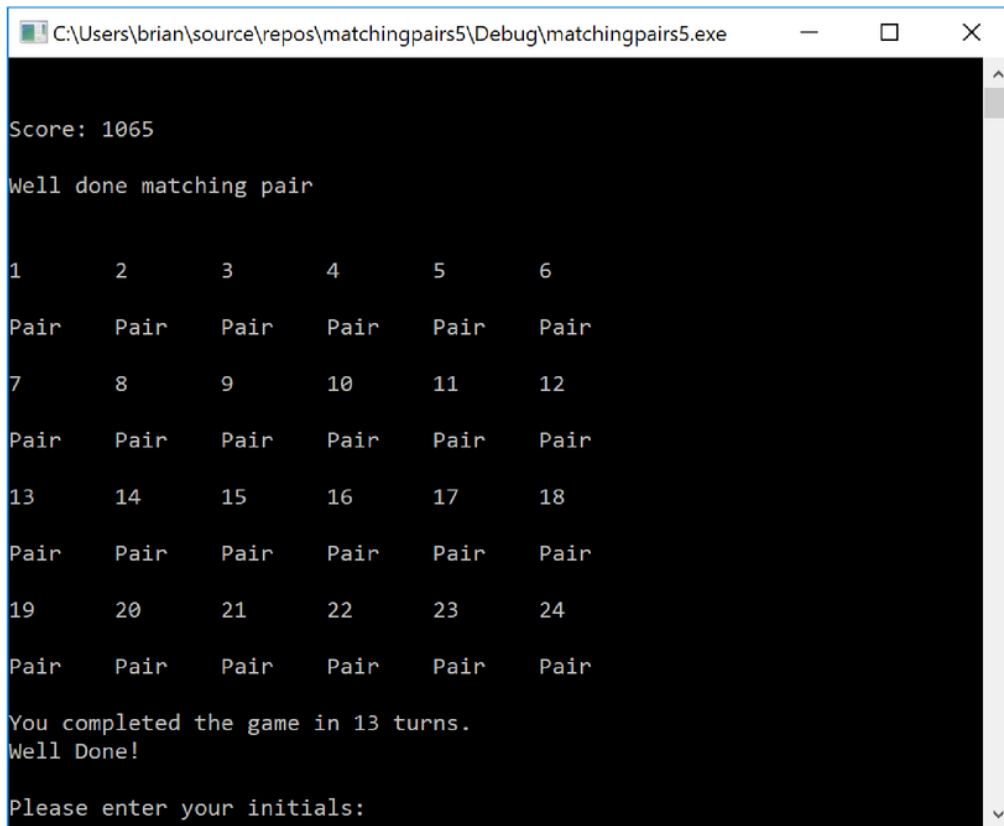


The screenshot shows a Windows application window titled "C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe". The game interface displays the player's name "Name: H4BPM" and a high score of "Highscore: 200". The current score is "Score: 615", which is circled in red. Below the score, it says "Well done matching pair". The game board is a 6x4 grid of slots, numbered 1 to 24. The current state of the grid is as follows:

1	2	3	4	5	6
Pair	Pair	Pair	Pair	Pair	Pair
7	8	9	10	11	12
****	****	Pair	****	****	****
13	14	15	16	17	18
Pair	Pair	****	****	****	****
19	20	21	22	23	24
Pair	Pair	Pair	****	Pair	Pair

At the bottom of the window, it says "Please select a Grid Slot between 1 and 24" followed by a cursor.

On completion the game informs the player that its completed and how many attempts it took to complete the game. The game prompts the player to enter their initials .



```
C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe

Score: 1065

Well done matching pair

1      2      3      4      5      6
Pair   Pair   Pair   Pair   Pair   Pair
7      8      9      10     11     12
Pair   Pair   Pair   Pair   Pair   Pair
13     14     15     16     17     18
Pair   Pair   Pair   Pair   Pair   Pair
19     20     21     22     23     24
Pair   Pair   Pair   Pair   Pair   Pair

You completed the game in 13 turns.
Well Done!

Please enter your initials:
```

The high score table is update, saved then the results are loaded again.As the player had a qualifying high score the score is present within the table.

```
You completed the game in 13 turns.  
Well Done!
```

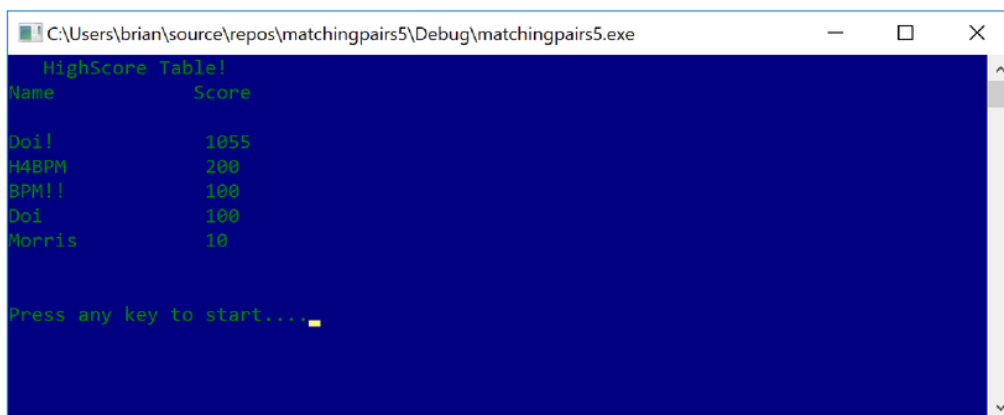
```
Please enter your initials: Doi!
```

```
HighScore Table!
```

Name	Score
Doi!	1055
H4BPM	200
BPM!!	100
Doi	100
Morris	10

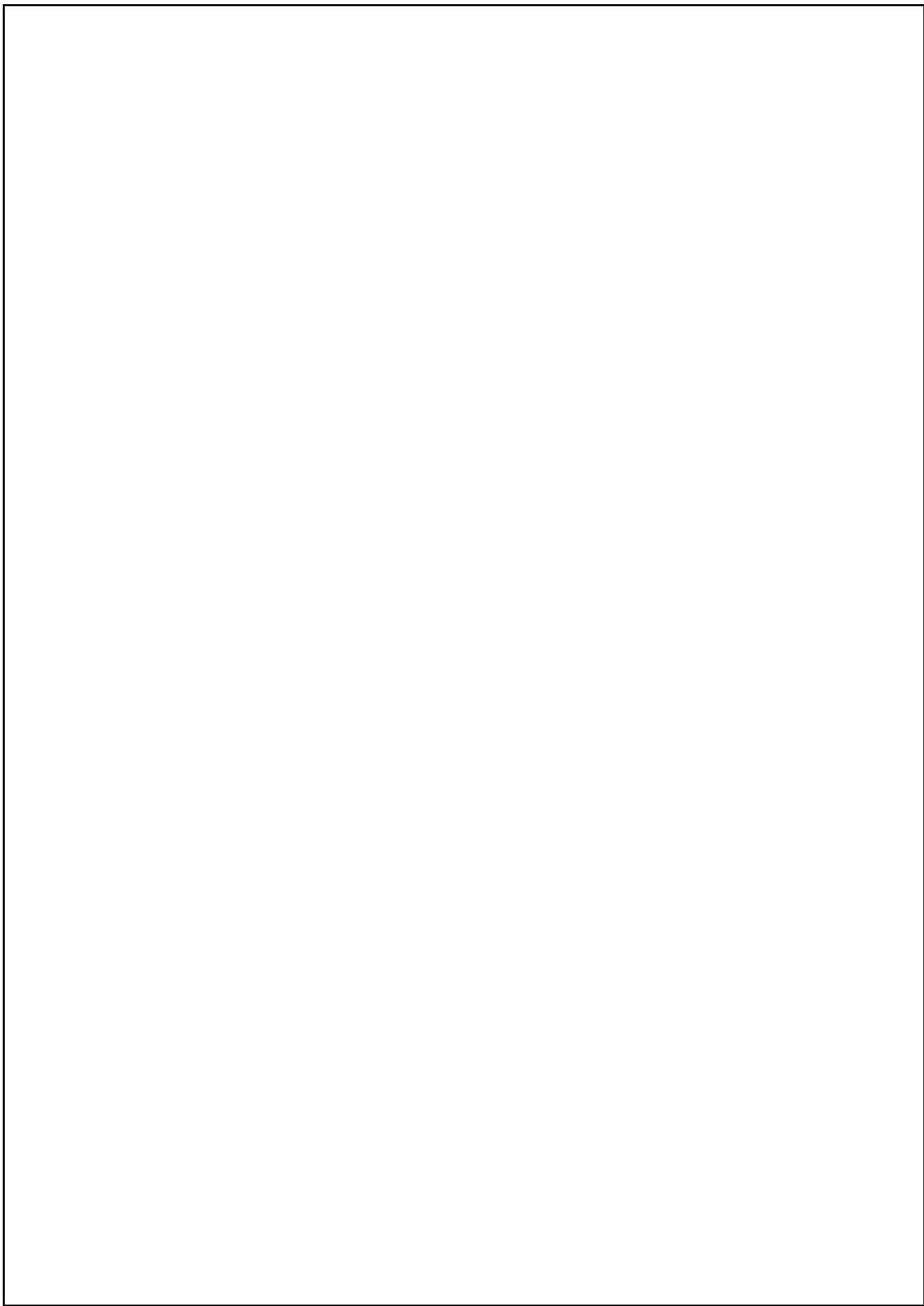
```
Press any key to start....
```

Starting the game again will open and process the highscore.txt file with the updated values and present the new values through the game until they are improved upon.



The screenshot shows a Windows command prompt window titled "C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe". The window displays the same high score table as the previous image, with the text "HighScore Table!" at the top. The table lists five entries: Doi! (1055), H4BPM (200), BPM!! (100), Doi (100), and Morris (10). Below the table, it says "Press any key to start....".

```
C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe  
HighScore Table!  
Name      Score  
Doi!      1055  
H4BPM     200  
BPM!!     100  
Doi       100  
Morris    10  
  
Press any key to start....
```



Part C Testing with negative numbers

Negative numbers are validated within the 'Selection' function for values greater than 24 or less than 1 on input and the user is requested to enter valid numbers

```
C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe

Pair      ****      ****      ****      ****      ****
7         8         9        10        11        12
****      ****      ****      Pair      ****      ****
13        14        15        16        17        18
****      ****      ****      ****      ****      ****
19        20        21        22        23        24
****      Pair      ****      ****      ****      Pair

Please select a Grid Slot between 1 and 24
-6

Please enter a choice between 1 and 24
-7

Please enter a choice between 1 and 24
-8

Please enter a choice between 1 and 24
2

Please select a Grid Slot between 1 and 24
16
```

Part C Testing with boundary values

Selecting grid slots that have been run already will inform the user that the number has already been selected and prompt the user to re-enter a valid number.

```
C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe
Name: Doi!           Highscore: 1055

Score: 280
Well done matching pair

 1      2      3      4      5      6
Pair   Pair   ****   ****   ****   ****
 7      8      9     10     11     12
****   ****   ****   Pair   ****   ****
13     14     15     16     17     18
****   ****   ****   Pair   ****   ****
19     20     21     22     23     24
****   Pair   ****   ****   ****   Pair

Please select a Grid Slot between 1 and 24
1
You have already selected this slot

Please select a Grid Slot between 1 and 24
```

Similarly the game will inform you if the same to grid slots have been chosen twice in the same go.

```
C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe

1      2      3      4      5      6
Pair   Pair   ****   ****   ****   ****
7      8      9      10     11     12
****   ****   ****   Pair   ****   ****
13     14     15     16     17     18
****   ****   ****   Pair   ****   ****
19     20     21     22     23     24
****   Pair   ****   ****   ****   Pair

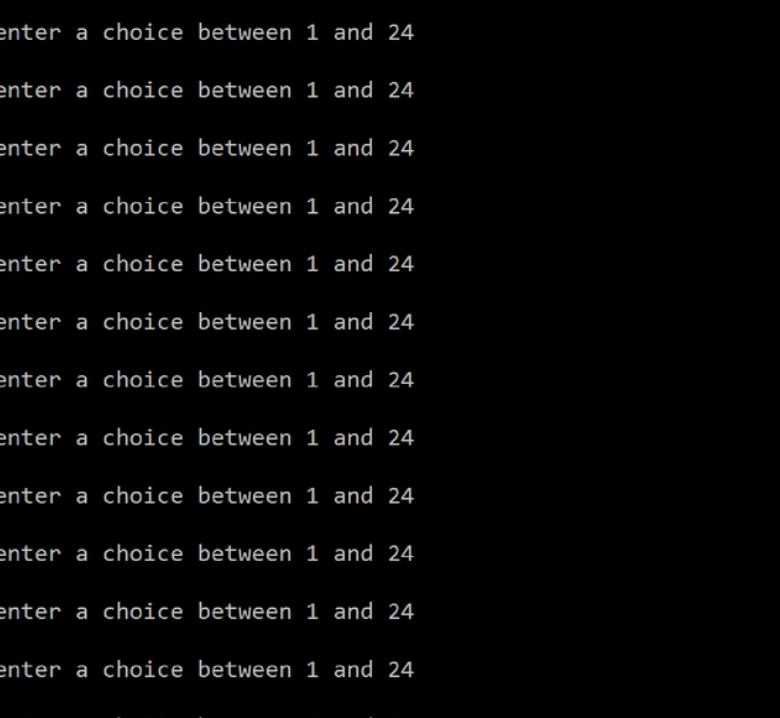
Please select a Grid Slot between 1 and 24
3

Please select a Grid Slot between 1 and 24
3
You selected 1 and 1
You selected the same Gridslot twice please select two different Gridslots

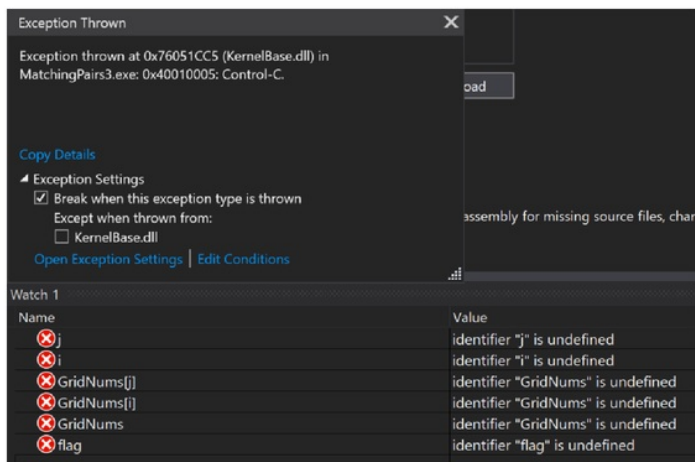
Please select a Grid Slot between 1 and 24
```

Part C Testing with impossible values

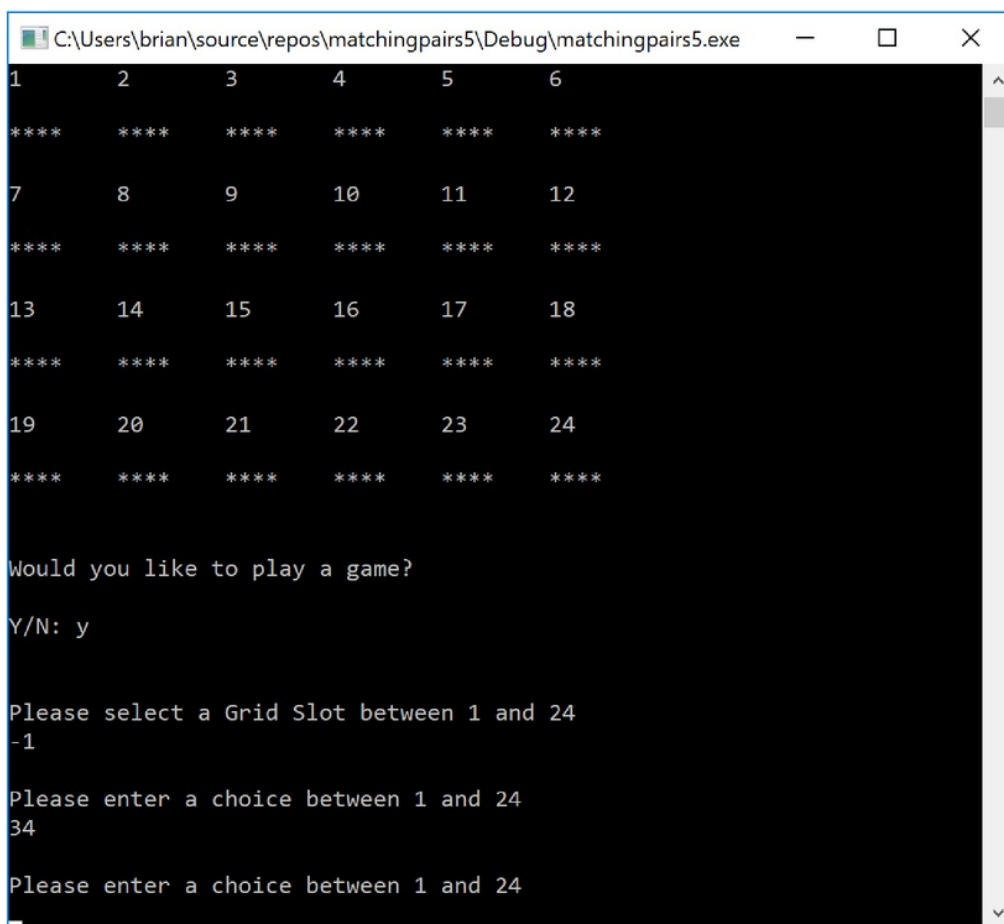
Entering a floating point number caused the application to crash.



The screenshot shows a Windows command prompt window with the title bar "C:\Users\brian\source\repos\matchingpairs5\Debug\matchingpairs5.exe". The window contains 13 identical lines of text, each reading "Please enter a choice between 1 and 24". The text is displayed in a monospaced font on a black background. The window's title bar includes standard Windows controls: a minimize button, a maximize button, and a close button.



Integer numbers larger than the grid numbers or were negative values were captured by the 'Selection' function



Entering a non numerical value again caused an issue and the application would crash out.

[illegible]

Improvements

There are several options that could be added to the code to improve the look and feel.

The scoring mechanism was modified to allow for more flexibility in the scoring by creating a score for matching a pair and taking a mark off for not pairing and removing points every time the player cycles through a turn. There fore the score became a cumulative score not a low score.

The high score table could be increased to store the 10 ten best scores, sorted into descending order improvements were made to the program that included showing the high table at the start of the game and at the end of the game.

The styling of the main game could be improved with more effective use of asci art to 'draw' a more aesthetically pleasing screen that is more representative or the cards this game is trying to mimic.

More work could be used to capture and handle invalid character types to limit the game crashing.

Bibliography

- [1] "how-to-change-text-color-of-cc-console.html," [Online]. Available:
<http://blockofcodes.blogspot.com/2013/06/how-to-change-text-color-of-cc-console.html>.
[Accessed 1 April 2019].
- [2] V. Karunakar, "https://www.geeksforgeeks.org/difference-getchar-getch-getc-getche/,"
[Online]. [Accessed 1 April 2019].
- [3] M. McGrath, "C Programming In Easy Steps 4th Edition," Easy Steps Limited, 2012.

Brian Morris Part C Final

GRADEMARK REPORT

FINAL GRADE

/100

GENERAL COMMENTS

Instructor

PART C

Coding/Annotation- Not entirely efficient - especially the 4 x for loops - couldn't these be combined? They look identical to me. Good use of functions with parameter passing. You could have put the subsequent for loop in the DrawRow() function to - this would have saved you repeating it 4 times. Good validation and a working game produced. 17 /20

Documentation - Breakdown is good, but don't forget to number the stages. 1, 1.1, 1.2, 2, 2.2 etc is helpful. This helps you to be clear in stating which stage to return to when looping. Testing is good overall - be sure to test all boundaries. 13/15

Total 30/35

Overall total for assignment: 55+30

85%

Test mark: 93%

Overall mark for module: 87%

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

PAGE 19

PAGE 20

PAGE 21

PAGE 22

PAGE 23

PAGE 24

PAGE 25

PAGE 26

PAGE 27

PAGE 28

PAGE 29

PAGE 30

PAGE 31

PAGE 32
