**CS 3340.001 Project Report: MARS MIPS Jumbline 2 Game**

**Program Description**

This program is a game that helps you get in touch with your literal side. What it does is gives you some random letters and lets you figure out what words can those letters make. We implemented two sounds that can let you know whether you were right or wrong.

The first thing the program does is display the welcome message. Next, it will prompt the user to input his selection. After it gets the selection, the program will select N amount of letters randomly and display them to the user. Then it will inform the user what their letters are and how many words of length 2, 3, … to N can be made.

The user will be prompted to choose whether they want to shuffle(rearrange) the letters or take a guess at one of the words. If the user selects to shuffle, the program will randomly rearrange the letters to make the process easier for the user and display it to them. If the user selects the ‘guess’ option, they will be given a space to input their guess.

When the input is received, the program will then parse the dictionary file and check if the input matches with any word from the dictionary file. If the guess is correct, the program will increase the user’s score by the length of the word correctly guessed and play the sound corresponding to a correct guess. After a correct guess, the program will display the letters once more, the updated count of possible words, and the new score. The user will again be given the options of shuffling or guessing. If the guess is wrong, the user is told of their incorrect guess and a sound corresponding to a wrong guess will be played. The program then reminds the user of the letters, count of possible words, and the score.

**Challenges and Solutions**

This project was a great challenge for my teammates and I, and we were able to overcome them with some adversity. I believe it was a good experience in our young programming careers. Our first and hardest challenge was assembling the dictionary file and implementing it to our program. Once we got done with that, the rest worked out gradually, until we needed to be able to store the words we had found in the dictionary. To solve this challenge, we implemented a singly linked list structure with dynamic memory on the heap.

We went through a lot of challenges while working on this project, but I would have to say our toughest obstacle was actually working with the group. When everyone has a different schedule, work and school, it is very tough to work together. One of our team members, Nicholas, decided we should have a chat room online to discuss our plan, then he opened a GitHub repository where we could share our code together and provide feedback for each other. That really helped in solving our problem of assembling as a group. Near the end of our project, we were able to meet after class to polish up everything. I personally learned a lot by overcoming these challenges with my teammates and hope that this will benefit me in the future.

**Learning Process during the Project**

We actually learned the process of the MIPS programming language and the challenges that it poses in a real world application. Programming in a low level language was really challenging at first but once you get into the groove of things it seems to just flow just like all of the high level languages. All of the material that was taught in class came in handy to help understand the concepts of MIPS. I also learned how to work with a team of other students.

**Discussion of Algorithms and Data Structures**

When parsing the dictionary, the randomly chosen letters are known. The parsing procedure extracts every word from the entire dictionary, and checks if it can be made from the randomly chosen letters. This involves nested loops, and some extra data space to keep track of which letters have been used by the word. If the word is acceptable, a callback procedure supplied as a parameter is invoked and given the word. The callback procedure determines the length of the word and stores it in one of several linked list structures.

The linked lists are minimalistic, and the procedures to act on them are minimalistic too. The list procedures take as their first argument the address of the pointer to the linked list, so that they may modify the pointer. The find and remove procedures use linear traversal from the root node. While linked lists are very slow, they are easy to implement in assembly, and each linked list would not hold very many nodes anyway.

The words and the linked list nodes are allocated on the heap with syscall 9. Unfortunately, neither MIPS nor MARS provides a counterpart to syscall 9, so there is no way to free memory. The program could be trivially changed to free the memory of such an option existed, however. Thankfully the heap usage never increases after the initial formation of the linked lists, so running out of memory is not a concern.

**Contributions**

*Nicholas*

* Opened online communication methods and github repository.
* Assembled all the code that was submitted, optimize, and polish everything.
* Integrated individual code snippets from other group members

*Chris*

* Dictionary file implementation, dictionary file optimization, etc

*Ahmad*

* Input and Output / user interaction

*Kelby*

* Worked with Jerome and Chris on output and dictionary
* Implemented sounds

*Jerome*

* Worked on assembly of program with Nicholas to fix linked list bugs