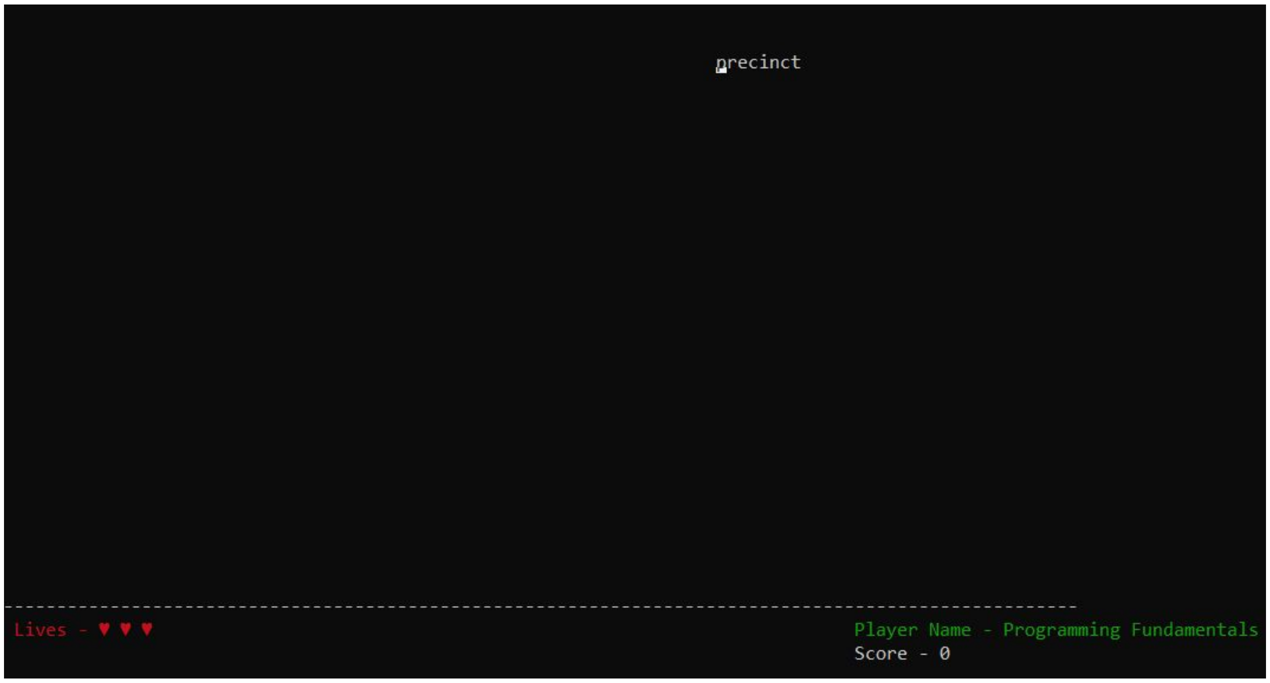
CS-118/CL-118 - PROGRAMMING FUNDAMENTALS

WORD TETRIS

FINAL PROJECT REPORT



```
precinct

-----------------------------------------------------------------------------------------------
Lives - ♥ ♥ ♥                                                Player Name - Programming Fundamentals
                                                             Score - 0
```

Introduction

With Computer Science being an emerging field with everyday developments and many students pursuing the valuable degree, typing fast is becoming a very important requirement. We, therefore, decided to implement a practical program that not only improves one's typing skills but also serves as a vocabulary trainer. From beginner to advanced, this program caters people with different skill levels and trains them to type fast and be quick with their reflexes. It is prepared and presented by the following team members:

- Muhammad Zain Ul Abiddin (20K-1731)
- Fawayya Rafi (20K-0331)
- Kanza Batool (20K-0423)

Background And Research

Early after the formation of project groups, it was an aim to make something that was practically useful to the date. Typing is something that many of us struggle with and want to improve. So, we decided to base our project on this idea. During early research on the project, we came across many games already available on the internet but none like we wanted.

Word Tetris is a program that works sequentially with every instruction executed one after another and faster computer speeds enable it to function fluidly. The initial implementation of this program was with arrays but it was a lost cause due to the constant flickering which greatly impacted the user experience. The program is built to work on Windows only and will not be supported by Mac and Linux due to the use of Windows Cursor library. However, it can be scaled to work with other operating systems by implementation of their libraries. Windows 7 does not natively support ANSI color codes and hence, the colors would not work with Windows 7.

It was also one of the top priorities to implement every programming technique that we have learned over the course of this semester. Word Tetris achieves that beautifully with most of the content covered.

Project Specification

Tools and technologies used:

- Dev C++ was used as the primary IDE for this project while VSCode was used interacting with the GitHub repository
- Three text files act as the database for this project; **wordlist.txt/bonus.txt** stores the words used in the program and **test.txt** stores the highscores
- The following libraries were used:
    - stdio.h (standard C library)
    - stdlib.h (standard C library)
    - time.h (time library used to implement the delay function)
    - conio.h (console input/out library used to implement the kbhit function)

    - string.h (string library for character array manipulation)
    - windows.h (library for Windows API, handles cursor commands and system console commands)

Programming Concepts used:

- Functions
- Pointers
- File Handling
- Structures (Structs)
- String/Character Manipulation
- Sorting
- Loops/Iterations
- Randomization
- ANSI Color Codes (Escape Sequences)

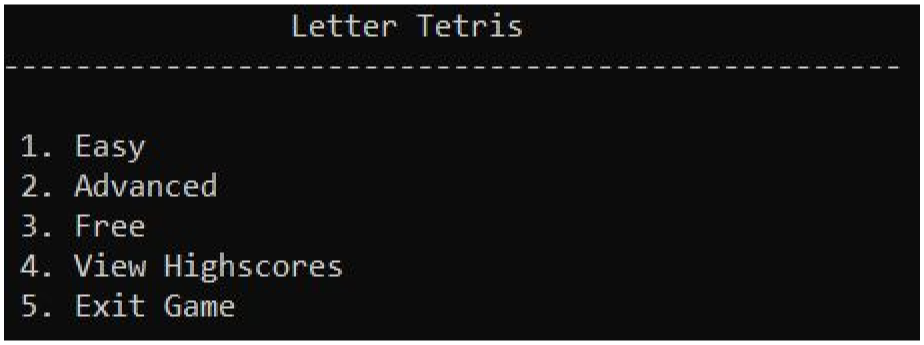
Problem Analysis

One of the major objectives of every project is to create an approach towards the required goal. In this project, getting the words to fall was the most important part. Initially, the approach was made using arrays and printing them using nested loops. However, this method was not suitable for a game like this as it produced a lot of flickering. The windows cursor library was then used to solve the problem and print only the required amount of data on the console screen with the luxury of clearing it whenever needed.

The second main objective was to ensure competitiveness in a game and for that purpose a database was needed in order to store the high scores by each user. This was implemented using a text file. Binary filing is an alternative if the program needs to be published commercially

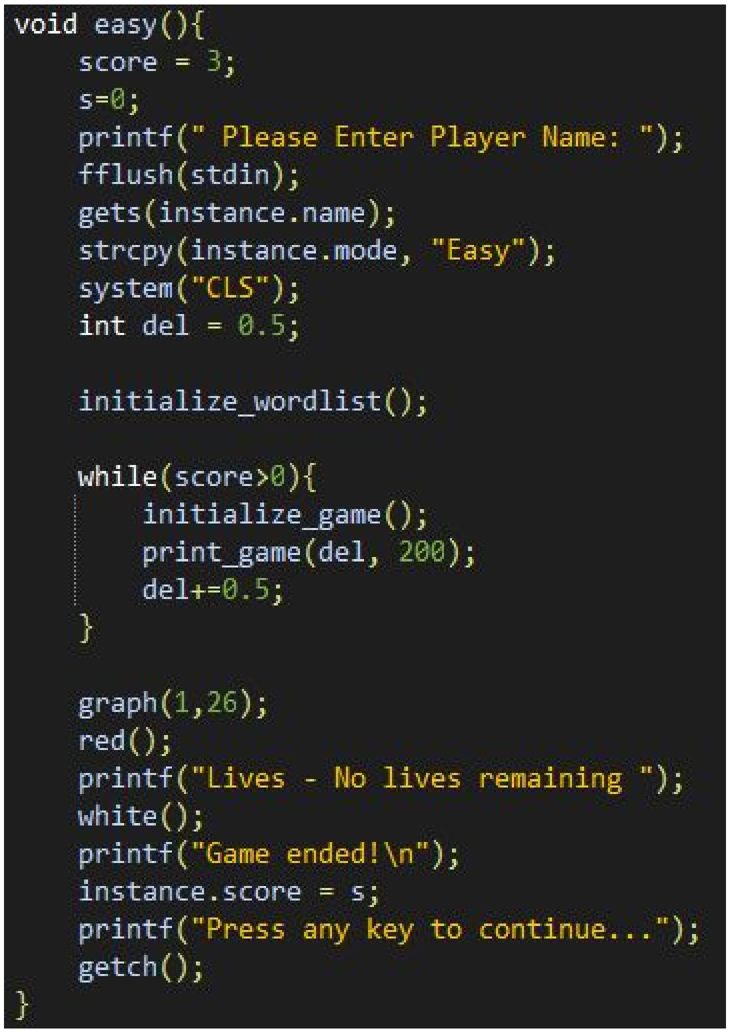
The next page features Solution Design.

Solution Design



```
                 Letter Tetris
-----------------------------------------------------

 1. Easy
 2. Advanced
 3. Free
 4. View Highscores
 5. Exit Game
```

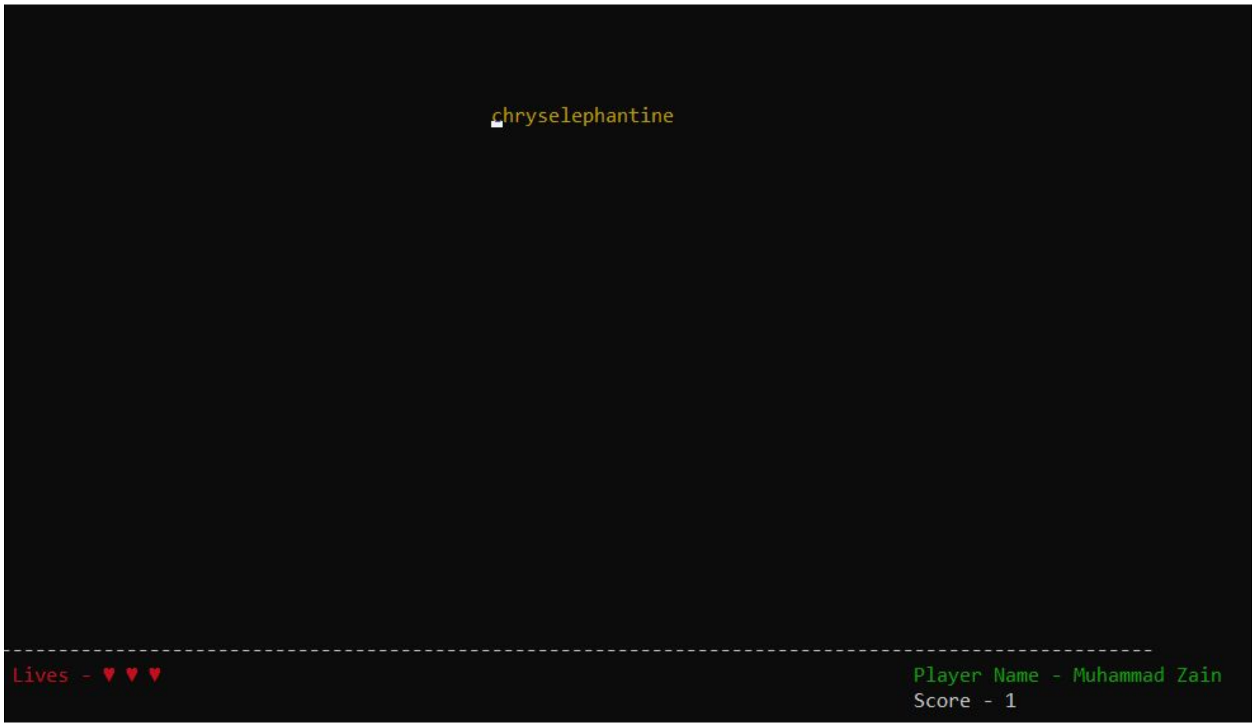
The game features three modes for users with different levels of skills. Basic code is the same for all three modes with minor modifications.



```
void easy(){
    score = 3;
    s=0;
    printf(" Please Enter Player Name: ");
    fflush(stdin);
    gets(instance.name);
    strcpy(instance.mode, "Easy");
    system("CLS");
    int del = 0.5;

    initialize_wordlist();

    while(score>0){
        initialize_game();
        print_game(del, 200);
        del+=0.5;
    }

    graph(1,26);
    red();
    printf("Lives - No lives remaining ");
    white();
    printf("Game ended!\n");
    instance.score = s;
    printf("Press any key to continue...");
    getch();
}
```

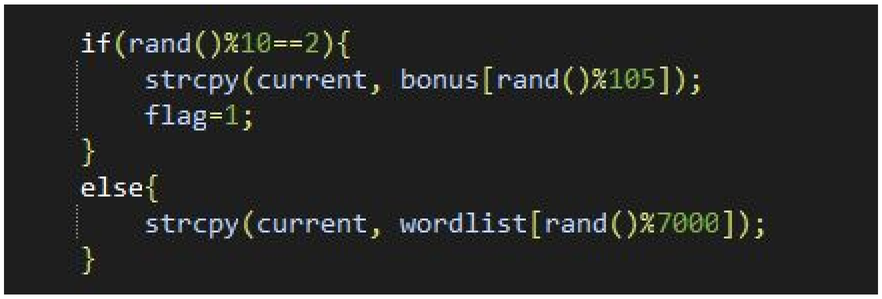
Easy Mode is similar to advanced mode with a delay step of 200 milliseconds. It should be noted that the speed of words falling increases with 0.5 milliseconds each time the user hits enter. Each user is allowed three lives per mode after which the game ends and returns to the main menu. Programming is modular and there are many functions that are interconnected to each other. Advanced mode starts the drop from a speed of 130 milliseconds and is best suited for users with some experience with this programming or other similar typing programs. The free mode is for practice as the user loses no lives and can play for as long as they want.

Every game has a bonus mechanic in place for its users. In this program, the bonus appears in the form of long English words that cost no life but give the player points that are worth their length. These words are stored in the **bonus.txt** file.



```
chryselephantine

Lives - ♥ ♥ ♥                                                    Player Name - Muhammad Zain
                                                                 Score - 1
```

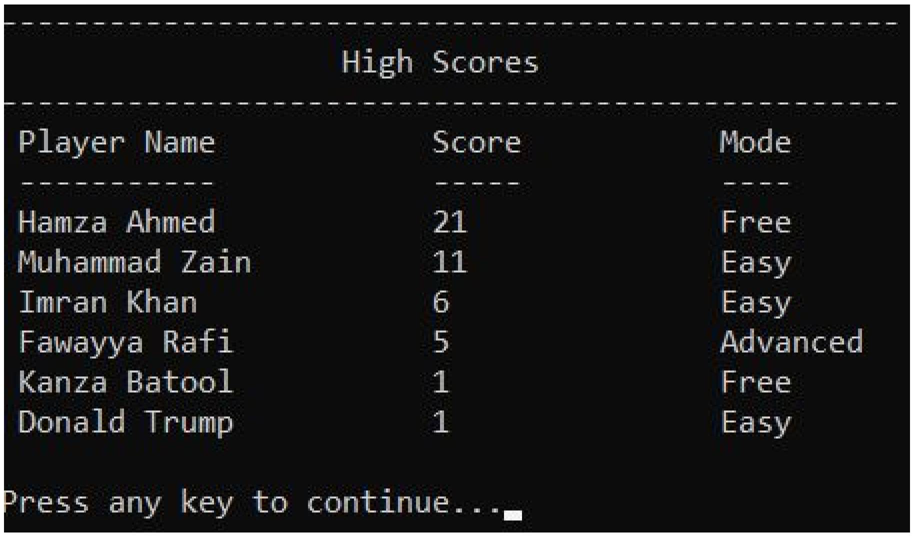
For these words to appear, a randomizer function uses basic probability (10% in this case) to determine how often these words would appear. This concept of rarity is the basic functionality that we find behind many popular games such as Fortnite and PUBG where the amount of loot or a certain gun is programmed in by the creators and they are spawned accordingly. In simple words, each time a word spawns, there is a 10 percent chance that it is a bonus word.



```
if(rand()%10==2){
    strcpy(current, bonus[rand()%105]);
    flag=1;
}
else{
    strcpy(current, wordlist[rand()%7000]);
}
```

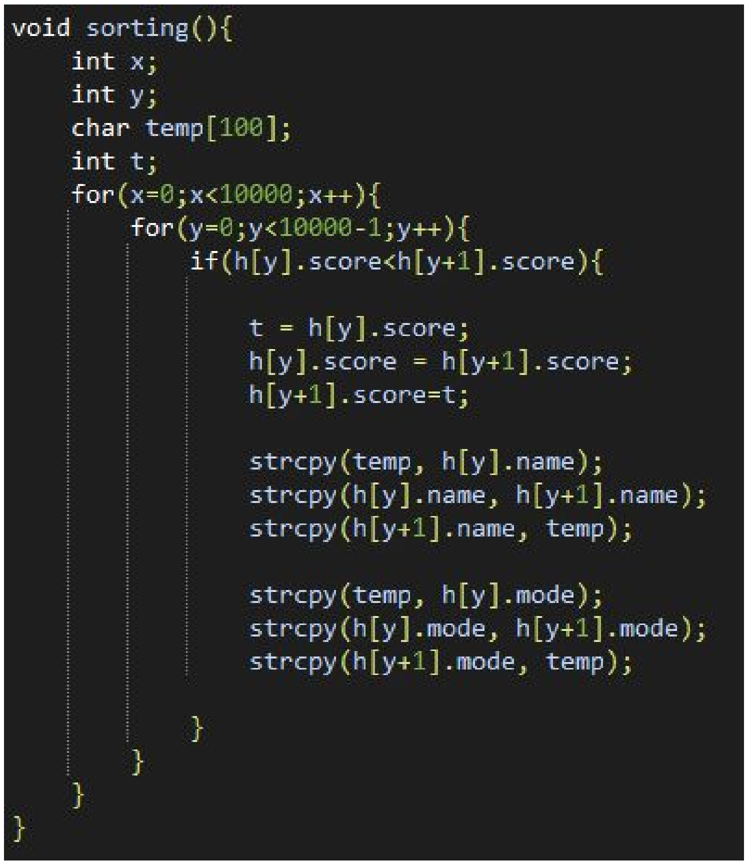
^^code block for randomization

To ensure a competitive environment, a feature to view highscores was also added.

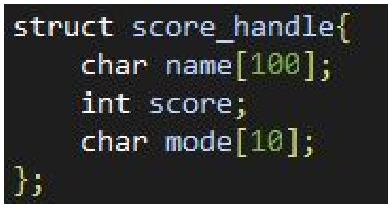


```
-------------------------------------------------
                  High Scores
-------------------------------------------------
 Player Name           Score          Mode
 -----------           -----          ----
 Hamza Ahmed           21             Free
 Muhammad Zain         11             Easy
 Imran Khan            6              Easy
 Fawayya Rafi          5              Advanced
 Kanza Batool          1              Free
 Donald Trump          1              Easy

Press any key to continue...
```

The implementation for this feature requires sorting. Bubble sorting was used to manipulate the main structure array in this program.



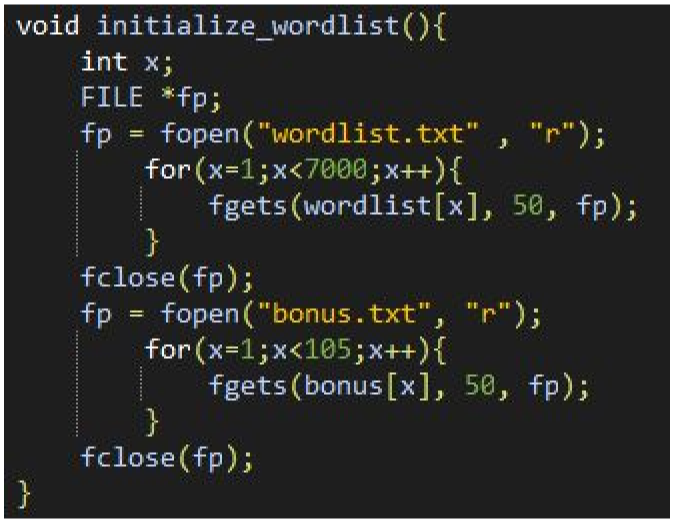
```
void sorting(){
    int x;
    int y;
    char temp[100];
    int t;
    for(x=0;x<10000;x++){
        for(y=0;y<10000-1;y++){
            if(h[y].score<h[y+1].score){

                t = h[y].score;
                h[y].score = h[y+1].score;
                h[y+1].score=t;

                strcpy(temp, h[y].name);
                strcpy(h[y].name, h[y+1].name);
                strcpy(h[y+1].name, temp);

                strcpy(temp, h[y].mode);
                strcpy(h[y].mode, h[y+1].mode);
                strcpy(h[y+1].mode, temp);

            }
        }
    }
}
```



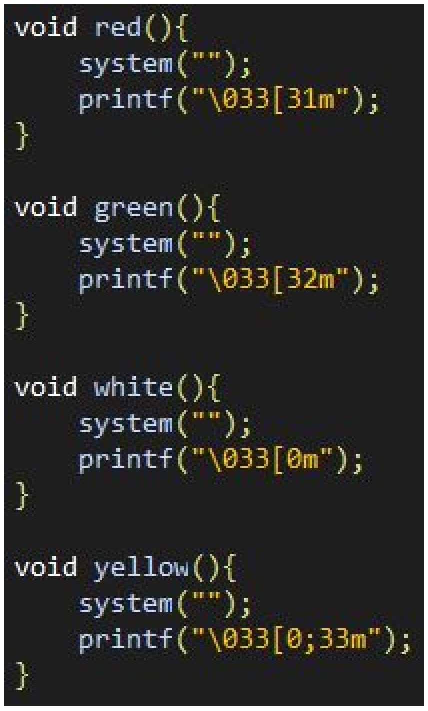
```c
struct score_handle{
    char name[100];
    int score;
    char mode[10];
};
```

The program uses one structure to manage the entire database and its running instance. This structure has two objects; **struct score_handle instance; struct score_handle h[10000];**

Arrays are used in order to read and update data from the text files. The function **initialize_wordlist()** uses file pointers to read words from the text files and store them in their respective arrays. This function is called every time a mode is selected to read if any new words have been added to the file by the user.

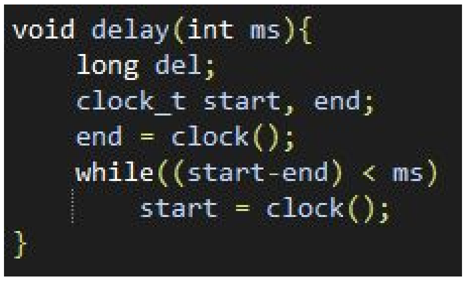


```c
void initialize_wordlist(){
    int x;
    FILE *fp;
    fp = fopen("wordlist.txt" , "r");
        for(x=1;x<7000;x++){
            fgets(wordlist[x], 50, fp);
        }
    fclose(fp);
    fp = fopen("bonus.txt", "r");
        for(x=1;x<105;x++){
            fgets(bonus[x], 50, fp);
        }
    fclose(fp);
}
```



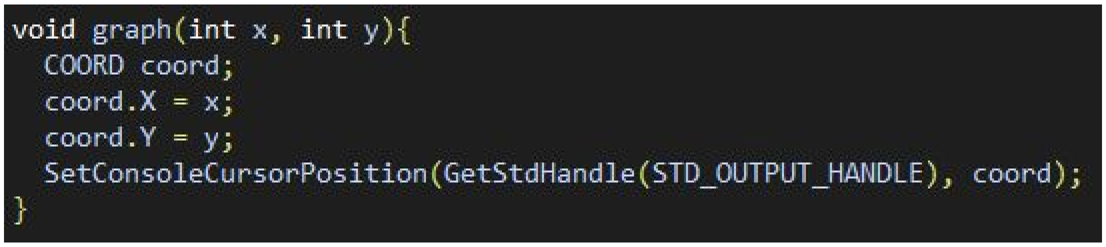
```c
void red(){
    system("");
    printf("\033[31m");
}

void green(){
    system("");
    printf("\033[32m");
}

void white(){
    system("");
    printf("\033[0m");
}

void yellow(){
    system("");
    printf("\033[0;33m");
}
```

Color coding is done using ANSI colors codes that are available natively with Windows 10 as part of the terminal. These codes also work with Mac and Linux operating systems. Four functions are used throughout the code block to manipulate the colors in the console whenever needed. Printing these codes in the console window changes the color of the console and **white()** has to be used every time one wants to revert back to the original console color.



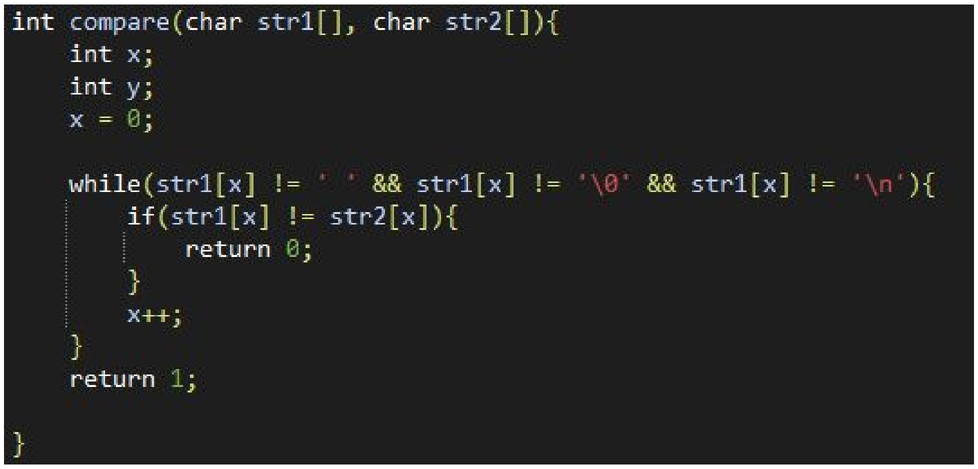
```
void delay(int ms){
    long del;
    clock_t start, end;
    end = clock();
    while((start-end) < ms)
        start = clock();
}
```

Delay function is a very important part of this program and it is linked with the time library that fetches the local time and pauses the program till the difference between the start and end time is not equal to the time preset for each mode.



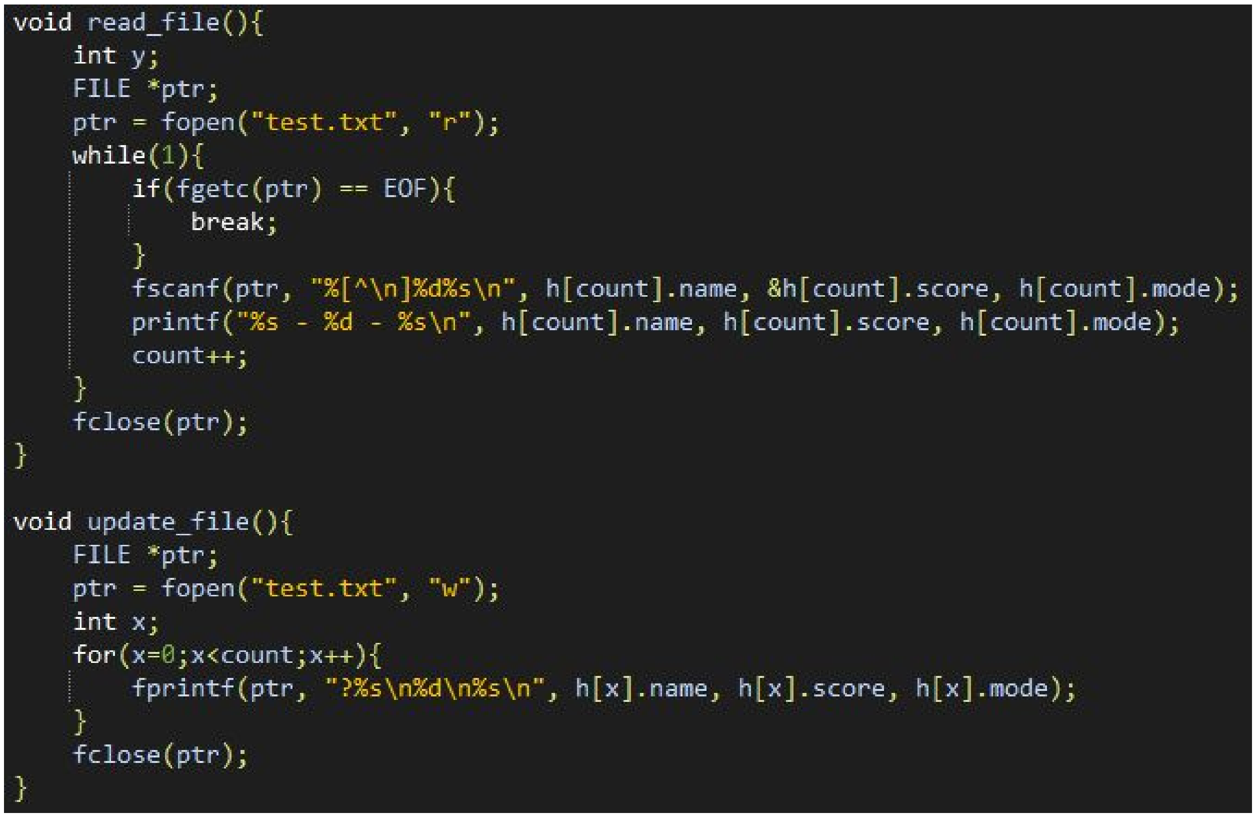
```
void graph(int x, int y){
  COORD coord;
  coord.X = x;
  coord.Y = y;
  SetConsoleCursorPosition(GetStdHandle(STD_OUTPUT_HANDLE), coord);
}
```

The graph function is associated with the **windows** API and is used to manipulate the cursor in order to print text at whatever position required by the program. This function is the primary reason why this program will not work with any other operating system other than Windows. Mac has a similar library called **cursor** and functions from that library can be used to interact with the Mac terminal and scale this program to other systems.



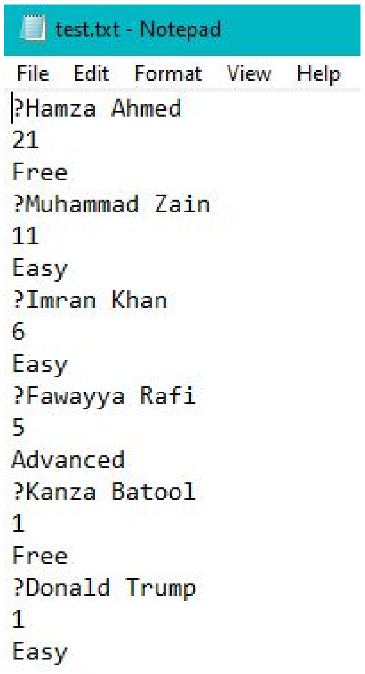
```
int compare(char str1[], char str2[]){
    int x;
    int y;
    x = 0;

    while(str1[x] != ' ' && str1[x] != '\0' && str1[x] != '\n'){
        if(str1[x] != str2[x]){
            return 0;
        }
        x++;
    }
    return 1;

}
```

Compare is a user defined function made specifically for this program that checks if the typed word is correct after the user presses enter.



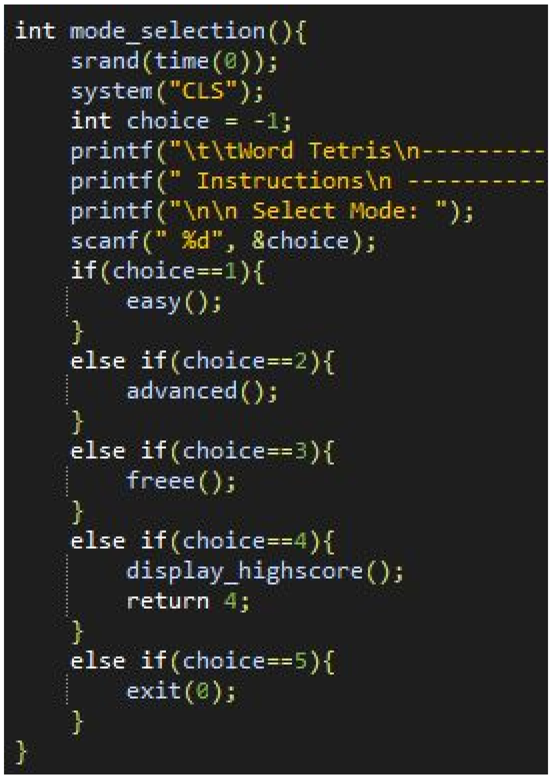
```
void read_file(){
    int y;
    FILE *ptr;
    ptr = fopen("test.txt", "r");
    while(1){
        if(fgetc(ptr) == EOF){
            break;
        }
        fscanf(ptr, "%[^\n]%d%s\n", h[count].name, &h[count].score, h[count].mode);
        printf("%s - %d - %s\n", h[count].name, h[count].score, h[count].mode);
        count++;
    }
    fclose(ptr);
}

void update_file(){
    FILE *ptr;
    ptr = fopen("test.txt", "w");
    int x;
    for(x=0;x<count;x++){
        fprintf(ptr, "?%s\n%d\n%s\n", h[x].name, h[x].score, h[x].mode);
    }
    fclose(ptr);
}
```

The functions **read_file()** and **update_file()** are designed to update the file whenever the user exits a certain mode. They follow a specific design in order to read and write data to the file.



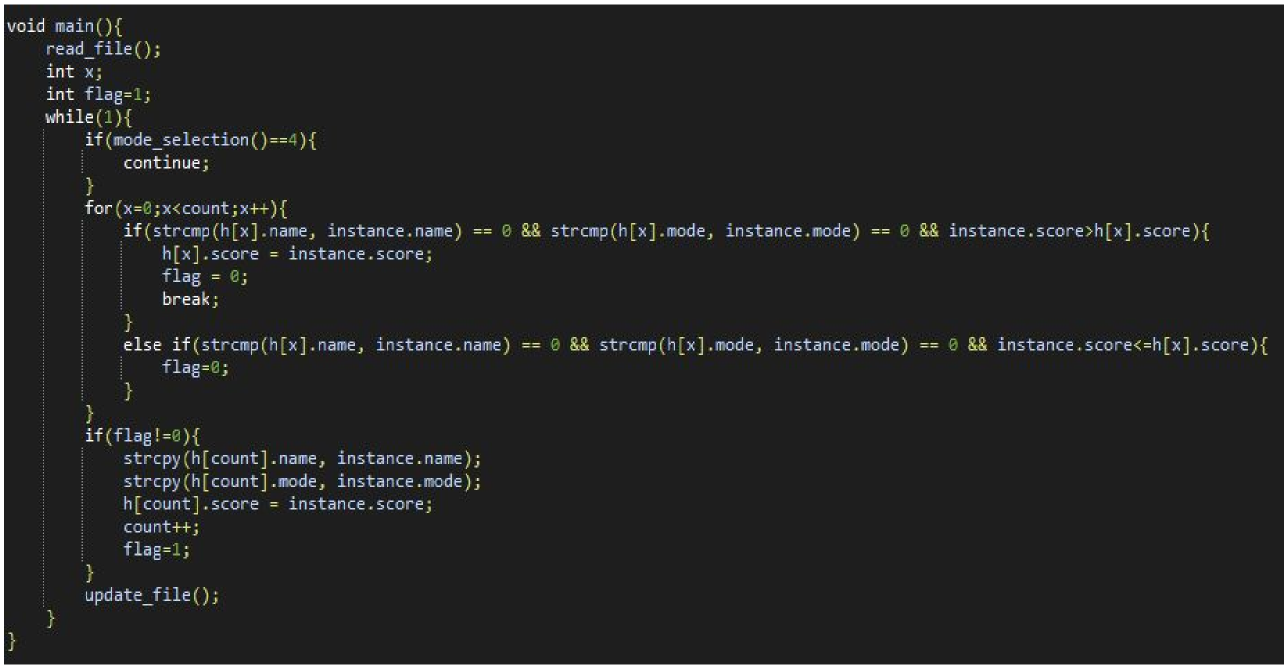
```
?Hamza Ahmed
21
Free
?Muhammad Zain
11
Easy
?Imran Khan
6
Easy
?Fawayya Rafi
5
Advanced
?Kanza Batool
1
Free
?Donald Trump
1
Easy
```

Each new user data is appended with a '**?**' character at the start of their name which helps the program detect End of File and separation of data.



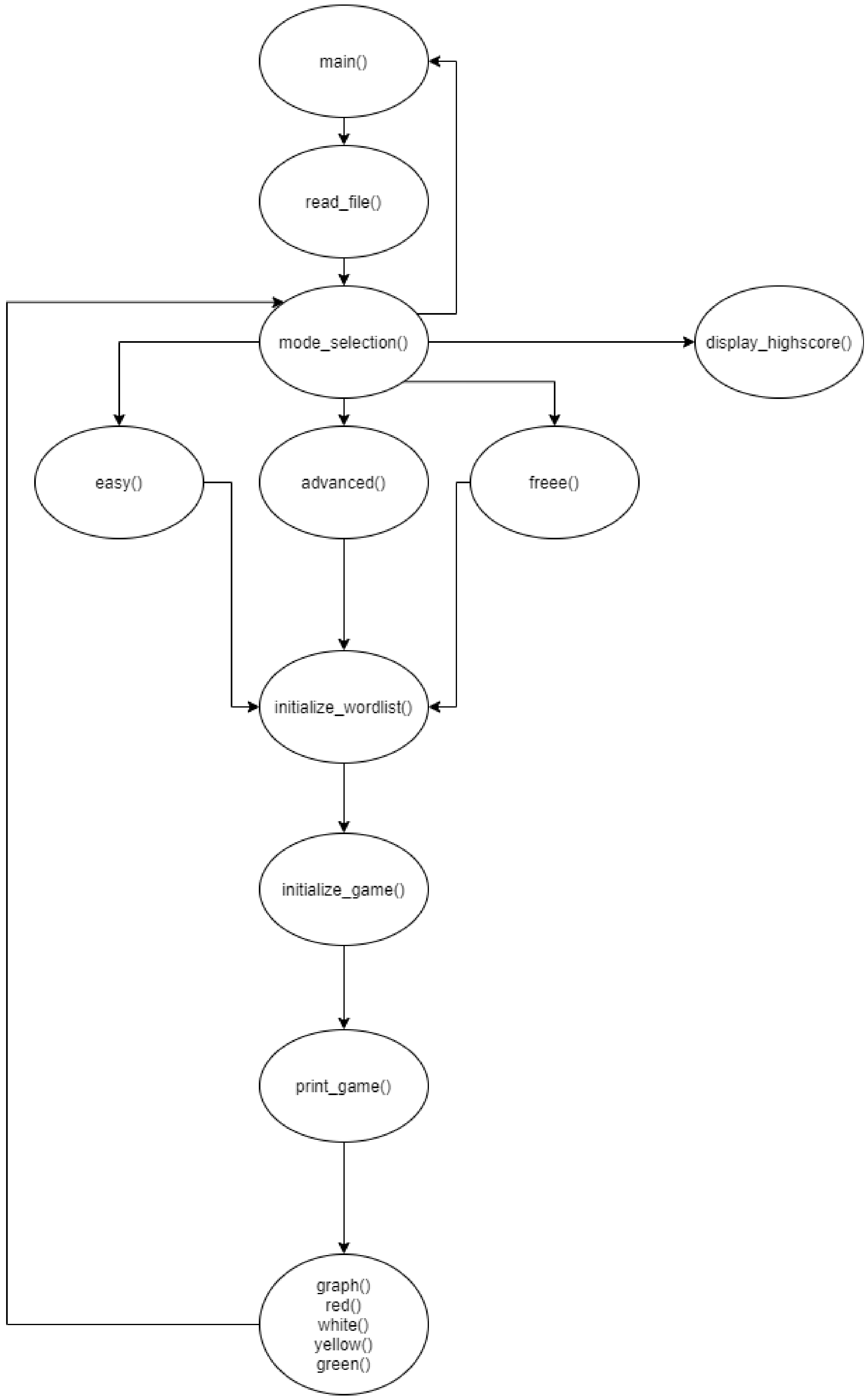
```c
int mode_selection(){
    srand(time(0));
    system("CLS");
    int choice = -1;
    printf("\t\tWord Tetris\n---------
    printf(" Instructions\n ----------
    printf("\n\n Select Mode: ");
    scanf(" %d", &choice);
    if(choice==1){
        easy();
    }
    else if(choice==2){
        advanced();
    }
    else if(choice==3){
        freee();
    }
    else if(choice==4){
        display_highscore();
        return 4;
    }
    else if(choice==5){
        exit(0);
    }
}
```

The **mode_selection()** function allows the user to navigate through the UI of this console program. It is called on by the **main()** function every time the user exits a mode.



```c
void main(){
    read_file();
    int x;
    int flag=1;
    while(1){
        if(mode_selection()==4){
            continue;
        }
        for(x=0;x<count;x++){
            if(strcmp(h[x].name, instance.name) == 0 && strcmp(h[x].mode, instance.mode) == 0 && instance.score>h[x].score){
                h[x].score = instance.score;
                flag = 0;
                break;
            }
            else if(strcmp(h[x].name, instance.name) == 0 && strcmp(h[x].mode, instance.mode) == 0 && instance.score<=h[x].score){
                flag=0;
            }
        }
        if(flag!=0){
            strcpy(h[count].name, instance.name);
            strcpy(h[count].mode, instance.mode);
            h[count].score = instance.score;
            count++;
            flag=1;
        }
        update_file();
    }
}
```

The **main()** function brings it all together by joining all functions with each other. The next page shows a hierarchical flowchart of this program with all the functions used.


main()
read_file()
mode_selection()
display_highscore()
easy()
advanced()
freee()
initialize_wordlist()
initialize_game()
print_game()
graph()
red()
white()
yellow()
green()

Implementation and Testing

The program was implemented using basic programming techniques and has been tested to filter out any bugs that might have been a part of the process. The program has been tested rigorously by all three members.

During testing, a bug was identified with scoring. Due to the program's dependence on loops, we cannot subtract scores. Therefore, any subtraction of scores was removed from the program.

Project Breakdown Structure

Timeline:

● 25th December 2020	-	Research
● 30th December 2020	-	Logic Design
● 10th January 2021	-	Coding Phase Begins
● 15th January 2021	-	Coding Phase Ends
● 20th January 2021	-	Testing
● 21st January 2021	-	Report
● 23rd January 2021	-	Submission Date

Workload Breakdown:

- Research - Zain
- Logic and Design - Zain, Kanza, Fawayya
- Coding - Zain, Kanza, Fawayya
- Testing - Kanza, Fawayya

The coding phase was the longest and most important phase during implementation of this project. All the functions were divided amongst members and a collaborative approach was applied while completing the project.

Results

The program works as intended and has exclusively been rated by batchmates as really helpful when it comes to typing speeds.

Conclusion

https://github.com/M1keZulu/WordGame-CS-118-Programming-Fundamentals

Future work:

Since the program is designed to be scalable thanks to modular programming, many new features can be added. One feature that is under consideration involves storing the words shown in the program and then outputting them at the end with their meanings in order to contribute to the vocabulary of the user. Another feature under consideration is to race against time and type a given set of words.