SpellMaster Testing and Specifications

Bitbusters

Yara Al Tassi – Hanan Zwaihed – Yasmin Halabi

Set.c:

Set* createSet()

Requires: nothing

Effects: creates an empty Set and returns it

Test Suite:

- Test Case1: just call the function
    - Expected Output: creates an empty set

Void addToSet (Set* set, const char* element)

Requires: nothing

Effects:

- If either set or element is null it returns
- If the element is already in the set it prints “element already in set” and returns
- Else it adds the element to the set

Test Suite:

- Test Case1: set= NULL
    - Expected Output: function returns
- Test case 2: element= NULL
    - Expected Output: function returns
- Test Case 3: element is already in the set

    - Expected Output: function prints “element already in set” and returns
- Test case 4: elements is not in the set
    - Expected Output: element gets added to the set

Int isInSet (const Set* set, const char* element)

Requires: nothing

Effects:

- If either set or element is null returns 0 (false)
- If element is in the set returns 1 (true)
- If element is not in the set returns 0 (false)

Test Suite:

- Test Case1: set is empty
    - Expected Output: function returns 0
- Test case 2: element= NULL
    - Expected Output: function returns 0
- Test Case 3: element is in set
    - Expected Output: function returns 1
- Test Case 4: element is not in set
    - Expected Output: function returns 0

Int getSize (const Set* set)

Requires: nothing

Effects:

- If set is empty (set-> head is NULL) returns 0
- Else returns the number of elements in the set

Test Suite:

- Test Case1: set is empty
    - Expected Output: returns 0
- Test Case 2: set has elements
    - Expected Output: returns number of elements

Void printSet (const Set* set)

Requires: nothing

Effects:

- If set is empty (set-> head is NULL) prints “set is empty”
- Else prints every 5 elements on a line separated by tab spaces

Test Suite:

- Test Case1: set is empty
    - Expected Output: prints “set is empty”
- Test Case 2: set has less than 5 elements
    - Expected Output: prints the elements on the same line separated by tab spaces
- Test Case 3: set has more than 5 elements
    - Expected Output:: prints every 5 elements on the same line separated by tab spaces (each 5 elements are on their own line)

char** SetToArr (const Set* set)

Requires: nothing

Effects:

- If set is null returns null
- Else returns an array containing the elements of the set

Test Suite:

- Test Case1: set= NULL
    - Expected Output: returns null
- Test case 2: set has elements
    - Expected Output: returns an array containing the elements in the set

Void freeWordArr (char** wordArr, int size)

Requires: size to be the actual size of the array and wordArr not to be NULL

Effects: frees wordArr from memory

Test Suite:

- Test Case1: wordArr has elements
    - Expected Output: wordArr's elements are deleted and memory is freed

Void removeFromSet (Set* set, const char* element)

Requires: nothing

Effects:

- If set or element are null OR the element is not found in the set, returns (does nothing)
- If element is found, removes it from the set

Test Suite:

- Test Case1: set= NULL
    - Expected Output: function returns
- Test case 2: element= NULL
    - Expected Output: function returns
- Test Case 3: element is in the set
    - Expected Output: element gets deleted from the set

- Test Case 4: only one element in the set (element to be removed)
    - Expected Output: element gets deleted from the set
- Test case 5: element is not in the set
    - Expected Output: nothing happens

Void clearSet (Set* set)

Requires: nothing

Effects:

- If set is empty (set->head=NULL) does nothing
- Else clears the set (removes all elements and frees them from memory)

Test Suite:

- Test Case1: set is empty
    - Expected Output: returns
- Test case 2: set has elements
    - Expected Output: all the elements are deleted and memory is freed

Void destroySet (Set* set)

Requires: nothing

Effects:

- If set is empty (set->head=NULL) does nothing
- Else clears all the elements and deletes the set (frees it from memory)

Test Suite:

- Test Case1: set is empty
    - Expected Output: returns
- Test case 2: set has elements

- Expected Output: all the elements are deleted and memory is freed

charmap.c

CharMap* initializeCharMap()

requires: nothing

effects: returns a charmap such that the keys are 0 to 25 and each key maps to an empty set

Test Suite:

- Test Case 1: Call function
  - Expected output: creates empty charmap

int idxOfKey(char key)

Requires: nothing

Effects:

- Transforms each character into an integer between 0 and 25
- If the character is not between ‘a’ and ‘z’ or ‘A’ and ‘Z’ returns -1

Test Suite:

- Test Case1: provide a character that is not between ‘a’ and ‘z’ or ‘A’ and ‘Z’
  - Expected Output: -1
- Test Case2: character key is between ‘a’ and ‘z’ or ‘A’ and ‘Z’
  - Expected Output: returns an int between 0 and 25 that represents the index of the character

void addToCharMap(CharMap* charMap, char key, const char* value)

Requires: nothing

Effects: adds element to charMap such that the the key of the element is the first letter of the value

Test Suite:

- Test Case1: if charmap is null
  - Expected Output: a new charmap is initialized and the value is added with its key
- Test Case 2: after initialization charmap is still null
  - Expected Output: memory allocation failed, function prints “failed to allocate memory
- Test Case 3: given a valid key and value
  - Expected Output: value is added to the charMap at the index of the provided key
- Test Case 4: given an invalid key
  - Expected Output: value is not added to the charMap

int isInCharMap (const CharMap* charMap, char key, const char* value)

requires: nothing

effects:

- returns true if value is in charmap
- false otherwise

Test Suite:

- Test Case1: value is not in the charMap
  - Expected Output: function returns 0

- TestCase2: value is in the charMap
    - Expected Output: function returns 1

void printCharMap(CharMap* charMap)

requires: nothing

effects:

- If charMap is null, prints “No elements found!”
- prints the values in the charmap such that each set’s words are separated by a tab space, and each five words are on their own line

Test Suite:

- Test Case1: empty charmap
    - Expected Output: prints “No elements found”
- TestCase2: valid charmap
    - Expected Output: prints elements in the charmap

void printKeyValue(CharMap* charMap)

requires: nothing

effects:

- If charMap is null, prints “No elements found!”
- prints the charmap in key : value format, such that the keys are between ‘a’ and ‘z’ and the values are sets containing words starting with the key

Test Suite:

- Test Case1: empty charmap
    - Expected Output: prints “No elements found”
- TestCase2: valid charmap

- Expected Output: prints elements in the charmap in key: value format

void destroyCharMap(CharMap* charMap)

requires: nothing

effects: deallocates the memory of the charmap

Test Suite:

- Test Case1: non-empty charmap
  - Expected Output: charmap is emptied and memory is deallocated
- Test Case2: empty charmap
  - Expected Output: destroy set function returns

Spellmaster.c:

char* Mode(char lastChar, charMap* charMap, char* mode)

Requires:

- lastChar to be the key of a set that contains values (is not NULL)
- charMap to be nonempty
- A string mode representing difficulty level : easy, medium, hard

Effects:

- if mode is “easy”, returns a word whose last letter maps to a set that maximizes the choices for the next player
- If mode is “hard”, returns a word whose last letter maps to a set that minimizes the choices for the next player
- If lastchar is ‘!’ then bot is starting the game, first choice will be made in the same manner as easy level

Test Suite:

- Test Case 1 : charMap has many words and the mode is easy
    - Expected Output : The bot will choose the set of words that has the most options for the user to choose from
- Test Case 2 : charMap has many words and the mode is hard
    - Expected Output : The bot will choose the set that has the least amount of words the user can choose from
- Test Case 3 : charMap has many words and the mode is medium :
    - Expected Output : Calls the medium function
- Test Case 4 : lastChar is '!'
    - Expected Output: Calls ModeHelper
- Test Case 5 : if charMap has only one word in the set that the last character maps to
    - Expected Output: Returns the word

char* ModeHelper(charMap* charMap)

Requires: charMap is nonEmpty

Effects:

- returns the word with the most available options for the next player
- Only runs 1 (if bot starts) or 0 (if player starts) times

Test Suite:

- Test Case 1: charMap has 1 word
    - Expected Output: returns the word
- Test Case 2: charMap has many words
    - Expected Output: returns the word whose last letter maps to the most words

Char * medium(char lastletter, CharMap *charmap)

Requires : charmap to be non empty, lastletter maps to a set that has words

Effects: Returns the words of medium frequency for the player to choose from

Test Suite:

- Test case 1: charmap has many words :
    - Expected Output: word of medium frequency will be returned
- Test case 2 : there's only one word in that set
    - Expected Output: returns that word
- Test case 3 : no words in that set
    - Expected Output: case handled by the driver
- Test case 4 : equal frequencies
    - Expected Output : word of medium frequency will be returned

Driver.c

Void playGame (char p1Name[] , char p2Name[] , CharMap * charMap)

Requires : non-empty charmap

Effects: runs the game while following its rules, function returns under below conditions:

1. A player chooses a word that is not in the list: other player wins
2. A player chooses a word that has been used before: other player wins
3. A player chooses a word that does not start with the letter that the previous word ended with: other player wins
4. A player chooses a word that leaves no options for the other player: the player who chose the word wins

Test Suite:

- Test case 1 : no more words to choose from
    - Expected Output: the player who chose the word should win since there are no words to choose from
- Test case 2 : choosing an invalid word
    - Expected Output: The player who chose that word will lose
- Test case 3 : choosing a word that has already been used
    - Expected Output : the player who chose that word will lose
- Test case 4 : choosing a word that doesn't start with the last letter of the previous word
    - Expected Output: the player who chose that word will lose

Void playWithBot(char p1Name[] , CharMap *charMap , char mode[])

Requires: non-empty charmap

Effects: runs the game with its rules but with the second player being the bot

Test Suite:

- Test case 1 : no more words to choose from
    - Expected Output: if it was the bot who ran out of words , it will lose, otherwise the player loses
- Test Case 2 : invalid word
    - Expected Output: if it was the bot who ran out of words , it will lose , otherwise the player loses
- Test case 3: choosing a word that has already been used
    - Expected Output: if it was the bot who chose that word , it will lose, else the player loses
- Test case 4 : choosing a word that doesn't start with the last letter of the previous word
    - Expected Output:: if it was the bot who chose that word, it will lose, else the player loses

void fileReading(CharMap *charMap)

Requires : initialized charMap

Effects:

- reads the file and and stores all the words according to charMap key: value pairs of characters and words starting with that character. Also prints “file opened successfully”
- if the file is null, prints “file failed to open” and returns

Test Suite:

- Test Case 1 : file is empty
    - Expected Output : failed to open file sentence will be printed
- Test Case 2 : file is non- empty
    - Expected Output : file will be opened successfully and will be read and stored in a charMap where each letter in it will have a set of words that start with that letter