Project 2
<Hangman Game>

CIS-5
Tatiana DeJaco
02/11/15

Introduction

Title: Hangman Game

Hangman is a two player game where player one chooses a word at random and player two guesses its letters. Each time player two guesses incorrectly, player one draws a part of a stick figure. If player two guesses correctly, player one writes the letter where it is located in the word. The game continues until player two is able to reveal the word. However, if player one completes the drawing of a stick figure, then player two has lost the game.

I have altered the game so that instead of it being a two player game, it is now the computer versus a player. The computer chooses a word at random and the player guesses its letters. Also, I have given the user an option to ask for a hint. However, they can only have one hint. If they do ask for a hint, their score decreases by 3 points.

Summary

Project Size: 500+ lines
The number of variables: around 19
The number of method(s): 19

In the previous project I used concepts such as: if statements, if-else statements, switch statements, arrays, for-loops, while-loops, and enumeration types, and overloaded functions. I updated my project by adding concepts like 2-dimensional arrays, vectors, structures, formatted outputs and defaulted parameters.

The computer reads a word from a file containing a list of around 300 random words. At the end of the game, a file containing the chosen word and the user's results: whether they won or lost, the number of strikes, and their score.

The project itself took about a week.

I used the libraries: iostream, iomanip, string, fstream, and vector.

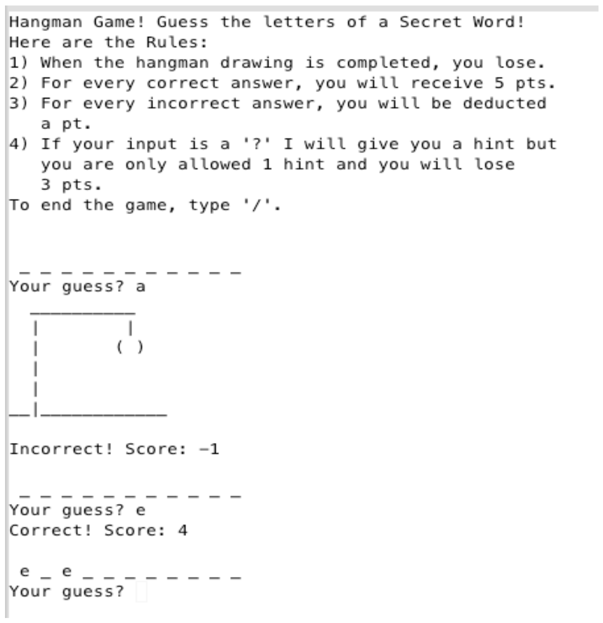
Sample Input and Output

Your guess? a <---- input

guess was incorrect so the hangman is drawn

Your guess? e <---- input

guess was correct so the blanks are filled

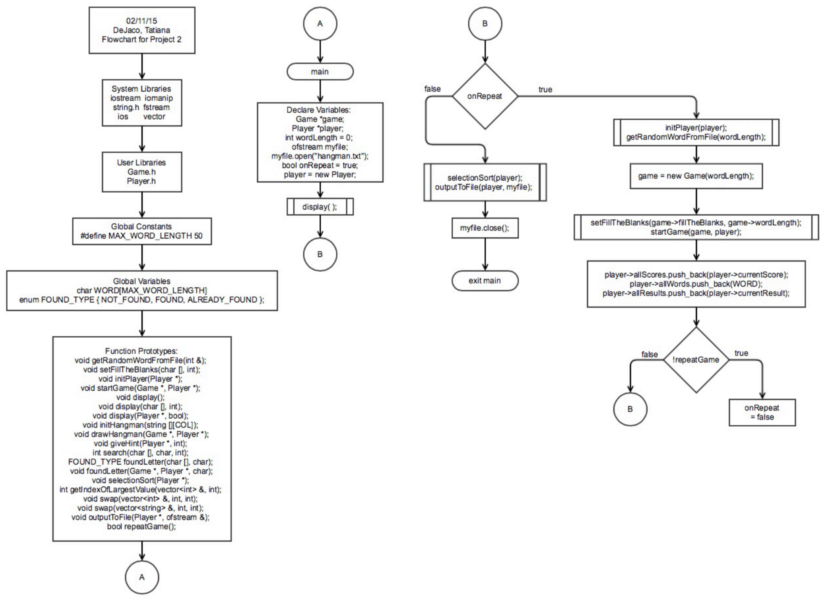


```
Hangman Game! Guess the letters of a Secret Word!
Here are the Rules:
1) When the hangman drawing is completed, you lose.
2) For every correct answer, you will receive 5 pts.
3) For every incorrect answer, you will be deducted
   a pt.
4) If your input is a '?' I will give you a hint but
   you are only allowed 1 hint and you will lose
   3 pts.
To end the game, type '/'.


 _ _ _ _ _ _ _ _ _ _ _
Your guess? a
  __________
  |        |
  |       ( )
  |
  |
__|___________

Incorrect! Score: -1

 _ _ _ _ _ _ _ _ _ _ _
Your guess? e
Correct! Score: 4

 e _ e _ _ _ _ _ _ _ _
Your guess?
```

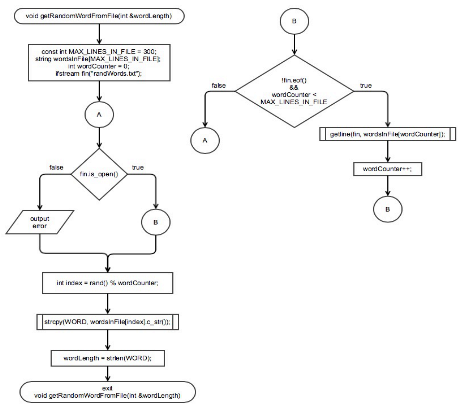
Pseudo Code

```
initialize hangman drawing

display fill in the blanks

while (the player has not yet found the word)

    read player's guess

    if  (guess == '?')
            then give them a hint about the word
    else if (guess ==  '/')
            then exit the game
    else
            then search for letter in word

    if  (player's found word == selected word)
            then player won the game
    else
            if  (incorrect guesses == 6 times)
                    then exit the game the player has lost

    display the fill in the blanks after every turn

display their results

continue the game until player no longer wants to play
```

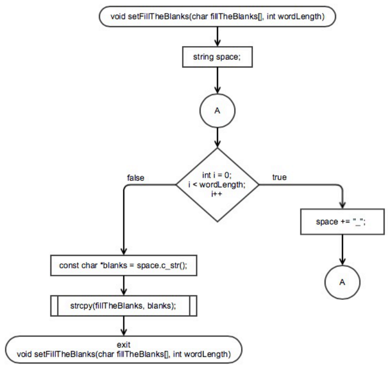
Flowchart for main(int argc, char** argv)


02/11/15
DeJaco, Tatiana
Flowchart for Project 2
System Libraries
iostream iomanip
string.h fstream
ios vector
User Libraries
Game.h
Player.h
Global Constants
#define MAX_WORD_LENGTH 50
Global Variables
char WORD[MAX_WORD_LENGTH]
enum FOUND_TYPE { NOT_FOUND, FOUND, ALREADY_FOUND };
Function Prototypes:
void getRandomWordFromFile(int &);
void setFillTheBlanks(char [], int);
void initPlayer(Player *);
void startGame(Game *, Player *);
void display();
void display(char [], int);
void display(Player *, bool);
void initHangman(string [][COL]);
void drawHangman(Game *, Player *);
void giveHint(Player *, int);
int search(char [], char, int);
FOUND_TYPE foundLetter(char [], char);
void foundLetter(Game *, Player *, char);
void selectionSort(Player *);
int getIndexOfLargestValue(vector<int> &, int);
void swap(vector<int> &, int, int);
void swap(vector<string> &, int, int);
void outputToFile(Player *, ofstream &);
bool repeatGame();
A
A
main
Declare Variables:
Game *game;
Player *player;
int wordLength = 0;
ofstream myfile;
myfile.open("hangman.txt");
bool onRepeat = true;
player = new Player;
display( );
B
B
onRepeat
false
true
selectionSort(player);
outputToFile(player, myfile);
myfile.close();
exit main
initPlayer(player);
getRandomWordFromFile(wordLength);
game = new Game(wordLength);
setFillTheBlanks(game->fillTheBlanks, game->wordLength);
startGame(game, player);
player->allScores.push_back(player->currentScore);
player->allWords.push_back(WORD);
player->allResults.push_back(player->currentResult);
!repeatGame
false
true
B
onRepeat
= false

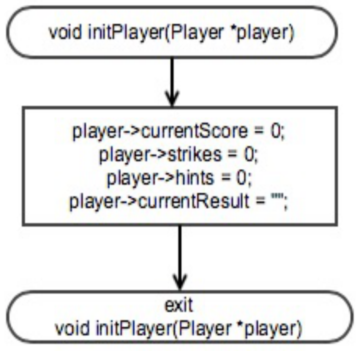
Flowchart for void getRandomWordFromFile(int &);


void getRandomWordFromFile(int &wordLength)
const int MAX_LINES_IN_FILE = 300;
string wordsInFile[MAX_LINES_IN_FILE];
int wordCounter = 0;
ifstream fin("randWords.txt");
A
fin.is_open()
false
true
output
error
B
int index = rand() % wordCounter;
strcpy(WORD, wordsInFile[index].c_str());
wordLength = strlen(WORD);
exit
void getRandomWordFromFile(int &wordLength)
B
!fin.eof()
&&
wordCounter <
MAX_LINES_IN_FILE
false
true
A
getline(fin, wordsInFile[wordCounter]);
wordCounter++;
B

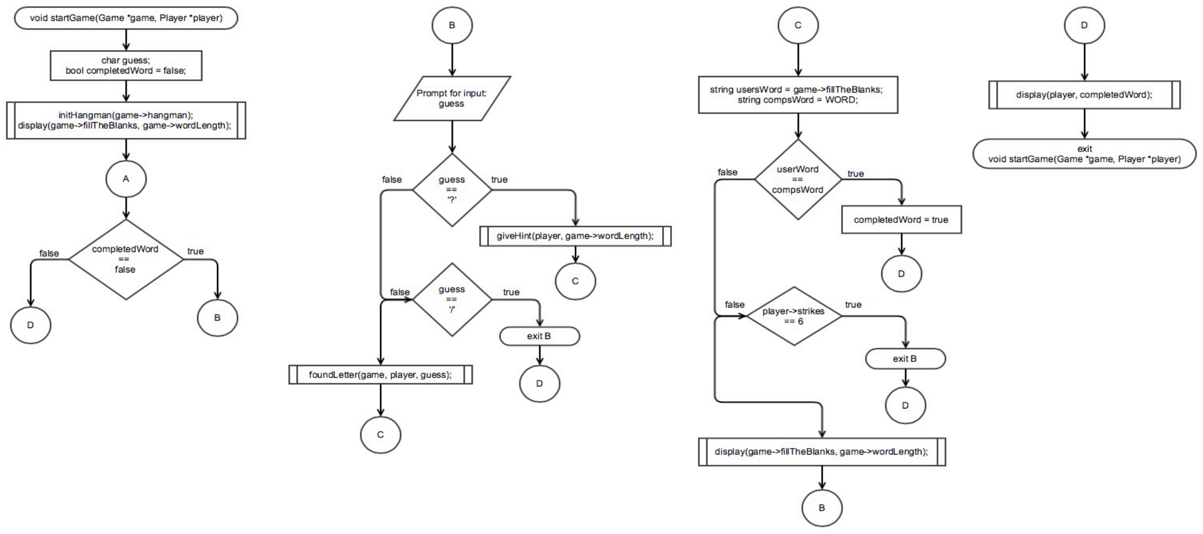
Flowchart for void setFillTheBlanks(char [], int);


void setFillTheBlanks(char fillTheBlanks[], int wordLength)
string space;
A
int i = 0;
i < wordLength;
i++
false
true
space += "_";
A
const char *blanks = space.c_str();
strcpy(fillTheBlanks, blanks);
exit
void setFillTheBlanks(char fillTheBlanks[], int wordLength)

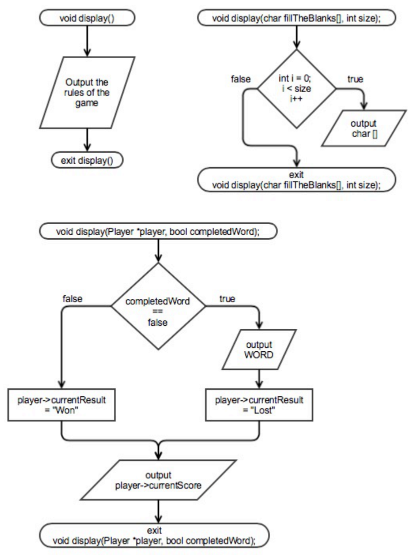
Flowchart for void initPlayer(Player *);


void initPlayer(Player *player)
player->currentScore = 0;
player->strikes = 0;
player->hints = 0;
player->currentResult = "";
exit
void initPlayer(Player *player)

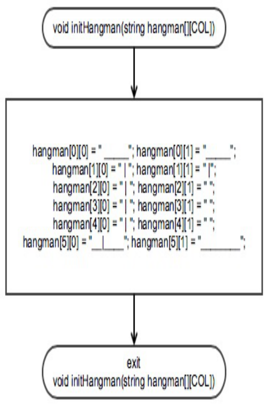
Flowchart for void startGame(Game *, Player *);


void startGame(Game *game, Player *player)
char guess;
bool completedWord = false;
initHangman(game->hangman);
display(game->fillTheBlanks, game->wordLength);
A
completedWord == false
false
true
D
B
B
Prompt for input: guess
guess == '?'
false
true
giveHint(player, game->wordLength);
C
guess == '/'
false
true
exit B
D
foundLetter(game, player, guess);
C
C
string usersWord = game->fillTheBlanks;
string compsWord = WORD;
userWord == compsWord
false
true
completedWord = true
D
player->strikes == 6
false
true
exit B
D
display(game->fillTheBlanks, game->wordLength);
B
D
display(player, completedWord);
exit
void startGame(Game *game, Player *player)

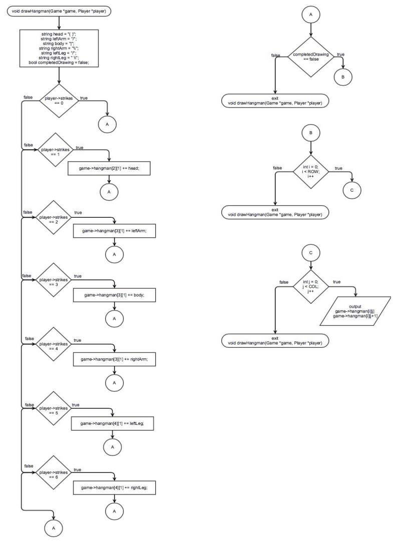
Flowchart for void display(), void display(char [], int), void display(Player *, bool);


void display()
Output the rules of the game
exit display()
void display(char fillTheBlanks[], int size);
int i = 0; i < size i++
false
true
output char []
exit
void display(char fillTheBlanks[], int size);
void display(Player *player, bool completedWord);
completedWord == false
false
true
output WORD
player->currentResult = "Won"
player->currentResult = "Lost"
output player->currentScore
exit
void display(Player *player, bool completedWord);

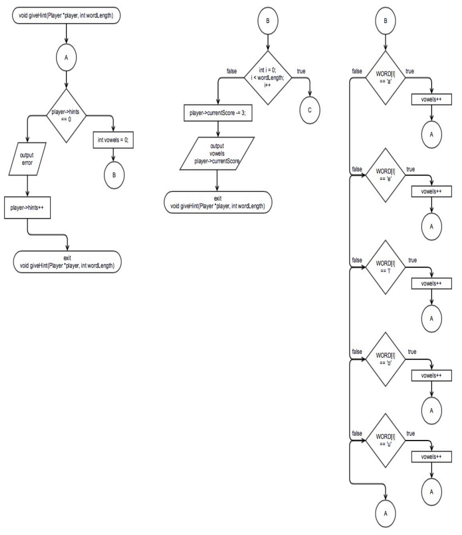
Flowchart for void initHangman(string [][COL]);


void initHangman(string hangman[][COL])
hangman[0][0] = " _____"; hangman[0][1] = " _____";
hangman[1][0] = " | "; hangman[1][1] = " |";
hangman[2][0] = " | "; hangman[2][1] = " ";
hangman[3][0] = " | "; hangman[3][1] = " ";
hangman[4][0] = " | "; hangman[4][1] = " ";
hangman[5][0] = " __|____"; hangman[5][1] = " ________";
exit
void initHangman(string hangman[][COL])

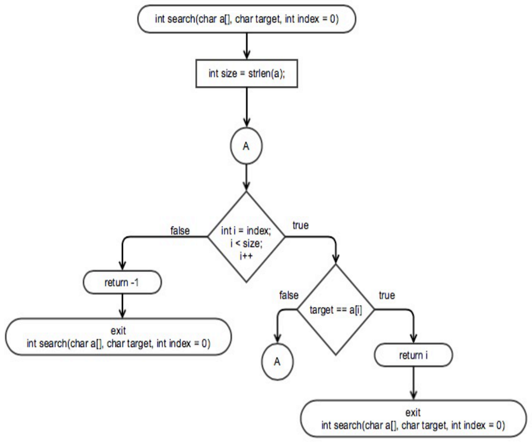
Flowchart for void drawHangman(Game *, Player *);


void drawHangman(Game *game, Player *player)
string head = "( )";
string leftArm = "/";
string body = "|";
string rightArm = "\\";
string leftLeg = "/";
string rightLeg = " \\";
bool completedDrawing = false;
player->strikes == 0
player->strikes == 1
game->hangman[2][1] += head;
player->strikes == 2
game->hangman[3][1] += leftArm;
player->strikes == 3
game->hangman[3][1] += body;
player->strikes == 4
game->hangman[3][1] += rightArm;
player->strikes == 5
game->hangman[4][1] += leftLeg;
player->strikes == 6
game->hangman[4][1] += rightLeg;
true
false
A
completedDrawing == false
B
exit void drawHangman(Game *game, Player *player)
int i = 0; i < ROW; i++
C
int j = 0; j < COL; j++
output game->hangman[i][j] game->hangman[i][j+1]

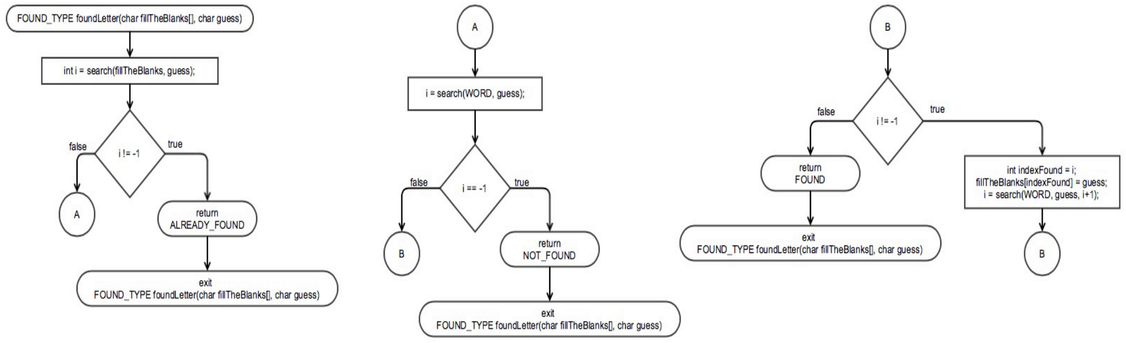
Flowchart for void giveHint(Player *, int);


void giveHint(Player *player, int wordLength)
A
player->hints == 0
output error
player->hints++
int vowels = 0;
B
exit void giveHint(Player *player, int wordLength)
int i = 0; i < wordLength; i++
false
true
player->currentScore -= 3;
output vowels player->currentScore
C
WORD[I] == 'a'
WORD[I] == 'e'
WORD[I] == 'i'
WORD[I] == 'o'
WORD[I] == 'u'
vowels++

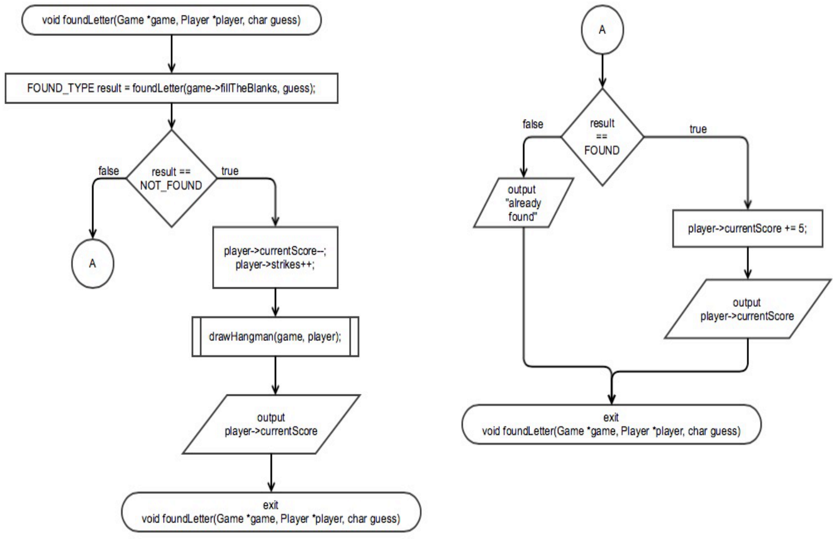
Flowchart for int search(char [], char, int);


int search(char a[], char target, int index = 0)
int size = strlen(a);
A
int i = index; i < size; i++
false
true
return -1
exit int search(char a[], char target, int index = 0)
target == a[i]
false
true
A
return i
exit int search(char a[], char target, int index = 0)

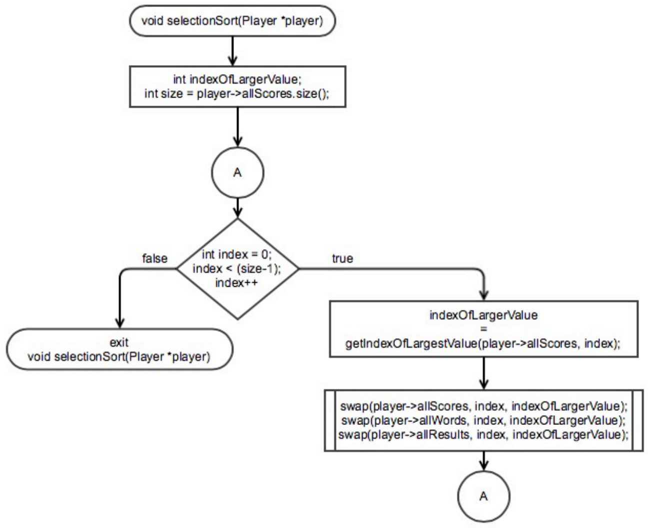
Flowchart for FOUND_TYPE foundLetter(char [], char);


FOUND_TYPE foundLetter(char fillTheBlanks[], char guess)
int i = search(fillTheBlanks, guess);
i != -1
false
true
A
return ALREADY_FOUND
exit FOUND_TYPE foundLetter(char fillTheBlanks[], char guess)
A
i = search(WORD, guess);
i == -1
false
true
B
return NOT_FOUND
exit FOUND_TYPE foundLetter(char fillTheBlanks[], char guess)
B
i != -1
false
true
return FOUND
exit FOUND_TYPE foundLetter(char fillTheBlanks[], char guess)
int indexFound = i; fillTheBlanks[indexFound] = guess; i = search(WORD, guess, i+1);
B

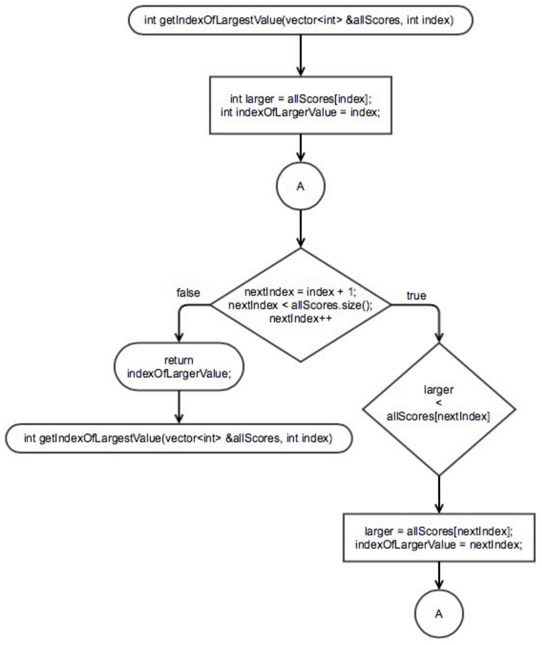
Flowchart for void foundLetter(Game *, Player *, char);


void foundLetter(Game *game, Player *player, char guess)
FOUND_TYPE result = foundLetter(game->fillTheBlanks, guess);
result == NOT_FOUND
false
true
A
player->currentScore--; player->strikes++;
drawHangman(game, player);
output player->currentScore
exit void foundLetter(Game *game, Player *player, char guess)
A
result == FOUND
false
true
output "already found"
player->currentScore += 5;
output player->currentScore
exit void foundLetter(Game *game, Player *player, char guess)

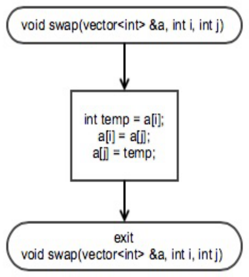
Flowchart for void selectionSort(Player *);


void selectionSort(Player *player)
int indexOfLargerValue;
int size = player->allScores.size();
A
int index = 0;
index < (size-1);
index++
false
true
exit
void selectionSort(Player *player)
indexOfLargerValue
=
getIndexOfLargestValue(player->allScores, index);
swap(player->allScores, index, indexOfLargerValue);
swap(player->allWords, index, indexOfLargerValue);
swap(player->allResults, index, indexOfLargerValue);
A

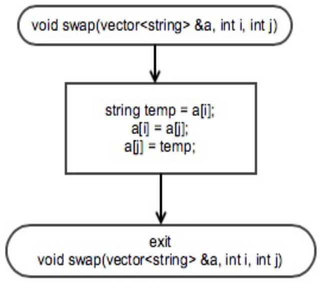
Flowchart for int getIndexOfLargestValue(vector<int> &, int);


int getIndexOfLargestValue(vector<int> &allScores, int index)
int larger = allScores[index];
int indexOfLargerValue = index;
A
nextIndex = index + 1;
nextIndex < allScores.size();
nextIndex++
false
true
return
indexOfLargerValue;
int getIndexOfLargestValue(vector<int> &allScores, int index)
larger
<
allScores[nextIndex]
larger = allScores[nextIndex];
indexOfLargerValue = nextIndex;
A

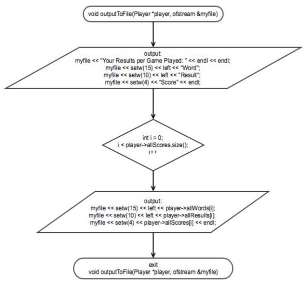
Flowchart for void swap(vector<int> &, int, int);


void swap(vector<int> &a, int i, int j)
int temp = a[i];
a[i] = a[j];
a[j] = temp;
exit
void swap(vector<int> &a, int i, int j)

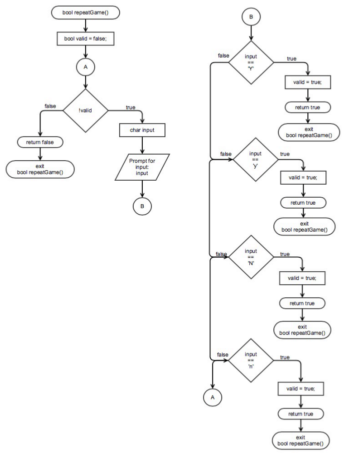
Flowchart for void swap(vector<string> &, int, int);


void swap(vector<string> &a, int i, int j)
string temp = a[i];
a[i] = a[j];
a[j] = temp;
exit
void swap(vector<string> &a, int i, int j)

Flowchart for void outputToFile(Player *, ofstream &);


void outputToFile(Player *player, ofstream &myfile)
output:
myfile << "Your Results per Game Played: " << endl << endl;
myfile << setw(15) << left << "Word";
myfile << setw(10) << left << "Result";
myfile << setw(4) << "Score" << endl;
int i = 0;
i < player->allScores.size();
i++
output:
myfile << setw(15) << left << player->allWords[i];
myfile << setw(10) << left << player->allResults[i];
myfile << setw(4) << player->allScores[i] << endl;
exit
void outputToFile(Player *player, ofstream &myfile)

Flowchart for bool repeatGame();



Major Variables

Type	Variable Name	Description	Location
Char []	WORD	Contains a random word read in from a file.	Main.cpp (Global Variable)
Game	game	A structure containing commonly used variables needed for the game.	Main.cpp
Player	player	A structure used like a database for player.	Main.cpp

int	wordLength	The length of the random word read in from a file.	Main.cpp

C++ Constructs

Chapter	New Syntax and Keywords	Location
2	cout	startGame(Game *, Player *);
	cin	startGame(Game *, Player *);
	output formatting	outputToFile(Player *, ofstream &);
	int	main(int argc, char** argv)
	char	main(int argc, char** argv)
	bool	startGame(Game *, Player *);
	string	drawHangman(Game *, Player *)
	assignment operator (+=)	drawHangman(Game *, Player *)
	arithmetic operator (+,-,*, /)	foundLetter(Game *, Player *, char)
	increment operator (++)	foundLetter(Game *, Player *, char)
	decrement operator (--)	foundLetter(Game *, Player *, char)
3	enumeration type	main.cpp (Global Variable)
	if-else statements	foundLetter(Game *, Player *, char)
	switch statements	giveHint(Player *, int)
	break	giveHint(Player *, int)
	while loops	foundLetter(char [], char)
	do while loops	startGame(Game *, Player *)
	for loops	search(char [], char, int)
4	functions	main.cpp
	pass by value	display(char [], int);
	global variable	main.cpp
	overloaded functions	- foundLetter(char [], char) - foundLetter(Game *, Player *, char)
5	void functions	main.cpp
	returning primitive data type	foundLetter(char [], char)
	procedural abstraction	startGame(Game *, Player *)
6	ifstream	outputToFile(Player *, ofstream &)
	ofstream	outputToFile(Player *, ofstream &)
	defaulted arguments	search(char [], char, int)

7	arrays	struct Game.h
	passing arrays between functions	search(char [], char, int)
	multi-dimensional array	struct Game.h
	searching an array	search(char [], char, int);
	sorting an array	selectionSort(Player *);
8	string array[]	struct Game.h
	vectors	struct Player.h
10	structures	struct Game.h, struct Player.h

Reference

1. http://www.stackoverflow.com
2. http://www.cplusplus.com
3. textbook

Program

```
/*
 * File:   main.cpp
 * Author: Tati
 * Created on February 8, 2015, 1:24 PM
 * Hangman Game
 */

// System Libraries
#include <iostream> // Reads Inputs
#include <iomanip>  // Formatting Output
#include <string.h>
#include <fstream>
#include <ios>         // Reads & Writes Files
#include <vector>
using namespace std;

//User Libraries
#include "Game.h"
#include "Player.h"

#define MAX_WORD_LENGTH 50

// Global Constants
char WORD[MAX_WORD_LENGTH];
enum FOUND_TYPE { NOT_FOUND, FOUND, ALREADY_FOUND }; // Used for comparing results

// Function Prototypes
void       getRandomWordFromFile(int &);              // Gets a random word from a file
void       setFillTheBlanks(char [], int);            // Sets how many blanks the user has to fill
void       initPlayer(Player *);                      // Initialize player without initializing allScores and allWords
void       startGame(Game *, Player *);               // Play the game
void       display();                              // Display game rules to player
void       display(char [], int);                     // Display's an array
void       display(Player *, bool);                   // Output if the user won or lost
void       initHangman(string [][COL]);               // initialize hangman drawing
void       drawHangman(Game *, Player *);             //  Draws Hangman
void       giveHint(Player *, int);                   // Give the user a hint when wanted
int        search(char [], char, int);                // Linear Search through array for target
FOUND_TYPE foundLetter(char [], char);                // Searches for the letter given by user
void       foundLetter(Game *, Player *, char);       // Searches for the letter given by user
void       selectionSort(Player *);                   // Sorts player's results
int        getIndexOfLargestValue(vector<int> &, int); // Used in and for selection sorting
void       swap(vector<int> &, int, int);             // Swaps the values: for scores
void       swap(vector<string> &, int, int);          // Swaps the values: for words
void       outputToFile(Player *, ofstream &);        // Output the word, # of strikes, and score to a file
```

```
bool      repeatGame();                         // Returns whether or not they want to play again

// Execution Here
int main(int argc, char** argv)
{
   // initialize random seed
   srand (time(NULL));

   // Declare Variables
   Game   *game;
   Player *player; // Create player database
   int wordLength = 0; // the length of the word selected

   // Open/Create a new file
   ofstream myfile; // Output file for the hangman results
   myfile.open("hangman.txt");

   display(); // Display game rules

   bool onRepeat = true; // assume player want to continue playing
   player = new Player;  // initialize database
   while (onRepeat)
   {
      initPlayer(player);
      getRandomWordFromFile(wordLength);
      game = new Game(wordLength); // Initialize game
      setFillTheBlanks(game->fillTheBlanks, game->wordLength);
      startGame(game, player);

      // Place score in vector
      player->allScores.push_back(player->currentScore);
      player->allWords.push_back(WORD);
      player->allResults.push_back(player->currentResult);

      if ( !repeatGame() )
         onRepeat = false;
      else
         cout << endl;
   }
   cout << endl;
   cout << "Your results were outputted to file: hangman.txt" << endl;

   selectionSort(player);       // Sort player's scores
   outputToFile(player, myfile); // output results

   // Close the hangman file
   myfile.close();
}

/****************************************************
 * Gets a random word from a file
 * @param wordLength - the length of the random word
 ****************************************************/
void getRandomWordFromFile(int &wordLength)
{
   const int MAX_LINES_IN_FILE = 300;
   string    wordsInFile[MAX_LINES_IN_FILE]; // array of words from file
   int       wordCounter = 0;              // the number of words in file

   // Computer will choose one word from the file of random words
   ifstream fin("randWords.txt"); // opening an input stream for file

   // Checking whether file could be opened or not. If file does not exist
   // or don't have read permissions, file stream could not be opened.
   if(fin.is_open())
   {
      // this loop run until end of file (eof) does not occur
      while(!fin.eof() && wordCounter < MAX_LINES_IN_FILE)
      {
         // Read a complete line into the array. Each line
         // contains one word.
         getline(fin, wordsInFile[wordCounter]);
         wordCounter++;
      }
   }
   else // file could not be opened
      cout << "File could not be opened." << endl;

   // Pick a string from the list
   int index  = rand() % wordCounter;
   strcpy(WORD, wordsInFile[index].c_str()); // Set the rand string to be the word
   wordLength = strlen(WORD);                // the length of the chosen word
}

/*******************************************************
 * Create the fill in the blanks that user will fill
 * throughout the game
 * @param fillTheBlanks - character array to initialize
 * @param wordLength - the length of the selected word
 *******************************************************/
void setFillTheBlanks(char fillTheBlanks[], int wordLength)
{
   // Create empty string that will be filled as the user guesses correctly
   string space;
   for (int i = 0; i < wordLength; i++)
      space += "_";

   const char *blanks = space.c_str(); // Create the empty string
   strcpy(fillTheBlanks, blanks);
}

/*********************************************************************
 * Initializes all variables in struct Player
 * @param player - player to be initialized
 *********************************************************************/
void initPlayer(Player *player)
{
   player->currentScore  = 0;
   player->strikes       = 0;
   player->hints         = 0;
   player->currentResult = "";
}

/*********************************************************************
 * Begin the game
 * @param game   - game variables
 * @param player - player variables
 *********************************************************************/
void startGame(Game *game, Player *player)
{
   // Declare Variables
   char guess; // User's guess

   initHangman(game->hangman);                    // Initialize Hangman
   //cout << "WORD was " << WORD << endl;         // Debugging purposes
   display(game->fillTheBlanks, game->wordLength); // display fill the blanks

   bool completedWord = false; // true if user completely filled the blanks
   do
   {
      // Prompt user for input
      cout << "Your guess? ";
      cin >> guess;

      // If user asks for a hint, give them one at random
      if (guess == '?')
         giveHint(player, game->wordLength);
      else if (guess == '/') // exit game when player types in '/'
         break;
      else
         foundLetter(game, player, guess);

      // Check if the blanks are filled and matches the computer's word
      string usersWord = game->fillTheBlanks; // User's word
      string compsWord = WORD;                // Comp's word
      if (usersWord == compsWord) // if blanks match the comp's word
         completedWord = true;
      else // blanks were not filled
      {
         if (player->strikes == 6) // Check if the user used up their strikes
            break;
      }
      display(game->fillTheBlanks, game->wordLength); // display blanks
   }
   while (completedWord == false);

   display(player, completedWord); // Display results to user
}

/***************************************************************************
 * Display the rules of the game
 ***************************************************************************/
void display()
{
   cout << "Hangman Game! Guess the letters of a Secret Word!"         << endl;
   cout << "Here are the Rules: "                                      << endl;
   cout << "1) When the hangman drawing is completed, you lose."       << endl;
   cout << "2) For every correct answer, you will receive 5 pts."      << endl;
   cout << "3) For every incorrect answer, you will be deducted"       << endl;
   cout << "   a pt."                                                  << endl;
   cout << "4) If your input is a '?' I will give you a hint but"      << endl;
   cout << "   you are only allowed 1 hint and you will lose    "      << endl;
   cout << "   3 pts."                                                 << endl;
   cout << "To end the game, type '/'."                                << endl;
   cout << endl;
}

/***************************************************************************
 * Displays the fill in the blanks
 * @param fillTheBlanks - character array to display
 * @param wordLength    - the length of the word
 ***************************************************************************/
void display(char fillTheBlanks[], int wordLength)
{
   // Display the blanks
   cout << endl;
   for (int i = 0; i < wordLength; i++)
      cout << " " << fillTheBlanks[i];
   cout << endl;
}

/***************************************************************************
 * Displays the user's results when game is over
 * @param player        - player variables
 * @param completedWord - True if user completed the word else false
 ***************************************************************************/
void display(Player *player, bool completedWord)
{
   cout << endl;
   // Set Score and Output results
   if (completedWord == false) // Word wasn't completed
   {
      cout << "You Lose! The word was " << WORD;
      player->currentResult = "Lost";
   }
   else // Word was completed
   {
      cout << "You have completed the game! Congratulations!";
      player->currentResult = "Won";
   }
   cout << endl;
   cout << "Score: " << player->currentScore;
}

/***************************************************************************
 * Initialize the hangman drawing
 * @param hangman - 2D array used to draw the hangman
 ***************************************************************************/
```

```
void initHangman(string hangman[][COL])
{
   hangman[0][0] = "  _____"; hangman[0][1] = "_____";
   hangman[1][0] = "  |    "; hangman[1][1] = "    |";
   hangman[2][0] = "  |    "; hangman[2][1] = "   ";
   hangman[3][0] = "  |    "; hangman[3][1] = "   ";
   hangman[4][0] = "  |    "; hangman[4][1] = "   ";
   hangman[5][0] = "__|____"; hangman[5][1] = "________";
}

/***************************************************************************
 * Draw parts of the hangman as the user guesses wrong
 * @param game   - game variables
 * @param player - player variables
 ***************************************************************************/
void drawHangman(Game *game, Player *player)
{
   string head     = "( )";
   string leftArm  = "/";
   string body     = "|";
   string rightArm = "\\";
   string leftLeg  = "/";
   string rightLeg = " \\";

   bool completedDrawing = false;
   if      (player->strikes == 0); // do nothing
   else if (player->strikes == 1)
      game->hangman[2][1] += head;     // draw head

   else if (player->strikes == 2)
      game->hangman[3][1] += leftArm;  // draw left arm

   else if (player->strikes == 3)
      game->hangman[3][1] += body;     // draw body

   else if (player->strikes == 4)
      game->hangman[3][1] += rightArm; // draw right arm

   else if (player->strikes == 5)
      game->hangman[4][1] += leftLeg;  // draw left leg

   else if (player->strikes == 6)
      game->hangman[4][1] += rightLeg; // draw right leg
   else
      completedDrawing = true;

   // Display Drawing if not complete
   if (completedDrawing == false)
   {
      for (int i = 0; i < ROW; i++)
      {
         for (int j = 0; j < COL - 1; j++)
            cout << game->hangman[i][j]
                 << game->hangman[i][j+1] << endl;
      }
   }
}

/***************************************************************************
 * Gives a hint to the user when wanted
 * @param player     - player variables
 * @param wordLength - the length of the selected word
 ***************************************************************************/
void giveHint(Player *player, int wordLength)
{
   // If user asks for a hint, give them one at random
   if (player->hints == 0)
   {
      int vowels = 0; // # of vowels in the word

      // Go through the WORD and count how many vowels are in there
      for (int i = 0; i < wordLength; i++)
      {
         switch(WORD[i])
         {
            case 'a': { vowels++; break; }
            case 'i': { vowels++; break; }
            case 'u': { vowels++; break; }
            case 'e': { vowels++; break; }
            case 'o': { vowels++; break; }
            default: break;
         };
      }
      player->currentScore -= 3;
      cout << "The word has: " << vowels << " vowel(s)." << endl;
      cout << "Score: "        << player->currentScore;
   }
   else
      cout << "No more hints are available." << endl;
   player->hints++;
}

/***************************************************************************
 * A Linear Search for target in a character array
 * @param a      - character array to search through
 * @param target - character to search for
 * @param index  - where to begin the search
 * @return -1 if the character was not found
 ***************************************************************************/
int search(char a[], char target, int index = 0)
{
   int size = strlen(a);
   for (int i = index; i < size; i++)
   {
      if (target == a[i])
         return i;
   }
   return -1;
}

/***************************************************************************
 * Searches for guess and returns if found, already found, or not found
 * @param fillTheBlanks - character array to search through
 *                        to check if character was already found
 * @param guess         - search target
 * @return FOUND_TYPE
 ***************************************************************************/
FOUND_TYPE foundLetter(char fillTheBlanks[], char guess)
{
   // Check first if letter was already found
   int i = search(fillTheBlanks, guess);
   if (i != -1) // if not NULL
      return ALREADY_FOUND;

   // Check if letter was not found in fills
   i = search(WORD, guess);
   if (i == -1) // if null
      return NOT_FOUND;

   // Check if letter was found in the WORD
   while (i != -1)
   {
      int indexFound = i;
      fillTheBlanks[indexFound] = guess; // fill in the blanks
      i = search(WORD, guess, i+1);
   }
   return FOUND;
}

/***************************************************************************
 * Increments/Decrements score or strikes depending on search results
 * @param game   - game variables
 * @param player - player variables
 * @param guess  - user's guess
 ***************************************************************************/
void foundLetter(Game *game, Player *player, char guess)
```

```
{
   // Find the letter guessed by user in the word
   FOUND_TYPE result = foundLetter(game->fillTheBlanks, guess);
   if (result == NOT_FOUND)
   {
      player->currentScore--;
      player->strikes++;
      // Draw hangman when user guess wrong
      drawHangman(game, player);
      cout << "\nIncorrect! ";
      cout << "Score: " << player->currentScore << endl;
   }
   else
   {
      if (result == FOUND)
      {
         player->currentScore += 5;
         cout << "Correct! ";
         cout << "Score: " << player->currentScore << endl;
      }
      else
         cout << "Letter was already found." << endl;
   }
}

/***************************************************************************
 * Selection Sorts player: scores, words, results
 * @param player - player variables
 ***************************************************************************/
void selectionSort(Player *player)
{
   int indexOfLargerValue;
   int size = player->allScores.size();

   for (int index = 0; index < (size-1); index++)
   {
      indexOfLargerValue = getIndexOfLargestValue(player->allScores, index);
      swap(player->allScores,  index, indexOfLargerValue);
      swap(player->allWords,   index, indexOfLargerValue);
      swap(player->allResults, index, indexOfLargerValue);
   }
}

/***************************************************************************
 * Grabs the index of the largest value in vector
 * @param allScores - the vector to sort
 * @param index     - index of current larger value
 * @return the index of the larger value
 ***************************************************************************/
int getIndexOfLargestValue(vector<int> &allScores, int index)
{
   int larger = allScores[index];
   int indexOfLargerValue = index;

   // Go through the array and compare the next values in the array
   // to the current smaller value.
   for (int nextIndex = index + 1; nextIndex < allScores.size(); nextIndex++)
   {
      // If the value of smaller is greater than that of a[nextIndex]
      // assign smaller to the next index's value and
      // assign the index of the smaller value to be the next index.
      if (larger < allScores[nextIndex])
      {
         larger = allScores[nextIndex];
         indexOfLargerValue = nextIndex;
      }
   }
   return indexOfLargerValue;
}

/***************************************************************************
 * Swaps values in a vector
 * @param a - vector to sort
 * @param i - index to sort
 * @param j - next index to sort
 ***************************************************************************/
void swap(vector<int> &a, int i, int j)
{
  // swaps values.
  int temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}

/***************************************************************************
 * Swaps values in a vector
 * @param a - vector to sort
 * @param i - index to sort
 * @param j - next index to sort
 ***************************************************************************/
void swap(vector<string> &a, int i, int j)
{
  // swaps values.
  string temp = a[i];
  a[i] = a[j];
  a[j] = temp;
}

/***************************************************************************
 * Output the user's results to a file
 * @param player - player variables
 * @param myfile - the file to output into
 ***************************************************************************/
void outputToFile(Player *player, ofstream &myfile)
{
   // Output a file when game is complete
   myfile << "Your Results per Game Played: " << endl << endl;
   myfile << setw(15) << left << "Word";
   myfile << setw(10) << left << "Result";
   myfile << setw(4)  << "Score" << endl;

   for (int i = 0; i < player->allScores.size(); i++)
   {
      myfile << setw(15) << left << player->allWords[i];
      myfile << setw(10) << left << player->allResults[i];
      myfile << setw(4)  << player->allScores[i] << endl;
   }
   myfile << endl;
}

/***************************************************************************
 * Asks user if they would like to play again
 * @return true if user wants to repeat game else false
 ***************************************************************************/
bool repeatGame()
{
   cout << endl << endl;
   bool valid = false;
   while ( !valid )
   {
      char input;
      cout << "Play Again(y/n)? ";
      cin  >> input;

      switch(input)
      {
         case 'Y':
         case 'y': { valid = true; return true;  break; }
         case 'N':
         case 'n': { valid = true; return false; break; }
         default:  { break; }
      };
   }
   return false;
}
```

```
/*
 * File:   Game.h
 * Author: Tati
 * Created on February 11, 2015, 12:38 PM
 */

#ifndef GAME_H
#define     GAME_H

#define ROW 6
#define COL 2

struct Game
{
    int         wordLength;
    char        *fillTheBlanks;
    std::string hangman[ROW][COL];

    // Initializes variables
    Game(int w)
    {
        wordLength    = w;
        fillTheBlanks = new char[w];
        for (int i = 0; i < ROW; i++) // Initialize the hangman string to empty
            for (int j = 0; j < COL; j++)
                hangman[i][j] = "";
    }
};

#endif      /* GAME_H */

/*
 * File:   Player.h
 * Author: Tati
 * Created on February 9, 2015, 10:36 PM
 * Database for player
 */

#ifndef PLAYER_H
#define     PLAYER_H

#include <vector>
#include <string>
#include <string.h>

struct Player
{
    // Initializes variables
    Player(): currentScore(0), strikes(0), hints(0), currentResult("") {}
    // Current Game Variables
    int                 currentScore;   // User's score
    int                 strikes;        // User's strikes
    int                 hints;          // User's number of hints used
    std::string         currentResult;  // User's result
    // All Game Variables
    std::vector<int>    allScores;
    std::vector<string> allWords;
    std::vector<string> allResults;
};

#endif      /* PLAYER_H */
```