GRADE 11 PORTFOLIO PROJECT

Krishna Patel

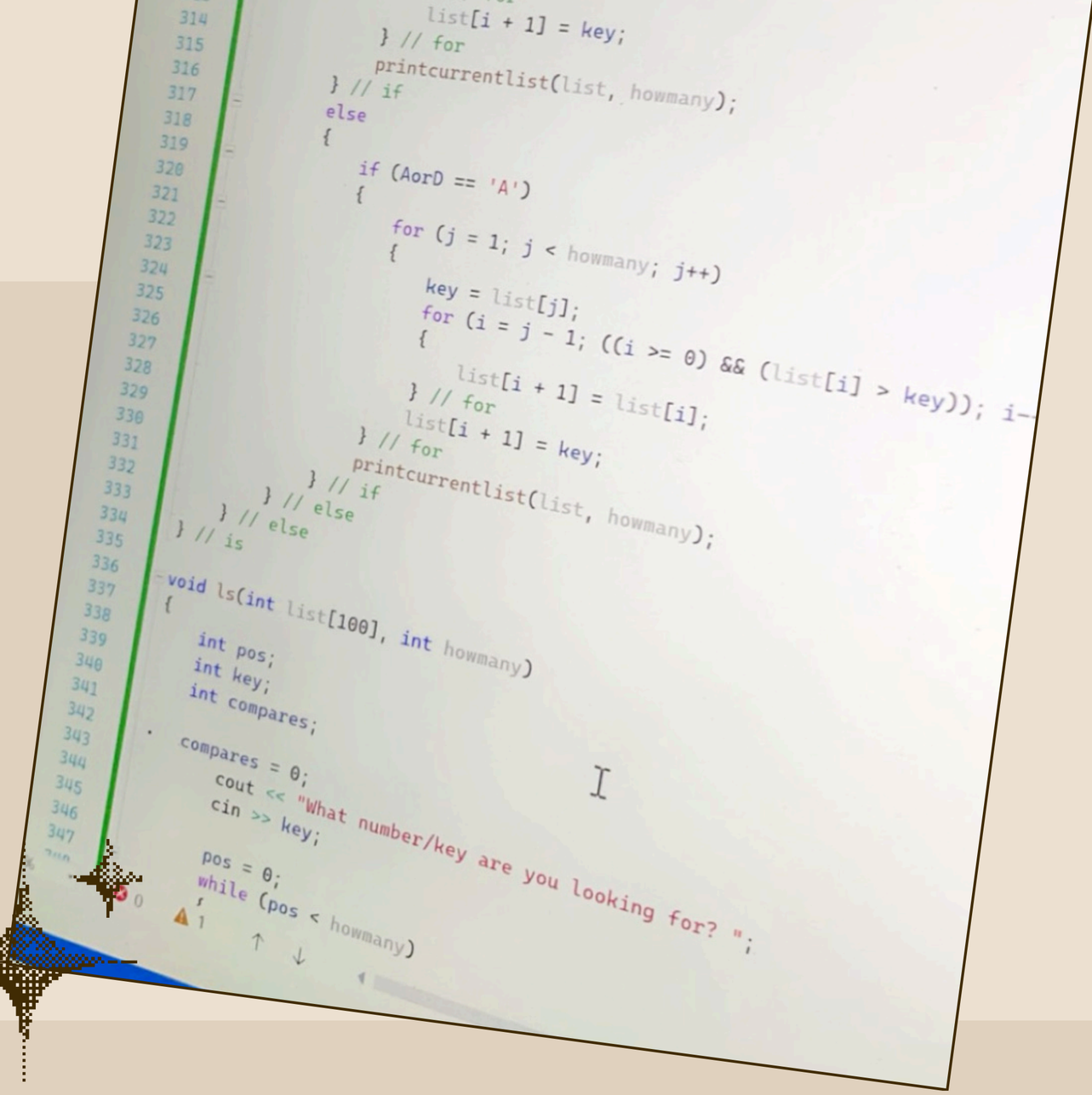
PROJECT OVERVIEW

This project is an advanced exploration of computer programming, building on foundational skills acquired in previous coursework. The focus of this portfolio project was to master higher-level programming concepts such as subprograms, parameters, arrays, strings, and searching/sorting algorithms in the C++ programming language.

BACKGROUND

Within a previous course, I was introduced to the basics of computer programming, primarily in the C++ programming language. This foundation included learning about input/output operations, decision-making structures, loops, and an introduction to subprograms. These concepts helped provide me a strong base for understanding how to write efficient and functional code.

In my Grade 11 Portfolio Project, I build on this foundation by exploring more advanced programming techniques. Delving into subprograms and parameters, I've learned to create modular and reusable code. The study of arrays and strings has deepened my ability to efficiently handle and manipulate data, while mastering searching and sorting algorithms has provided me with the tools to optimize data processing tasks.



ASSIGNMENT DETAILS

Computer Science 30S
C++ Assignment #1: *Parameters:*

#1 Since this is such a GIANT assignment (and in honour of the creators of "Older" and the They Might Be Giants Deck of Cards), write a program that asks the user to enter 5 letters (error check this to make sure they do!) and then prints out the GIANT versions of the letters vertically down the page. For example if the user enters the letters S, C, O, T, and T, it would print out a giant S (are you saying I have a giant S?), a giant C, and so on - you get to decide what each letter should look like (and you get to practice 26 times - feel free to make your own funky font style that will make you (and in a legally-enforcable way, me) rich!!) To do this, I'd suggest functions void get_letter and void print_letter (each would take 1 letter parameters - should it be value or reference parameters, people?!) Then, your mainline would call each of these functions 5 times in order to get and then print the 5 letters. (By the way, if you're gonna do that many letters, you'd better make sure you don't forget the stamps!!!!)

#2 You are to write a function called "power" that accepts two integer parameters: the base and the exponent. This function should return the answer of base raised to the power of exponent. For example, power(2,3) should return 8, and power(4,-2) should return 0.0625 (which is 1/16!). In order to test your function, your main program should have two sections: The first should be a loop that executes 3 times, each time asking you for a power and a base and then printing out the result each time. The second part of your program should then ask the person for 2 sets of base/exponents, and then calculating the value of the 2 expressions:

1) $Base1^{Power1} + Base2^{Power2}$ and 2) $(Base1^{Power1})$ to the power of $(Base2^{Power2})$

EACH IN ONE LINE OF C++ CODE!!

So, a sample run might look something like this (As always, user input is in **bold**)
Enter a base **3**
Enter an exponent **4**
3 to the power of 4 is 81
Enter a base **-2**
Enter an exponent **-3**
-2 to the power of -3 is -0.125
Enter a base **0**
Enter an exponent **0**
Invalid Input – both the case and exponent can't be 0! Re-enter….
Enter a base **53244**
Enter an exponent **0**
5244 to the power of 0 is 1
Enter a base **2**
Enter an exponent **3**
Enter a second base **3**
Enter a second exponent **1**
(2 to the power of 3) + (3 to the power of 1) is 11
(2 to the power of 3) to the power of (3 to the power of 1) is 512
(Just to show you the POWER of functions!)

#3 Q: In order to help your class success rate on Cards of Destiny Fridays, you are to write a computer version of The "Higher/Lower/The Same" Game to see how many in a row a user can get (Plus, now that marijuana is legal, "How High Can You Go?" kinda fits!) Each round, the program will let the player continue playing along as they are correct. Once they lose, the program will tell them how many they had correct and ask them if they want to play another round. Once the player quits, the program will tell them the average number of correct answers they had per round and the longest winning streak they had. This program could use functions from the past like int get_card(), int get_suit(), void print_card(int card, int suit), and int card_value(int card) – but be careful – this is different than other card_value functions, since in this game, all face cards do NOT have the same value! Note that because we're not keeping track of which cards have been dealt, it's quite possible that the same card may appear more than once in the same round! (Note: If this happens in a card game with your friends, though, they're totally cheating!)

As always, be sure to use significant commenting, indenting, and variable dictionaries to make your code as readable as possible. Your cards will look much nicer, of course, but a run might go something like this: (User responses are in **bold**)

Welcome to The "Higher/Lower/The Same" Game (aka "Try and Get A New High"!)

ROUND 1:
The card is J - will the next card be (H)igher, (L)ower, or the (S)ame? **W**
That is not valid! Enter H, L, or S only **L**
The card is 5 - You are Correct! You have 1 in a row!
The card is 5 - will the next card be (H)igher, (L)ower, or the (S)ame? **H**
The Card is A - You are Correct! You have 2 in a row!
The Card is A - will the next card be (H)igher, (L)ower, or the (S)ame? **H**
The Card is 5 - You Are Wrong! You got 2 correct in a row this round – a new high!
Do you want to play another round? (Y/N) **Y**

ROUND 2:
The card is 10 - will the next card be (H)igher, (L)ower, or the (S)ame? **L**
The Card is Q - You Are Wrong! You got 0 correct in a row this round.
Do you want to play another round? (Y/N) **N**

You played 2 rounds, with an average of 1 correct answer per round, and your highest score was 2 in a row!

#4 I think one of the reasons the Manitoba government is currently running an advertising campaign warning folks about the dangers of gambling is because we did so many gambling programs in Comp Sci 20S! And good news – Comp Sci 30S has even more! For the first of many in this course, we're going to have you write a program involving "Round and Round She Goes…Where It Stops, No One Knows"! No, we're not talking about your happy childhood on the Merry-Go-Round, we're talking about one of the few gambling games we haven't simulated yet – (Kanga-)Roulette! Basically, a Roulette wheel features the numbers from 1 to 36 inclusive, and each round, a little silver ball lands on one of those numbers (If only the Internet had sites that allowed you to play games, you could go see what it's like, but I know there's nothing like that out there!) We're going to play a simplified version of the game with the following betting options:
1) Betting on a Single number (This pays out at 35 to 1)

2) Betting whether the number will be Even or Odd (This pays out at 1 to 1), or
3) Betting on Two Numbers (This pays out at 17 to 1).
Again, like all the quality gambling games we've written in Comp Sci, you'll generate a random amount of money for the user at the start and validate that their bet is legal each time; be sure to validate the choice they make in each game, too! To do this, I'd suggest a menu driven program with functions like Menu, Initialize, GetBet, PlaySingle, PlayEven, PlayOdd, PlayTwoNumbers, etc – you'll note I haven't included any parameters, since I want you to figure out whether each of them should be value or reference! To give you some idea, here's a quick sample run: (Of course, you'll probably do yours in colour to make it look way better!)
Welcome to the (Kanga-)Rou-lette Game! You have $753 to start.
Do you want to play (S)ingle, (E)ven, (O)dd, (D)ouble numbers, or e(X)it? **R**
INVALID CHOICE – Pick (S)ingle, (E)ven, (O)dd, (D)ouble numbers, or e(X)it? **S**
What number you think it will be (1-36)? **50**
INVALID CHOICE – What number you think it will be (1-36)? **20**
What amount do you want to bet ($1 - $753)? **1000**
INVALID AMOUNT - What amount do you want to bet ($1 - $753)? **100**
Let's start the wheel……..The number is……32!
Since you bet on 20, you lose $100 and now have $653!
Do you want to play (S)ingle, (E)ven, (O)dd, (D)ouble numbers, or e(X)it? **O**
What amount do you want to bet ($1 - $653)? **50**
Let's start the wheel……..The number is……5!
Since you bet on Odd and 5 is an Odd number, you win $50 and now have $703!
Do you want to play (S)ingle, (E)ven, (O)dd, (D)ouble numbers, or e(X)it? **D**
What two numbers do you choose (1-36)? **22**
44
INVALID CHOICE - What two numbers do you choose (1-36)? **22**
33
What amount do you want to bet ($1 - $703)? **100**
Let's start the wheel……..The number is……22!
Since you bet on 22 and 33, you win 1700 (17 * $100) and now have $2403!
Do you want to play (S)ingle, (E)ven, (O)dd, (D)ouble numbers, or e(X)it? **E**
What amount do you want to bet ($1 - $2403)? 2403
Let's start the wheel……..The number is……21!
Since you bet on Even and 21 is an Odd number, you lose $2403 and now have $0!
You're Broke! No more (Kanga-)Rou-lette for You!

#5 Time for more gambling! This one plays a Slot Machine (although given this is happening in my class, maybe we should call it a Scott Machine!) For the second program in a row, we will start by ass……..(why didn't you say something?!!!) …igning the user a random amount of money from $1 to $1000. The user will then be allowed to make a bet for each round – once the bet is made, the Slot Machine prints out the 3 symbols that come up (**for full marks, there must be a dramatic pause between each one showing up to make it more exciting**!), and then the user's money is update based on what happens! My suggestion for the basic version of the game is to use 3 possible symbols (you can make them using whatever theme you want!) and that if all 3 are the same, the user wins 25 times their bet (which seems good, but since there is a 1 in 27 chance, the casino will still make money that way!), and if any 2 are the same, the user keeps their bet (although as many of you often do, feel free to go well beyond the basic version to add in whatever you want – just make sure to explain the rules to the user if they are different than the basic version!) The game continues as

long as the user wants to keep playing, unless they run out of $$$. As well, at the very end, you should tell them whether they won, lost, or broke even compared to the amount they started with. Of course you will also add in colours to make it look awesome (as awlays!), so a possible sample run of the game might look like

Welcome to the Scott Machine!
You start with $555.
How much do you want to bet ($1-$555)? **1000**
Invalid - How much do you want to bet ($1-$555)? **100**
For your $100 bet, the Scott Machine shows…..
==FONT==(delay)==OF==(delay)==WHEEL==
There are no matches, so you lost $100 and now have $455
Do you want to play again (Y/N)? **Y**
How much do you want to bet ($1-$455)? **200**
For your $200 bet, the Scott Machine shows…..
==WHEEL==(delay)==OF==(delay)==WHEEL==
Two windows match, so you keep your $200 and still have $455
Do you want to play again (Y/N)? **Y**
How much do you want to bet ($1-$455)? **100**
For your $100 bet, the Scott Machine shows…..
== FONT ==(delay)== FONT ==(delay)== FONT ==
All three windows match, you win the $2700 (27 * $100) jackpot and now have $3155!!!
Do you want to play again (Y/N)? **N**
You are leaving with $3155, which means you won $2600 since you started – congrats!

#6 Huuuuuuuuurrrrrrrry! Hurrrrrrrrrrrrrry Harrrrrrrrrrrrrd! What's a Canadian computer assignment without a curling question – especially in the middle of the Olympics! You are to write a program to simulate a curling game between the teams of your choice! Curling consists of 10 (or more, if it's a tie!) ends - only 1 team scores per end. Technically, it's possible for a team to score up to 8 points, but for our simulation, we'll assume no one can score more than 4 (generally true in high level curling). We'll use the following (invented) percentages in our game:
Chance of blank end (both teams score 0) = 15%
Chance of team with hammer scoring 1 = 20%
Chance of team with hammer scoring 2 = 15%
Chance of team with hammer scoring 3 = 10%
Chance of team with hammer scoring 4 = 5%
Chance of team without hammer stealing 1 = 19%
Chance of team without hammer stealing 2 = 10%
Chance of team without hammer stealing 3 = 5%
Chance of team without hammer stealing 4 = 1%
You are to play a 10-end (or extra end(s), if necessary!) game of curling. To do this, I'd suggest the following functiones:
bool coin_toss(void); this returns a true if the first team has the hammer or false if not - it basically flips a coin to see who gets the hammer.
void play_end(bool& first_team, int &t1score, int &t2score); this function plays an end, and determines (based on the percentages above) what happens - your function should adjust the team scores and also adjust the reference variable first_team to show which team scored.
Your main program should play a game by first calling coin_toss to decide which team gets the hammer, and then play 10 ends by calling play_end 10 times. After each end,

be sure to update the “who’s got the hammer” variable accordingly! As well, the mainline should print out the score and tell who wins as follows:

END	TEAM 30S	TEAM 40S
1	0	2
2	1	0
....		
10	3	0
	------	------
FINAL SCORE	9	7

TEAM 30S WINS 9-7!!!!

Your program must also handle the possibility of extra ends (keep playing until someone scores!).

Final Warning: For legal reasons, once you do get this program working, you must promise not to use it to illegally make money in the undoubtedly lucrative world of betting on curling!

Computer Science 30S

C++ Assignment #2: *The "Arrays" The Roof Assignment!!!*

1) In order to show your incredible mathematical knowledge, you are asked to write a program that manages a set of data. The data will be all integers, and you will read the data into an array and then call several functiones (Yeah!) to do the calculations. This will be a menu driven program with several options, including

(E)nter Data - asks how many values will be entered and then asks users to enter that many values

(H)ighest - finds the highest value in the data

(L)owest - finds the lowest value in the data

(R)ange – finds the highest value minus the lowest value

(P)ositive - tells how many numbers in the data set are positive

(N)egative - tells how many numbers in the data are negative

(A)verage - calculates the average of all the data (you know what I MEAN, right?)

(V)ariance – It's VARY important that to see the hints document to find out what this is!

(Q)uit - ends the program

You may assume a maximum of 15 values, so your value array will look like:

int data[[15]; /* Array with up to 15 numbers */

Each of the menu choices (and the part to print the menu itself!) will have to be a separate function (you're welcome!), and in at least one case, it will be very helpful to call other functions (See if you can a"RANGE" to figure out which one I mean!)

So, some function prototypes might be:

void Enter_Data(int data[15], int& howmany); - note use of the & here!

char Print_Menu(void); - be sure to error check for any illegal menu choices!

int Highest(int data[15], int howmany);.....

As well, for the H, L, R, P, N, and A functions, you also need to consider the possibility that no data has yet been entered, and so you can't do the function. If it can't be done, send back a "dummy" value (like INT_MAX or INT_MIN) as the return value, and print an appropriate (yet tasteful!) message to the user!

2) It's time for you to write your own version of.... The Factor Game!!!! You know, that's the one where each team chooses a number and then the other team gets points for every number left on the board that's a factor of the number picked! Our version will have two teams (although, for you super-keeners, feel free to write a "player vs the computer" version!), and the program will begin by asking the maximum number for the game (we'll assume it's no more than 100!). The two players will then take turns picking ~~their noses~~ numbers until there are no numbers left - at the end of the game, the program should print out who won and the final score! Each time it is a team's turn to pick, they should be given a printout of the "numbers used" board - using either colours or blanks, it should show the player what numbers are available (of course, your program will need to validate the player's choice, just in case they can't read!). To do this, I'd suggest a boolean array "taken", which will tell whether each number has been taken yet. As well, much of the mainline should look something like:

```
initialize(p1_score, p2_score, taken, max_num);
while (more_numbers(taken, max_num) )
    {pick_num(1, p1_score, p2_score, taken, max_num);
     if (more_numbers(taken,max_num))
          pick_num(2, p2_score, p1_score, taken, max_num)
     }
```

3) You've played it at Birthday Parties, You've played it at Spirit Week, so now it's time to write the computer version of ….. Musical Chairs! Your version should begin by

asking the user for how many players there are, and after getting and validating that the number of players is less than or equal to 10 (so it fits on one line!) and greater than 1 (so it's not boring!), your program should play rounds of Musical Chairs until there is only one player left (that's right – musical chairs is the original Survivor game!) Each round, the program should print out who got chairs and who was eliminated, and at the end, the program needs to tell who won. Plus, just so the user feels a part of what's going on, we will once again turn this into a gambling game where the user starts with a random amount from $1-$1000 and then gets to predict who will win – if they are right, they win (n-1) times their bet, (where n is the number of players); if they are wrong, they lose their bet and are asked if they want to play again after each round. So, a sample run might look like:

Welcome to DMC (Digital Musical Chairs)!
How many players are playing (from 2-10)? **25**
INVALID CHOICE! How many players are playing (from 2-10)? **5**
You have $777 – how much do you want to bet ($1-$777)? **800**
INVALID BET - how much do you want to bet ($1-$777)? **100**
Which player do you think will win (1-5)? **7**
INVALID PLAYER – who do you think will win (1-5)? **3**
Round 1:
Players Seated 4 2 1 5 so player 3 was eliminated!
Press any key to continue….
Round 2:
Players Seated 2 4 1 so player 5 was eliminated!
Press any key to continue….
Round 3:
Players Seated 1 4 so player 2 was eliminated!
Press any key to continue….
Round 4:
Players Seated 4 so player 1 was eliminated!
Press any key to continue….
PLAYER 4 WAS THE WINNER, so you lose $100 and now have $677.
Do you want to play again (Y/N)? **Y**
You have $677 – how much do you want to bet ($1-$677)? **100**
Which player do you think will win (1-5)? **2**
Round 1:
Players Seated 3 1 2 5 so player 4 was eliminated!
Press any key to continue….
Round 2:
Players Seated 2 5 1 so player 3 was eliminated!
Press any key to continue….
Round 3:
Players Seated 2 1 so player 5 was eliminated!
Press any key to continue….
Round 4:
Players Seated 1 so player 1 was eliminated!
Press any key to continue….
PLAYER 2 WAS THE WINNER, so you win $400 ($100*(5-1)) and now have $1077.
Do you want to play again (Y/N)? **N**

Note that in order to maximize the dramatic tension, for full marks, there must be a delay in printing each player as they are seated!

Computer Science 30S C++ Assignment #3:

1) In class, we wrote a program that ***-ed words, but who says letters can't be obscene, too? Write a function int replace(char string[80], char badchar, char newchar), which goes through string, and replaces all occurrences of "badchar" with "new char". It then returns the **number** of replacements that were done. For example, replace(string, 'a', '$') would replace every 'a' with '$' and return the number of a characters that were replaced with $ characters. Then, write a mainline that asks the user for the number of strings, and then for each string read in, the program allows the user to type in as many replacements as they want. Only print the final version of the string after **all** replacements have been done as well as the **total** number of replacements. For example, a sample run would be:
How many strings are there? **2**
Enter string #1: **The quick brown fox jumps over the lazy dog.**
Enter character to replace: **o**
What should o be replaced with? **@**
Do you want another replacement? (Y/N) **Y**
Enter character to replace: (Note – a space was typed here!)
What should r be replaced with? **+**
Do you want another replacement? (Y/N) **N**
After a total of 12 replacements, the final string is:
The+quick+br@wn+f@x+jumps+@ver+the+lazy+d@g.

Enter string #2: **Computer Science Rocks!!!**
Enter character to replace: **!**
What should o be replaced with? **?**
Do you want another replacement? (Y/N) **N**
After a total of 3 replacements, the final string is: Computer Science Rocks???

2) What is the most common (and one of the least secure) passwords? That's right – it's "password"! For that reason, some websites randomly generate a password for you. You are to write a program that asks the user for the length of password they want (maximum of 24 characters – I know you've got free time, but you don't have that much free time!) The new password can be made up of whatever characters you want, but it must include at least all upper and lower case letters and numbers. After randomly generating the password for them, you should make them type it in (twice!) in order to make sure that they've actually got it – if not, keep forcing them to type it in (twice) until they get it correct!

So, here's a couple of sample runs:
What length of password do you want (max = 24)? **100**
Invalid length – choose between 1 and 24? **8**
Your new password is f3K2G1yP
Type your new password ********
Confirm your new password by re-typing it ********
Password reset!

What length of password do you want (max = 24)? **6**
Your password is b16W5v
Type your new password ******
Confirm your new password by re-typing it ******
Passwords don't match – type your new password ******
Confirm your new password by re-typing it ******

Password reset!

3) You may have seen a “motivational” meme like this:
Putting aside the questionable grammar (Coincident? Don’t they mean Coincidence?), you are to write a program to use the numeric position (starting at 1 for A, 2 for B, etc) for letters of the alphabet to find the “numeric value” of whatever words the user enters. The words may be entered with upper or lower case, so your program has to handle that as well! Start off by asking the user for the number of words they want to check, and your program should then print the value of each word. As well, if the user enters a 100% word, print a message that confirms that. Here’s a sample run:

Welcome to the Word Alphabetic Knowledge And Numeric Data Analyzer (WAKANDA)
How many words do you want to analyze? **5**
Enter word #1: knowledge
KNOWLEDGE has a WAKANDA value of 11 + 14 + 15 + 23 + 12 + 5 + 4 + 7 + 5 = 96%.
Enter word #2: HARDWORK
HARDWORK has a WAKANDA value of 8 + 1 + 18 + 4 + 23 + 15 + 18 + 11 = 98%.
Enter word #1: ATtiTuDe
ATTITUDE has a WAKANDA value of 1 + 20 + 20 + 9 + 20 + 21 + 4 + 5 = 100%.
CONGRATULATIONS – You have discovered a 100% word – ATTITUDE!!!!!
….
For full marks, you must give at least 4 DIFFERENT 100% words that you found in your comments at the start of the program

4) You remember how much fun the “Too High / Too Low” game was last year? What if I told you that you could play an even more complicated version of the game every time you went to your locker (Pandemic side note – have you folks ever even had a locker at Dakota?) We’re going to write the “Too High / Too Low - Locker Opening” version! You kids and your new fangled locks - when I went to school, all we had were **number** combinations - but noooo - you kiddies have to have **letter** combinations on your locks! Suppose your locker opens with a 3-letter, **random** alphabetic code (to keep it simpler, we’ll only work with CAPITAL letters!) Write a function void open_locker(void) that begins by generating a random code, and then plays the Too High / Too Low game with the user until they guess the combination. (In order to decide if each guess is too high or too low, a certain function we talked about that compares strings will be **very** helpful (Hint, Hint, Hint, Hint, Hint!)) Once the user guesses the combo, tell them how many tries it took, and give them a rating (you’ll have to play the game a few hundred times to decide what ratings are suitable). Finally, write a mainline that has the user play the game once, and then asks “Play Again? (Y/N)” to allow the user to play until they’re bored (as if that would ever happen!). At the very end, it should tell them the average number of tries it took and what their best score was. So a sample run might look like:
Try #1: Guess the 3-letter combo: **JAA -** Too LOW
Try #2: Guess another 3-letter combo: **TAA -** Too HIGH
Try #3: Guess another 3-letter combo: **QAA -** Too LOW
Try #4: Guess another 3-letter combo: **SAA -** Too HIGH
Try #5: Guess another 3-letter combo: **RAA -** Too LOW
Try #6: Guess another 3-letter combo: **RJA -** Too HIGH
Try #7: Guess another 3-letter combo: **RDA -** Too HIGH
Try #8: Guess another 3-letter combo: **RBA -** Too HIGH
Try #9: Guess another 3-letter combo: **RAJ -** Too LOW
Try #10: Guess another 3-letter combo: **RAT**
YOU GOT IT IN 10 TRIES - YOUR RATING IS... Possible Burglar!!!

Do you want to Play Again? (Y/N) **Y**
Try #1: Guess the 3-letter combo: **JAA -** Too HIGH
Try #2: Guess another 3-letter combo: **DAA -** Too LOW
Try #3: Guess another 3-letter combo: **GAA -** Too HIGH
Try #4: Guess another 3-letter combo: **EAA -** Too HIGH
Try #5: Guess another 3-letter combo: **DJA -** Too LOW
Try #6: Guess another 3-letter combo: **DRA -** Too HIGH
Try #7: Guess another 3-letter combo: **DLA**
YOU GOT IT IN 7 TRIES - YOUR RATING IS... Amazing!!!
Do you want to Play Again? (Y/N) **Y**

You played 2 time(s), and it took you an average of 8.5 tries to solve each one! Your best score was 7 tries!

5) M and M - not a candy bar, not the wrapper... it's the game Master Mind!! As discussed on the assignment, MasterMind has four spaces, each of which can be any one of Blue, Green, Black, Yellow, Purple, or Red (feel free to pick whatever colours you want!) Each round, the player guesses the 4 positions, and is told a) if they are correct, or b) the number of spots they have **exactly** correct, and the number of spots where they have the correct colour, but in the **wrong** place (Like LINGO, this is another game that Wordle totally stole ideas from!). The player gets a total of 10 guesses to get the correct answer; if they haven't done it after 10 guesses, then they lose (In that case, be sure to tell them the correct answer, too!) ! Write a function bool master_mind(void) that plays a game of master_mind and then returns a "true" if the player won and "false" if the player lost. Write a mainline that plays the game at least once, and asks the user if they want to Play Again. Once the user says they don't want to play again, your program should tell the user how many games they won, how many games they lost, their winning percentage and what their best score was. To give you some idea of how the game might work (and to ensure this is a 5-page assignment!), here's a sample run (although again, feel free to use some other colours!)
Possible Code Colours: (B)lue, (G)reen, (Y)ellow. (P)urple, (R)ed, or b(L)ack
Enter guess #1: **GPRY** ====> 1 Exact, 1 Wrong Place
Enter guess #2: **GLBP** ====> 2 Exact, 1 Wrong Place
Enter guess #3: **YRRL** ====> 1 Exact, 0 Wrong Place
Enter guess #4: **BBRG** ====> 0 Exact, 1 Wrong Place
Enter guess #5: **BRGY** ====> 1 Exact, 0 Wrong Place
Enter guess #6: **PLBY** ====> 1 Exact, 1 Wrong Place
Enter guess #7: **GLRB** ====> 2 Exact, 0 Wrong Place
Enter guess #8: **GLYP** ====> 2 Exact, 1 Wrong Place
Enter guess #9: **GLPB** ====> 3 Exact, 0 Wrong Place
Enter guess #10: **GLPG** ====> 3 Exact, 0 Wrong Place
YOU LOST - The Correct Answer was G**L**P**L**!!!! Play Again? (Y/N) **Y**
Enter guess #1: **BGLP** ====> 0 Exact, 2 Wrong Place
Enter guess #2: **YRBG** ====> 0 Exact, 2 Wrong Place
Enter guess #3: **LBYL** ====> 3 Exact, 0 Wrong Place
Enter guess #4: **LBYY** ====> 2 Exact, 0 Wrong Place
Enter guess #5: **LBBL** ====> 3 Exact, 0 Wrong Place
Enter guess #6: **LBRL** ====> YOU WIN!!!!! Play Again? (Y/N) **N**
You played 2 games - 1 Win and 1 Loss for a winning percentage of 50%, and your best score was getting it in 6 guesses!