PROJECT 2

simon

CIS-5 40651
Leo Gutierrez
Date: 02/12/2014

Introduction

The title of this game is "simon". It is based on the game from the late '70's simon where a player follows a random pattern set by the game that is displayed in the form of light and sound. The game progresses until the player does not enter the same pattern sequence lit up by the game.

Our version is slightly different in that it doesn't use any color or any sounds. It will use four squares that will display one letter representing a color in one of the four squares. The object of the game is to enter the sequence pattern with the keyboard after the random letters are shown on the screen. If the sequence pattern is the same as what was displayed, then the game will continue. If the sequence pattern entered by the user does not match the sequence pattern displayed, the game ends. In this final version, the user can enter a skill level of 1, for beginner, 2 for intermediate and 3 for advanced. Depending on which level they choose, the length of the sequence pattern would be 10, 15 or 25 letters respectively. There will also be a time limit on how long the user has to make their entry. The higher the skill level, the less time you have to make the entries. If the players fails to enter the matching sequence pattern in the allotted time, then the game ends for them as well. You win the game by entering the sequence pattern that matches exactly to what is randomly stored at game time.

Summary

The project is about 740 lines long for this final attempt and is using 30+ variables, with 5 arrays, including 2 vectors and 19 functions with an overloaded function, using several of the functions to return variables, reference variables and the use of 7 include libraries.

Added functionality to this final version of the game include 2 files to write out and read in user names and scores, create vector arrays using the files and then sorting the vectors to display the top 10 scores at the end of the game. It will also add a skill level, 1 through 3, that will be input by the user before the game begins which will determine the size of the array to be used as the sequence pattern. I also added at the last minute 1 additional screen that displays the top 5 players top 5 scores. This was done in order to fulfill the requirement of using a 2 dimensional array and performing a search on an array.

Description

The game will consist of an initial output screen that will display the layout of the "game board" with an initial instruction to use the CAPS LOCK key to play the game. The letters flashed on the screen are the letters that are stored in the array and in order for the user input to match the array, they must be matched by character. An image of the initial screen can be viewed below.

The initial screen will look something like the image below.

```
        S I M O N

             |

------------   +   ------------

             |


***** TURN ON THE CAPS LOCK KEY *****
*****       TO PLAY THIS GAME       *****


READY TO PLAY…    HIT ENTER…
```

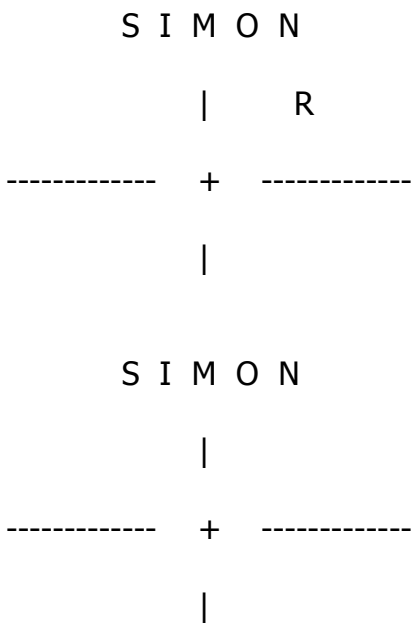
Once the game begins, the screen will refresh and display only the four squares shown above. After about a second or two, the screen will refresh again and this time, one of the squares will contain a letter. Note that the same letter will always be displayed in the same square should it appear. So the letter "G" will always appear in the top left square, the letter "R" will appear in the top right square, the letter "Y" will appear on the bottom left square, and the letter "B" will appear in the bottom right square. The letters were selected to mimic the original game of Green, Red, Yellow and Blue. So on the first and subsequent iterations, the screen will alternate as below…

```
        S I M O N                    blank screen

             |

------------   +   ------------

             |
```



The game will alternate like this depending on how far the user gets with entering the correct sequence pattern. After the first iteration, the screen will clear and the user will see the message below to enter the pattern.

Enter the pattern ===>

The user then would enter the R and hit enter. At this point, after the code verifies the user's input and compares it to the first column in the sequence pattern array, it will continue and display the sequence pattern again with 1 more letter added to the sequence pattern on up until reaches the max of for the skill level entered at the beginning of the game. If the sequence pattern is incorrect and does not match what is in the array, then the game ends and a score is displayed. On the second iteration, the user might enter RB for the sequence pattern, and so on… Let us assume that they entered the incorrect sequence pattern on the second iteration. The following message would then be displayed.

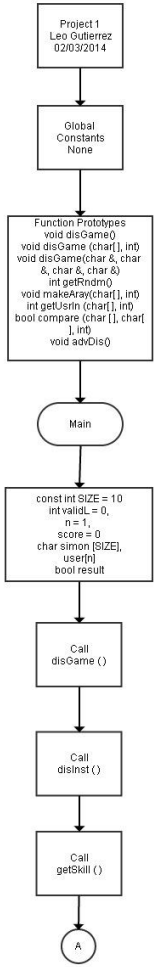
You lost the game. Thank you for playing!
Your score is: 1

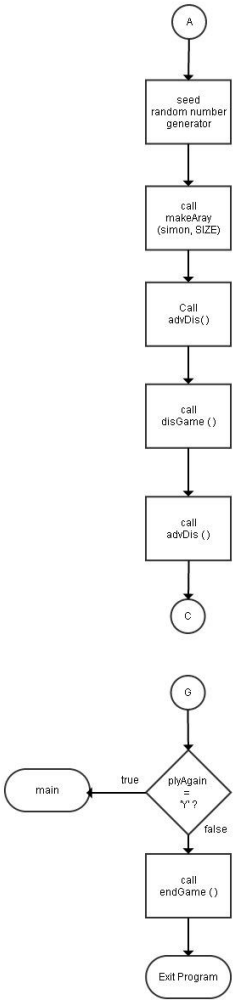
If the user wins the game, then the following will be displayed.

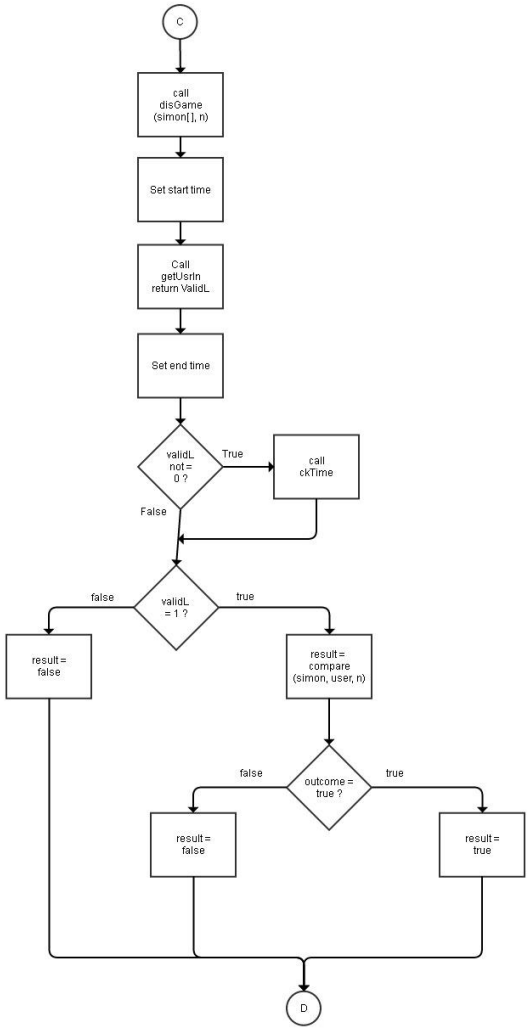
You Win!!!!
Your score is: 10

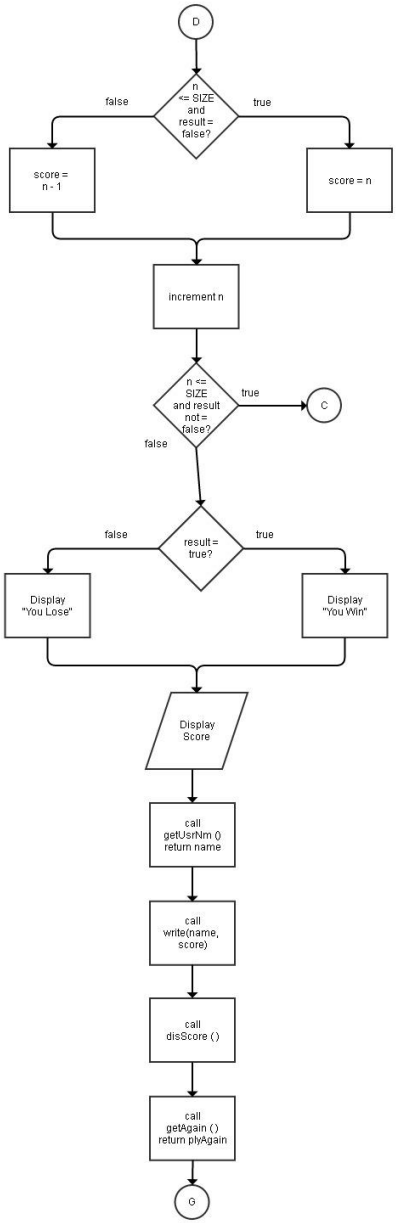
The following five pages are the flow chart of this game.

Project 1
page 1
CIS5

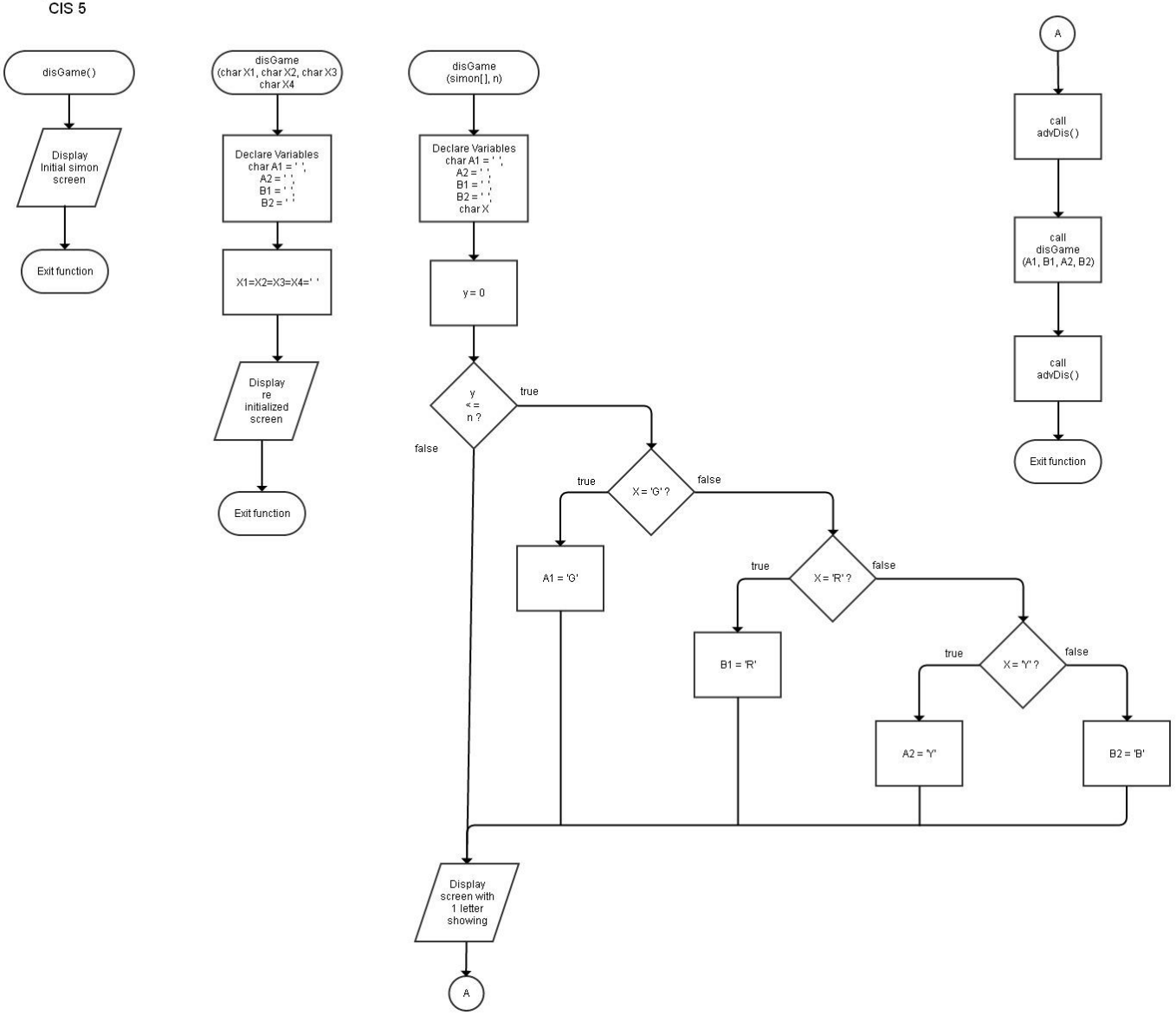




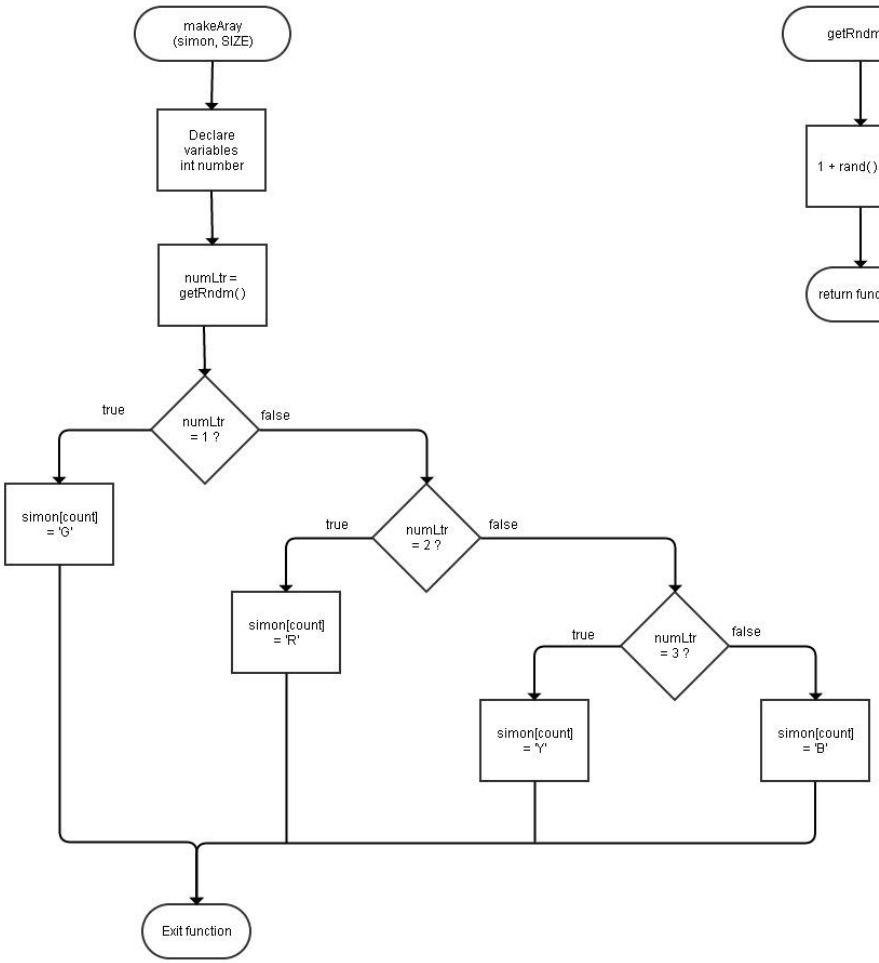


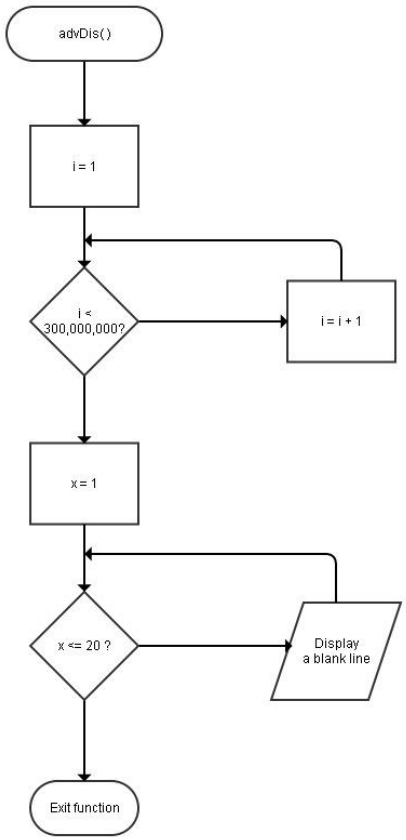


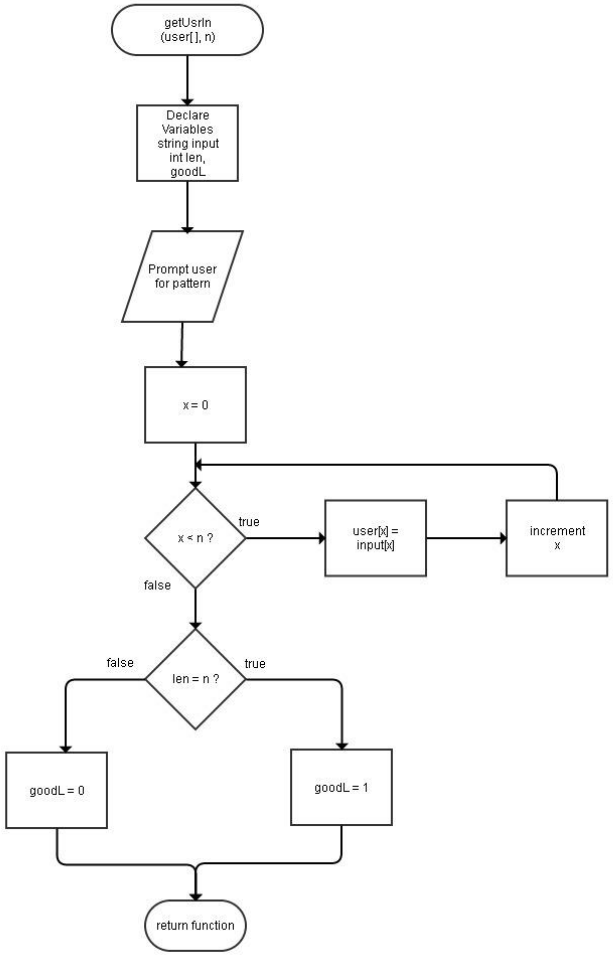
Project 1
page 2
CIS 5

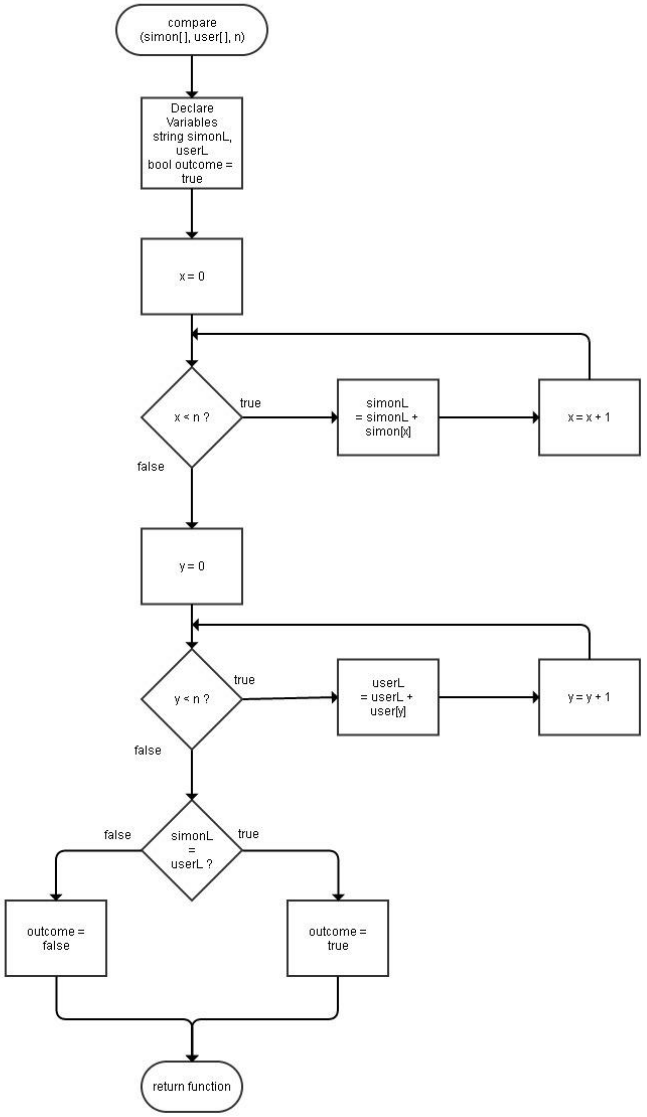


Project 1
page 3
CIS 5

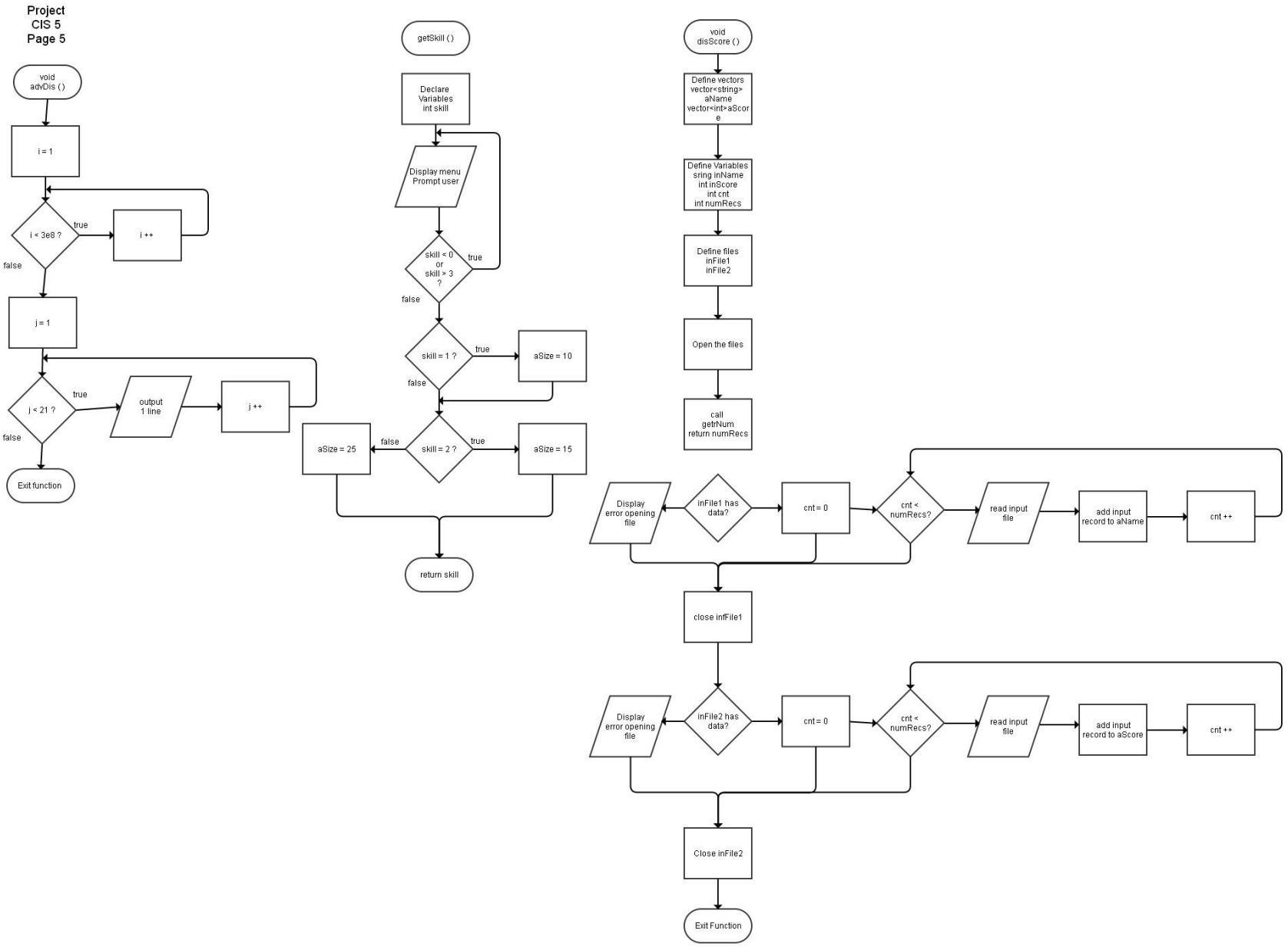





getUsrIn
(user[], n)
Declare
Variables
string input
int len,
goodL
Prompt user
for pattern
x = 0
x < n ?
true
user[x] =
input[x]
increment
x
false
len = n ?
false
true
goodL = 0
goodL = 1
return function


compare
(simon[], user[], n)
Declare
Variables
string simonL,
userL
bool outcome =
true
x = 0
x < n ?
true
simonL
= simonL +
simon[x]
x = x + 1
false
y = 0
y < n ?
true
userL
= userL +
user[y]
y = y + 1
false
simonL
=
userL ?
false
true
outcome =
false
outcome =
true
return function

Project
CIS 5
Page 5



Function Name	Variable Name	Variable Type	Description
Main()	SIZE = 1	int	Initialize array to 1 but will eventually contain the SIZE of the array once the user select the skill level.
	validL = 0	int	Holds the value returned from function getUsrIn () and used to determine if the user entered the correct length of the pattern.
	n = 1	int	n is the current iteration the game is on
	simon[ ]	char	simon [ ] holds the pattern array that is filled with 10 random values.
	user[ ]	char	user[ ] hold the user input and is used to compare to the simon array.
	result	bool	After the iteration is done displaying and comparing the user a value of true or false is stored here
	score = 0	int	Holds the final score
	timeElps = 0	int	Hold the time elapsed used to determine time user took to input pattern sequence.
	plyAgain	char	Y or N to play the game again.
	name	string	Holds the users input name
	skillL	int	Holds the skill level input by the user

	timeCk	bool	Returns true or false if the user entered within their time limit.
void disGame()	A1 = ' '	char	Sets initial display value
	B1 = ' '	char	Sets initial display value
	A2 = ' '	char	Sets initial display value
	B2 = ' '	char	Sets initial display value
void disGame(char, char, char, char)	A1 = ' '	char	Resets initial display value
	B1 = ' '	char	Resets initial display value
	A2 = ' '	char	Resets initial display value
	B2 = ' '	char	Resets initial display value
	X1 = ' '	char	Ref variable to send back to clear screen
	X2 = ' '	char	Ref variable to send back to clear screen
	X3 = ' '	char	Ref variable to send back to clear screen
	X4 = ' '	char	Ref variable to send back to clear screen
void disGame(char, int)	A1 = ' '	char	Clears the value for a letter
	B1 = ' '	char	Clears the value for a letter
	A2 = ' '	char	Clears the value for a letter
	B2 = ' '	char	Clears the value for a letter
void makeAray (char simon[], int SIZE)	numLtr	int	Receives a random number from getRndm()
int getUsrIn(char user[], int n)	input	string	Holds the input from the user
	len	int	Hold the string length of the user input

	goodL	int	Returns a 1 of the length is good and 0 if the length if bad.
bool compare(char simon[], char user[], int n)	outcome	bool	Returns true if the two arrays match, else returns false
	simonL	string	Put the entire simon array into the string for comparison
	userL	string	Put the entire user array into this string for comparison
void advDis()	No local variables used		
int getRndm()	No local variables used		
int getSkill(int &aSize)	skill	int	Used to return the skill level selected by the player.
void disScore()	aName	vector<string>	Used to hold a list of names
	aScore	vector<int>	Used to hold a list of scores
	inName	string	Hold the name to display
	inScore	int	Holds the score to display
	cnt	int	Holds a counter value
	numRecs = 0	int	Holds the number of scores in the score file.
int getrNum()	numRecs = 0	int	Hold the number of records in the file that will be returned to another function
	inScore	Int	The name of the records in the file.
void sortAray(vector<int> aScore,	DISPLAY = 10	const int	Limit of the number of scores to display

vector<string> aName, int numRecs)			
	SIZE = 5	Const int	Limit the number of players scores to display
	swap	bool	Flag for the bubble sort
	temp1	int	Temporary holder for the bubble sort
	temp2	string	Temporary holder for the bubble sort
	numScr	int	Holds the number of scores to display from the array in case there are less than 10.
	users[]	string	Holds a list of users
	noMatch	bool	Flag for the search routine.
	x	int	Counter for the search routine
	u	int	Counter for the search routine.
bool ckTime (int SkillL, int timeElps, int &validL)	tLimit	bool	Flag to return if the time limit for the player was exceeded.
void write(string name, int score)	No local variables used		
void disInst()	No local variables used		
string getUsrNm()	usrName	string	Used to hold the user name input by the user to be stored to a file.
char getAgain()	again = 'N'	char	Used to hold the return value of again.
int endgame()	No local variables used		
void srch (vector<string> aName, vector<int>	SIZE	const int = 5	Hold the size of the number of scores to display for each

aScore, string users[], int numRecs)			player.
	usr = 5	int	Declares the row of the scoreLst array to 5.
	scr = 5	int	Declares the columns of the scoreLst array to 5.
	scoreLst[usr][scr]	int	Array to hold 5 scores for 5 players.
	x = 0	int	Counter variable
	y = 0	int	Counter variable
	found = false	bool	Flag to indicate a record was found. Initialized to false.
	srchName	string	Hold area to search for a name.

References

1. Gaddis, Tony (2012). Starting Out With C++ From Control Structures through Objects, 7th Edition

The SIMON program

```
/*
 * File:   main.cpp
 * Author: Leo Gutierrez
 * Project 2 - SIMON game
 *
 * Object - to remember the pattern flashed on the screen and input the same
 *      pattern on the keyboard for as long as possible.
 *
 *
 * Upgrades to project 1
 *      1. Ask for a skill level of 1 beginner, 2 intermediate, 3 advanced.
 *      2. At the end of the game, win or lose, get user name
 *      3. Save the user name and score to a file
 *      4. Display the top 10 highest scores
 *      5. Added a timer to allow a certain amount of time per skill level
 *
 * Created on January 30, 2014, 5:34 PM
 */

//System Libraries
#include <iostream>
#include <iomanip>
#include <ctime>
#include <cstdlib>
#include <string>
#include <fstream>
#include <vector>
using namespace std;

//Global Constants

//Function Prototypes
void disGame ();                            //Display initial game screen
void disGame (char [], int);                  //Display game screen
void disGame (char &, char &, char &, char &); //Display reinit screen
void disInst ();                           //Display some initial instr
int getRndm ();                             //Get random # 1-4
void makeAray (char [], int);                 //Create Simon array
int getUsrIn (char [], int);                //Get user input and fill array
bool compare (char [], char [], int);          //Compare usr vs Simon arrays
void advDis ();                            //Scroll display and wait a sec
int getSkill (int &);                      //Get skill level
```

```
void disScore ();                          //display the high scores
int getrNum ();                            //return the number of recs
void sortAray (vector<int>, vector<string>, int);
bool ckTime(int, int, int &);               //verify the user input time
void write(string, int);                   //Write name and score to flies
string getUsrNm ();                         //Get user name
char getAgain ();                          //Ask user to play again or end
int endGame();                             //Stop the game
void srch(vector<string>, vector<int>, string [], int);  //search array

//Begin Program
int main() {

   //Declare Variables
   int validL = 0, n = 1, SIZE = 1, score = 0, timeElps = 0;
   char simon[SIZE], user[n];
   bool result, timeCk;
   int skillL;
   string name;
   char plyAgain;

   //Display initial screen
   disGame();

   //Display initial instructions
   disInst();

   //Get skill level
   skillL = getSkill(SIZE);

   //Set the random number generator
   srand(static_cast<unsigned int>(time(0)));

   //Create an array with SIZE elements and populate the whole thing now
   makeAray(simon, SIZE);

   //The actual game is looped here
   //Loop here while the pattern entered by the user still matches the Simon
   //array, user has entered the complete pattern of SIZE columns and within
   //the time limits of the level chosen.
   do{
      //Begin playing the game until the pattern by the user is broken
      advDis ();                    //Advance screen to clear it and wait
      disGame ();                    //Display with all spaces
```

```
        advDis ();                  //Advance screen to clear it and wait
        disGame (simon, n);         //Display with 1 character of pattern

        //Set the clock to start counting seconds here
        int strTime = time(0);

        //Get the user input pattern up to the nth iteration
        validL = getUsrIn(user, n);

        //Set the clock to stop the time here after the user has input
        //their pattern and hit the enter key.
        int endTime = time(0);

        //Calculate time elapsed
        timeElps = 0;                       //reset time
        timeElps = endTime - strTime;

        //Verify the user time entry only if the validL came back true.  If
        //it came back false "0", then there's no need to check the time
        //entry they lost the game for invalid length.
        if (validL != 0)
           timeCk = ckTime(skillL, timeElps, validL);

        //Validate user input
        if (validL == 1)
           result = compare(simon, user, n);
        else
           result = false;

        //Validate of win or loose to display the proper score
        if ((n <= SIZE) && (result == false))
           score = n - 1;
        else
           score = n;

        //Increment iteration
        n++;

}while ((result != false) && (n <= SIZE));

    //If result is false - user looses the game otherwise they're a winner.
    if (!result)
       cout << "You lost the game.\n";
    else
```

```
        cout << "You Win!!!!!\n";

    //Display the score
    cout << "Your score is : " << score << endl;

    //Go get the user's name to store in a file
    name = getUsrNm();

    //Save the user name and score to files appending to the end.
    write(name, score);

    //Input names and score into an array and display
    disScore();

    //Prompt user to play again...
    plyAgain = getAgain();

    //Continue if "Y"es, otherwise end the game.
    if (plyAgain == 'Y')
        main();
    else
        endGame();

    return 0;
}

//This function is called by main.  It displays the game board on the screen.
void disGame (){

    //Declare and initialize the variables
    char A1 = ' ', B1 = ' ', A2 = ' ', B2 = ' ';

    //Display the game
    cout << "           S I M O N          \n\n";
    cout << "\t" << A1 << "\t" << "|\t" << B1 << "\n\n";
    cout << "   ----------" << "\t+   ----------\n\n";
    cout << "\t" << A2 << "\t" << "|\t" << B2 << "\n\n\n";

}

//This function is called by disGame to reset the values of the 4 squares
// to "X"s.
void disGame (char &X1, char &X2, char &X3, char &X4){
```

```
    //Declare and initialize the variables
    char A1 = ' ', B1 = ' ', A2 = ' ', B2 = ' ';

    //Re-initialize all 4 squares to an asterisk.
    X1 = X2 = X3 = X4 = ' ';

    //Display the game
    cout << "           S I M O N          \n\n";
    cout << "\t" << A1 << "\t" << "|\t" << B1 << "\n\n";
    cout << "   ----------" << "\t+   ----------\n\n";
    cout << "\t" << A2 << "\t" << "|\t" << B2 << "\n\n\n";
}

//This function displays one of the letters and then calls a reset screen
//to "X" everything out again before displaying the next letter.
void disGame (char simon[],int n){

    //Declare and initialize the variables
    char A1 = ' ', B1 = ' ', A2 = ' ', B2 = ' ';
    char X;

    //Read the array 1 by 1 up until n to display each letter within the
    //iteration.
    for (int x = 0; x < n; x++){
        X = simon[x];

        if (X == 'G')
            A1 = 'G';
        else if (X == 'R')
            B1 = 'R';
        else if (X == 'Y')
            A2 = 'Y';
        else
            B2 = 'B';

        //Display the game
        cout << "           S I M O N          \n\n";
        cout << "\t" << A1 << "\t" << "|\t" << B1 << "\n\n";
        cout << "   ----------" << "\t+   ----------\n\n";
        cout << "\t" << A2 << "\t" << "|\t" << B2 << "\n\n\n";

        //Delay putting out the reset of the screen for a couple of seconds
        //before showing the next letter then scroll down.
        advDis ();
```

```
        disGame (A1, B1, A2, B2);

        //Delay refreshing the screen for a couple of seconds and the scroll
        //down.
        advDis ();
    }
}

//This function is called by main.  It fills an array with random letters
//based on an integer value coming back.
void makeAray (char simon[],int SIZE){
    //Declare variables
    int numLtr;

    //Get a random integer and send it back to populate the array.
    //Calls the random function.  Based on the number 1 - 4 will assign a
    //letter to represent color (G)reen (R)ed (Y)ellow and (B)lue
    for (int count = 0; count < SIZE; count++){
        numLtr = getRndm();
        switch (numLtr){
            case 1 : simon[count] = 'G';            //G for Green
                     break;
            case 2 : simon[count] = 'R';            //R for Red
                     break;
            case 3 : simon[count] = 'Y';            //Y for Yellow
                     break;
            default: simon[count] = 'B';            //B for Blue
                     break;
        }
    }
}

//Get a random number 1 - 4 to use to populate the pattern in Simon array.
int getRndm(){

    return 1 + rand()%4;                    //Limit the number from 1 -4
}

//This function is called by main.  Prompt the user for the current pattern
//and validate the entry.
int getUsrIn(char user[], int n){

    //Declare variable
    string input;
```

```
    int len, goodL;

    //Prompt user to enter the pattern
    cout << "Enter the pattern ===> ";
    cin >> input;

    //Populate the user array with the input variable character by character.
    for (int x = 0; x < n; x++){
        user[x] = input[x];
    }

    //Test the length of the input.  Must be equal to the current iteration.
    len = input.length();
    (len == n) ? goodL = 1 : goodL = 0;

    //Return 1 if the length is good, otherwise it returns 0.
    return goodL;
}

//This function is called by main.  It will compare the Simon array vs the
//user array to the nth column and return true or false.
bool compare (char simon [], char user [], int n){

    //Declare variables
    bool outcome = true;             //Return value
    string simonL, userL;            //Variables to hold comparison values.

    //Populate a variable from the Simon array up to the nth iteration.
    for (int x = 0; x < n; x++){
        simonL += simon[x];
    }

    //Populate a variable from the user array up to the nth iteration.
    for (int y = 0; y < n; y++){
        userL += user[y];
    }

    //Compare the variables and return T or F.
    (simonL == userL) ? outcome = true : outcome = false;

    //Exit function
    return outcome;
}
```

```
//This function is called by main and disGame to advance the lines down 20
//lines and wait a couple of seconds to hold the screen to view
void advDis (){

   for (int i = 1; i < 3E8; ++i);    //Wait a couple of seconds
   for (int j = 1; j < 21; j++){     //Push down 20 lines to clear screen
      cout << "\n";
   }
}

//This function is called by main function.  It is used to get the skill
//level the user wishes to play.
int getSkill (int &aSize){

   //Declare variables
   int skill;

   //Display menu and get skill level.  Populate the size of the array
   //based on skill level chosen.
   cout << "Skill level 1 - Beginner \n";
   cout << "Skill level 2 - Intermediate \n";
   cout << "Skill level 3 - Advanced \n";
   cout << "Select a skill level ===> ";
   cin >> skill;

   while (skill < 0 || skill > 3){
      cout << "Invalid skill level...\n\n";
      cout << "Skill level 1 - Beginner \n";
      cout << "Skill level 2 - Intermediate \n";
      cout << "Skill level 3 - Advanced \n";
      cout << "Select a skill level ===> ";
      cin >> skill;
   }

   //Assign the size of the "simon" array to one of the following.
   switch (skill){
      case 1 : aSize = 10;
      break;
      case 2 : aSize = 15;
      break;
      default : aSize = 25;
      break;
   }
```

```
    //Returns the skill level selected by the user
    return skill;
}

//This function is called by main.  It will read the just updated high
//scores file and list the top 10 highest scores
void disScore (){

    //Define vectors to store input from file
    vector<string> aName;
    vector<int> aScore;

    //Define variables
    string inName;
    int inScore;
    int cnt;
    int numRecs = 0;

    //Define the files
    ifstream inFile1;               //Input user names
    ifstream inFile2;               //Input user scores

    //Open the input files
    inFile1.open("userNames.txt");
    inFile2.open("userScores.txt");

    //Get number of records from the name.txt file
    numRecs = getrNum();

    //Read the names from the file and store in a
    //vector to be sorted later.
    if (inFile1)
    {
        for (cnt = 0; cnt < numRecs; cnt++){
           while (inFile1){
               getline(inFile1, inName);
               aName.push_back(inName);
             }
         }
    }
    else
    {
       //Display error opening the file
       cout << "Error opening the file 1.\n";
```

```
    }

    //Close the user name file.
    inFile1.close();

    //Read the scores from the file and store in a
    //vector to be sorted later.
    if (inFile2)
    {
       for (cnt = 0; cnt < numRecs; cnt++){
          while (inFile2 >> inScore){
             aScore.push_back(inScore);
           }
        }
    }
    else
    {
       //Display error opening the file
       cout << "Error opening the file 2.\n";
    }

    //Close the user score file.
    inFile2.close();

    //Sort the name and scores array dictated by the highest to lowest score.
    //Goes to the sort function if there is more than 1 score saved,
    //otherwise list the 1 score in the file here.
    if (numRecs > 1)
       sortAray(aScore, aName, numRecs);
    else
    {
       //Display the top 10 high scores
       cout << "\t\t TOP 10 SCORERS " << endl;
       cout << "\t Player " << "\t\tScore" << endl;
       for (int z = 0; ((z < numRecs)); z++ ){
         cout << "\t" << setw(10) << aName[z] << "\t\t" << aScore[z] << endl;
       }
    }
}

//This function is called by disScore.  It is used to read the number of
//scores in the userScores.txt file to get the range to fill the vectors.
int getrNum(){
```

```
    //Declare variables
    int numRecs = 0;
    int inScore;

    //Define input file
    ifstream inFile;                    //Input scores

    //Open the user scores file to count the number of scores in it
    inFile.open("userScores.txt");

    //Get the number of actual records first.
    if (inFile)
    {
         //Read the names in the file and count them.
        while (inFile >> inScore)
        {
               numRecs++;
        }
    }
    else
    {
        //Display error opening the file
        cout << "Error opening the file 1.\n";
    }

    //Close the input file
    inFile.close();

    return numRecs;
}

//This function is called by disScore function.  It does a parallel sort of
//two vectors in descending order and display the top 10 scores.  It then
//stores the top 5 players by high scores into an array and passes that
//to a function that will display those 5 players top 5 scores.
void sortAray (vector<int> aScore, vector<string> aName, int numRecs){

    //Declare variables
    const int DISPLAY = 10;
    const int SIZE = 5;
    bool swap;
    int temp1, numScr;
    string temp2;
    string users[SIZE];                     //Array of 5 users
```

```
bool noMatch;                          //No Match
int x = 0, u = 0;                      //initialize counters

//Bubble sort
do
{
   swap = false;
   for (int count = 0; count < (numRecs - 1); count++)
   {
      if (aScore[count] < aScore[count + 1])
      {
         temp1 = aScore[count + 1];
         aScore[count + 1] = aScore[count];
         aScore[count] = temp1;

         temp2 = aName[count + 1];
         aName[count + 1] = aName[count];
         aName[count] = temp2;

         swap = true;
      }
   }
}while (swap);

//Scroll down 20 lines to get a clean display
for (int j = 1; j < 21; j++){     //Push down 20 lines to clear screen
   cout << "\n";
}

//Display the top 10 high scores
cout << "\t\t TOP 10 SCORERS " << endl;
cout << "\t Player " << "\t\tScore" << endl;

//Loop up to 10 times.  If less than 10 than loop until the number of
//records has been reached.  Once it has reached 10, then continue to
//display only the top 10.
if (numRecs < DISPLAY)
   numScr = numRecs;
else
   numScr = DISPLAY;

for (int z = 0; z < numScr; z++ ){
   cout << "\t" << setw(10) << aName[z] << "\t\t" << aScore[z] << endl;
}
```

```
    //This stores the top 5 scoring names in an array called users[].
    //Read the next name in the vector.
    while (u < SIZE){

        if ((aName[x] != users[0]) && (aName[x] != users[1]) &&
            (aName[x] != users[2]) && (aName[x] != users[3]) &&
            (aName[x] != users[4])){
                    noMatch = true;
        }

        if (noMatch){
            users[u] = aName[x];
            u++;
            noMatch = false;
        }

        x++;
    }

    //Search for the scores and put into a 2 dimensional array
    srch (aName, aScore, users, numRecs);

}

//This function is called by main.  It is used to verify the time it took a
//user to input the pattern.  Time allowed is dictated by skill level chosen
//at the beginning of the game.
bool ckTime(int skillL, int timeElps, int &validL){

    //Declare variable
    bool tLimit;                            //Time within limit T or F

    //Verify time it took user to input pattern.
    //Skill level 1 = 18 second limit
    //Skill level 2 = 14 second limit
    //Skill level 3 = 10 second limit
    if (skillL == 1){
        if (timeElps > 18){
            validL = 0;
            cout << "\n\nToo much time! ";
            cout << "Time elapsed was " << timeElps << " seconds.\n\n";
        }
    }
```

```
    if (skillL == 2){
        if (timeElps > 14){
            validL = 0;
            cout << "\n\nToo much time! ";
            cout << "Time elapsed was " << timeElps << " seconds.\n\n";
        }
    }
    if (skillL == 3){
        if (timeElps > 10){
            validL = 0;
            cout << "\n\nToo much time! ";
            cout << "Time elapsed was " << timeElps << " seconds.\n\n";
        }
    }

    return tLimit;

}

//This function is called by main.  It is used to write out the user name and
//their score to separate files.
void write(string name, int score){

    //Save the user name and score to files appending to the end.
    //Define files
    fstream outFile1;                   //User name
    fstream outFile2;                   //User score

    //Open the files
    //Advanced file i/o from Gaddis chapter 12
    outFile1.open("userNames.txt",ios::out | ios::app);
    outFile2.open("userScores.txt",ios::out | ios::app);

    //Output name and score to the files
    outFile1 << name << "\n";
    outFile2 << score << "\n";

    //Close the files
    outFile1.close();
    outFile2.close();

}

//This function is called by main.  It displays some basic instruction to
```

```
//play the game and waits for user to hit the enter key to continue.
void disInst (){

   //Display initial message to let the user know to turn on the caps lock
   cout << "***** TURN ON THE CAPS LOCK KEY *****\n";
   cout << "*****     TO PLAY THIS GAME     *****\n\n";
   cout << "\n\n\n  READY TO PLAY...   HIT ENTER... \n";
   cin.get();
}

//This function is called by main.  It returns the user name input by the
//player to be stored in a file with their score.
string getUsrNm (){

   //Declare variables
   string usrName;

   //Ask user for their name (up to 10 characters).
   cin.ignore();
   cout << "Enter your name (up to 10 characters): ";
   getline(cin, usrName);

   //Verify the length of the name.  Must be < 10 characters
   int inLen = usrName.length();

   //Keep asking user for input until a valid name is entered.
   while (inLen > 10){
      cout << "Enter your name (up to 10 characters): ";
      getline(cin, usrName);
      inLen = usrName.length();
   }

   return usrName;
}

//This function is called by main.  Prompts user if they want to play again.
char getAgain (){

   char again = 'N';

   cout << "\n\n   PLAY AGAIN (Y)? \n";
   cout << "hit any other key to exit...";
   cin >> again;
```

```
   return again;
}

//This function is called by main.  Diaplays ending message and a return 0.
int endGame(){

   //End the game to end
   cout << "\nThank you for playing!\n";

   return 0;
}

//This function is called by sortAray.  This searches the array for all
//scores by each of the 5 users and puts, the scores in a 2 dimensional
//array.  There can be up to 5 scores used for display.
void srch (vector<string> aName, vector<int> aScore, string users[],
       int numRecs){

   //Declare variables
   const int SIZE = 5;
   int usr = 5;
   int scr = 5;
   int scoreLst[usr][scr];        //scoreLst [row][column]
   int x = 0, y = 0;             //Subscripts
   bool found = false;           //Flag to indicate if the value was found
   string srchName;              //Hold name to search the aName array

   //Initialize the array scoreList array
   for (int row = 0; row < SIZE; row++)
   {
      for (int column = 0; column < SIZE; column++)
      {
         scoreLst[row][column] = 0;
      }
   }

   //srchName = users[0];
   for (int i = 0; i < SIZE; i++){
      srchName = users[i];
      x = 0;
      scr = 0;
      while (x < numRecs){
         if (aName[x] == srchName)             //If value found
         {
```

```
            found = true;                    //Set the flag
            usr = i;
            //Pull the score and store it for the user
            if (found){
                scoreLst[usr][scr] = aScore[x];
               scr++;
               found = false;
            }
         }
      x++;
      }
   }

   //Prompt user to continue after they've read the top 10 scores
   cout << "\n\n\n  HIT ENTER TO CONTINUE SCORES... \n";
   cin.get();

   //Push down 20 lines to clear the screen
   for (int j = 1; j < 21; j++){
      cout << "\n";
   }

   //Display the top 5 Players
   cout << "\t\t\t TOP 5 PLAYERS\n\n ";
   cout << "\t Player " << "\tScore 1" << "\tScore 2" << "\tScore 3"
          << "\tScore 4" << "\tScore 5" << endl;

   for (int row = 0; row < SIZE; row++)
   {
      cout << "\t" << setw(10) << users[row];
      for (int column = 0; column < SIZE; column++)
      {
         cout << "\t" << scoreLst[row][column];
      }
      cout << "\n";
   }
}
```