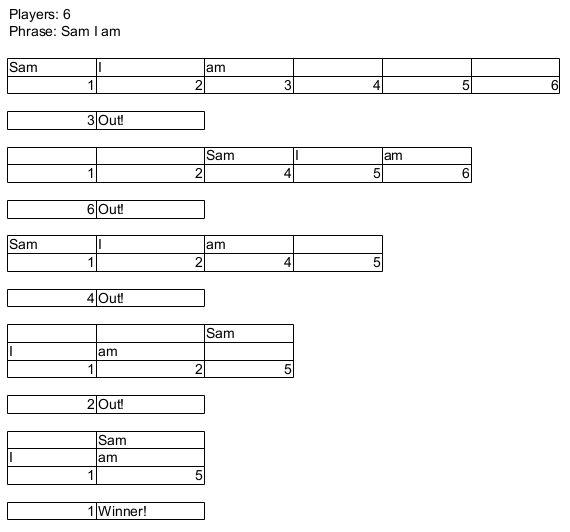
Nicholas Grogg

Dr. Edwards

CS151

**Project 3: Parts 1 and 2**

**Part 1, understand the problem**



To win Alice needs to be player 1.

**Part 2, plan**

Introduction

At it's roots this project is essentially going to be a heavily modified version of the linked list lab with the last node pointing to the head of the list. I'll be running the entire program within a while loop so users can easily run the program as many times as they want. I'll be allowing the user to name each node. Using variables that will be determined in the beginning of the program I'll create a linked list with multiple nodes to allow the user to enter multiple words. I'll be implementing a floor and ceiling for phrase length in the interest of time and simplicity.. I'll also be limiting players to at least 2 and at most 10. I will either assign a junk value or a NULL value to empty nodes in the linked list.

**Algorithm for program**

1. Program starts
2. While loop while bool = true
3. user enters length of phrase and number of people playing
4. while loop for >2 players or <10 players
5. while loop for phrase >1 or <10.
6. instantiate linked list
7. User enters phrase, enters word by word into list
8. for loop to output list
9. delete whichever node has the last word
10. repeat steps 8 and 9 until one node remains
11. Output which player wins
12. Ask if the user would like to play again
13. If yes, repeat steps 3 through 12
14. If no, set bool to false
15. Return 0

I may consider using an array to avoid problems with duplicate words, although I'll try to avoid arrays if possible.

I'll have three files, a main.cpp file, a class.cpp file and a class.h file.

**Variables**

**Main variables**

int PlayerCount, for how many people are playing

int WordCount, how many words in the array

bool KeepPlaying, true by default

(back up plan)

string array, if my default plan doesn't work

**Class variables**

Struct variables

string NodeName, name for node

int node value, for keeping track of the nodes

string NodeWord, I'll have a bunch of these to hold the phrase

ListNode \*head

**Classes**

Phrase\_list class

A class that's based on the linked list program

Struct ListNode

**Functions**

all functions will be in the class or struct

Default constructor to create an empty list

Copy constructor, because that's what's needed with a destructor

Destructor, to destroy the constructor and return memory to the heap

void DeleteNode, delete a node

void AddNode, add a node

void AddWord, add a word to a node

void OutputNodes, output nodes

(backup)

void Array2Node, add array to node, hopefully won't need