Final Project

Data Structure

Project Name**: Wordle**

Name**:**

Muhammad Alfin Rizqullah - 2502036842

Justin Theofilus Yonathan - 2502036382

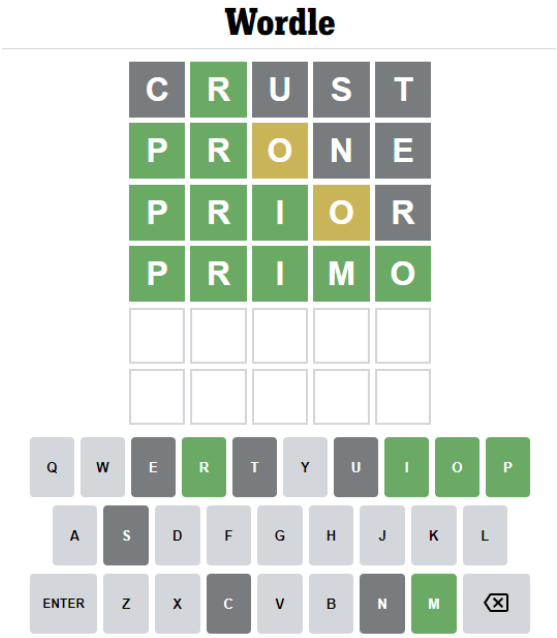
Ian Wirawan - 2502009596

Class**: L2BC**

Course Code**: COMP6048001**

Problem Description

For our group's final project we had a couple of options to choose from. Initially, we had a quiz program and a crossword game in mind, but later on, decided to choose another idea which was Wordle. Wordle is a popular internet word guessing game that provides a simplistic but ingenious design. In order to play wordle, the user is given a 5x6 table which represents the word length and number of attempts respectively, within 6 attempts the user is given the chance to guess the word of the day. If the user guesses a word containing correct letters in the correct position it will then turn that specific tile to the color green. On the other hand, if the user were to input a word containing correct letters but its position is incorrect it will then turn the tile yellow. Finally, if the user were to input a word containing incorrect letters then the tile will turn grey.



Our group planned on making a similar GUI complement the code, but due to some issues we had with the GUI program, we ended up changing it last minute to a CLI program instead. All in all, despite a few minor issues we had, we managed to finish the working code for the game.

Code Implementation

```
    //Initial Promt Welcoming Players
cout<<"Welcome to WORDLE!! "<<"\n"<<endl;

while(run){
    //Checking if attempts are used up
    if(attempts < 1){
            //Game Over Prompt
            cout<<" "<<endl;
            cout<<"GAME OVER, Attempts left: "<<attempts<<endl;
            break;
    }
    //Prompt Asking Players to Enter their Inputs
    cout<<" "<<endl;
    cout<<"You have "<<attempts<<" attempts left."<<endl;
    cout<<"Enter Your Answer(Five Letters): "<<endl;
    //Requesting User Input
    cin>> ans1;
    cin>> ans2;
    cin>> ans3;
    cin>> ans4;
    cin>> ans5;
    //Turning Initial Chars Into Uppercase Chars
    char char1 = toupper(ans1);
    char char2 = toupper(ans2);
    char char3 = toupper(ans3);
    char char4 = toupper(ans4);
    char char5 = toupper(ans5);
    cout<<" "<<endl;
```

Accepting Input

This block of code job is to accept any inputs that the user would enter. It would be able to split a string and capitalize each letter so it would match the answer key.

```
//Storing User Input In an Array
char concat [5] = {char1,char2,char3,char4,char5};

//Error Checking For:
//1st letter
if(key[0] == char1){
        cout<<char1<<" Exists in word, Position correct."<<endl;

}else if(char1 == key[1] || char1 == key[2] || char1 == key[3] || char1 == key[4]){
        cout<<char1<<" Exists in word, Position incorrect"<<endl;
}else {
        cout<<char1<<" Does not exist in word"<<endl;
}
//2nd letter
if(key[1] == char2){
    cout<<char2<<" Exists in word, Position correct."<<endl;

}else if(char2 == key[0] || char2 == key[2] || char2 == key[3] || char2 == key[4]){
        cout<<char2<<" Exists in word, Position incorrect"<<endl;
}else {
        cout<<char2<<" Does not exist in word"<<endl;
}
//3rd letter
if(key[2] == char3){
    cout<<char3<<" Exists in word, Position correct."<<endl;

}else if(char3 == key[0] || char3 == key[1] || char3 == key[3] || char3 == key[4]){
        cout<<char3<<" Exists in word, Position incorrect"<<endl;
}else {
        cout<<char3<<" Does not exist in word."<<endl;
}
```

Error Checking

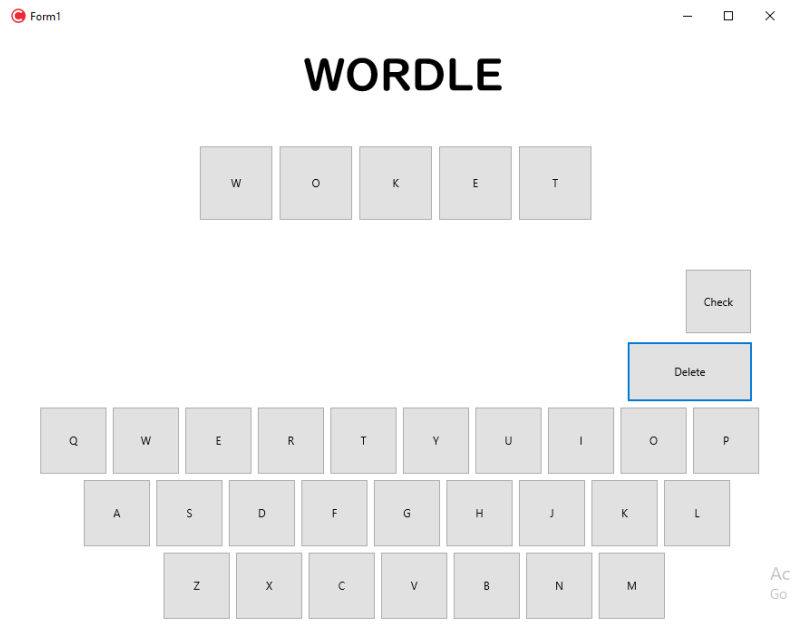
With the string being split, the characters go to an error-checking loop to figure out which letter matches the answer key. It will loop 6 times.

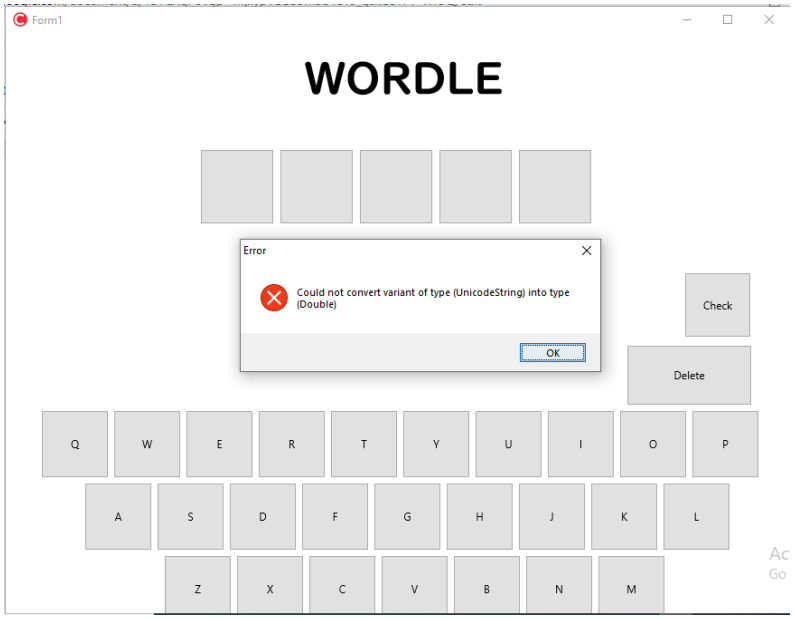
```
//4th letter
if(key[3] == char4){
    cout<<char4<<" Exists in word, Position correct."<<endl;

}else if(char4 == key[0] || char4 == key[1] || char4 == key[2] || char4 == key[4]){
        cout<<char4<<" Exists in word, Position incorrect."<<endl;
}else {
        cout<<char4<<" Does not exist in word."<<endl;
}
//5th letter
if(key[4] == char5){
    cout<<char5<<" Exists in word, Position correct."<<endl;

}else if(char5 == key[1] || char5 == key[2] || char5 == key[3] || char5 == key[0]){
        cout<<char5<<" Exists in word, Position incorrect."<<endl;
}else {
        cout<<char5<<" Does not exist in word."<<endl;
}
//Stating Winning Conditions
if(key[0] == char1 && key[1] == char2 && key[2]==char3 && key[3]==char4 && key[4] == char5){
        //Winning Prompt
        cout<<"YOU WON!! CONGRATULATIONS."<<endl;
        break;
}
attempts --;
```

```
//Initializing Variables and Arrays
int attempts = 6;
bool win = false;
bool run = true;
char ans1,ans2,ans3,ans4,ans5;
string AnsKey[] =  {"FROTH", "FLUID", "MONKE","GOOSE","ATONE","MOUTH","PLAIN"};
array<int,7> s= {0,1,2,3,4,5,6};
unsigned seed = 1;

//Function to Randomize Answer Key
string selectAnsKey(){
        shuffle(s.begin(),s.end(),default_random_engine(seed));
        return AnsKey[s[4]];
}
//Assigning Result of Randomization to a Variable
string key = selectAnsKey();
```

Initializing

Attempts are initialized for the for loop to limit how many attempts the user can have, this line of code also has the boolean win condition. It also houses the answer randomly but it does not seem to work properly.


Form1
WORDLE
W
O
K
E
T
Check
Delete
Q
W
E
R
T
Y
U
I
O
P
A
S
D
F
G
H
J
K
L
Z
X
C
V
B
N
M


Form1
WORDLE
Error
Could not convert variant of type (UnicodeString) into type (Double)
OK
Check
Delete
Q
W
E
R
T
Y
U
I
O
P
A
S
D
F
G
H
J
K
L
Z
X
C
V
B
N
M

Alternative Data Structures and Theoretical Analysis

For this project our data structure of choice is Array, we chose this data structure specifically because of its simplicity in storing multiple elements. Originally we were planning to also create a linked-list variant of this game, but due to spending soo much time trying to troubleshoot the GUI portion of the previous variant of this game, we ran out of time. In the event that we did create a linked-list variant of this project, we theorize that it would run slower but uses up less memory. We come to this conclusion because a linked list does not have the capability to access elements randomly, only sequentially it can search with. Meanwhile, the array has the upside of being faster at search because it can simply point at an index to access it, but of course, it will take up more memory.

Manual (How to use)

Since our program is a CLI based program, it is quite simple to run. When the user runs the code the program will give a text saying “Welcome to WORDLE”, “You have 6 attempts left.”, “Enter your answer (Letter by letter): “. After the user inputs the first word and presses enter, it will then show them whether the word contains the correct letters in correct position, correct letters in the wrong position, or if it doesnt have any correct letters. It will then loop back to the same prompt as when the code first ran except this time it will say “You have 5 attempts left” instead of 6 attempts. When the number of attempts reach 0 it will show a prompt saying “GAME OVER, Attempts left: 0” and then end the program. If the user guesses all 5 letters correctly in the correct positions then it is supposed to show a text saying “YOU WON!! CONGRATULATIONS.” and then end the game as well.

Result

At the end of this project, our program has successfully fulfilled our criteria in terms of how the application work. With it, we are able to error check and distinguish which letters are in the answer key but not in the right position or simply do not exist in the key at all. It is also convenient for the user because it allows them to simply input the entire word, instead of one-by-one, while also marking each letter for the error-checking. Although so, we have to state that our program is not completely bug-free, we have an issue where the program registers more than 5 letter inputs as a valid answer and we don’t know how to stop this from happening. Another bug in our program lies within the randomization algorithm which doesn’t work for reasons that we don’t know. To add to the matter, since we spent most of our time figuring out how the GUI program worked we were pressed for time, which made us unable to solve these bugs for the moment. Although we had some issues and bugs, we are able to create a working program that represents the actual Wordle game to a certain extent and because of that we are satisfied with our results.