***Note: make sure testGrid.dat file is in same folder as executable**

**Program 5 – Parts 1&2**

**Demetrius Johnson**

**CIS 350 – Summer II**

**Professor Thomas G. Steiner**

**August 19, 2021**

# 1. Problem Statement

Write a program to read in a grid of letters from a file, and then interactively allow a user to enter phrases until the user wants to quit. For each phrase the program must output whether or not the phrase is found and, if found, specifically where it is located. In addition, an output file will record what is displayed on the screen plus steps to find/not find the word or phrase.

# 2. Requirements

## 2.1. Assumptions

Input file values will be integers stating grid dimensions of character grid, followed by the grid
File FORMAT is correct
Check that all characters in the grid are characters
    else out put error and terminate
Check that values on line 1 is greater than 0 (grid dimensions)
    else output error message and terminate program
Uppercase letters in the file will be converted to lowercase – warning message written but program continues
Perform File Error Checks (file exists/can be opened..etc)
No letter may appear more than one time in any part of a solution.
    (i.e. phrase "go out" the letter "o" at position [3] [5] cannot be used for both "go" and "out"; there needs to be two "o"s.
File will contain only one character grid
The grid of letters for the program will be stored in a text file formatted as follows:
    Line 1: Two integers separated by a single blank space. These will represent the number of rows.
    Remaining lines: number of rows lines each containing number of columns characters.
The user input will be phrases of words, with a single space between each word. Ignore multiple space if they occur – <u>no message</u> – test case!
The program needs to edit the user entered data.
    Each phrase will be entered on a single line (assumption!).
User input will be converted to all lowercase letters
User will be informed of invalid word or phrase and be prompted to reenter word
Program terminates when user quits
A recursive, backtracking algorithm must be used to search the grid
    No letter may appear more than one time in any part of a solution
        i.e. phrase "go out" the letter "o" at position [3] [5] cannot be used for both "go" and "out"; there needs to be two "o"s.
    Hierarchy of direction choice for a search: right, down, left, up
    If the last letter in a word is at location (i, j), the first letter of the next word must be at one of locations (i, j+1), (i+1, j), (i, j-1), or (i-1, j).
    The direction chosen to find the first letter of a word is the same direction that must be used for all of the letters of the word.

If a phrase is not found in the grid, output should simply state that
If phrase found, program must find one occurrence of the phrase and indicate it

## 2.2. Specifications

MAIN FUNCTION

Display message “Welcome to the Backtracking Word Search Test Program” to user
Display message “Enter output file name: ” to user
Read and use the user entered output file name
If output file cannot be used
Display message “file <user output file name> cannot be opened – program terminated” to user
Display message “Welcome to the Backtracking Word Search Test Program” to output file
Display message “Testing Default Scenario” to user and output file
Create a 4x4 character table and test functionality
Call printTable function
Call a test phrase to search using backtrack algorithm
Call printPhraseTable
Display message “Enter file name for character table data: ” to user
Read user entered input data file name
Display message “File name for character table data: <input file name>” to output file
Display message “Testing File Data” to user and output file
Perform file validation
If cannot open
Display message “file <user input file name> cannot be opened or does not exist – program terminated”
If file exists but is empty
Display message “file <user input file name> contains no data – program terminated”
If number of rows is invalid (<= 0)
Display message “Error: Invalid number of rows: ” <row> “. Program terminated.”
If number of columns is invalid (<= 0)
Display message “Error: Invalid number of columns: ” <col> “. Program terminated.”
Read in all characters from the input file into a 2D dynamic array While not end of file
Display message <row> “x”<col> “ 2D array (word search character table) created from user file…building table…” to user and output file

If a character from the table is invalid (non-alphabetical)
Display message “Error: Invalid character in table: ” <char> “in position ” <[i][j]> “. Program terminated.” to user and output file
If a character from the table is uppercase
Display message “Warning: Uppercase letter in table input: ” <char> “in position ” [i][j] “. Converted to lower case.” to user and output file
If EOF reached
Display message “User input table read successfully…program launch is complete: ready to search words. User table: ” To user and output
Call printTable
Display message “Enter a letter, word, or phrase to search (alphabetical characters and spaces only). Enter a single non alphabetical character to quit the program.” to user
While user input is not a single alphabetical character
Display message “Enter a word or phrase to search, or enter a single nonalphabetical character to quit: ”
Get user input and store it in a string
Display message “User search phrase: ” <user input string> to output file
Call string parse function to delete leading and additional white spaces
If string size is 1 and contains a nonalphabetical
Display message “A single nonalphabetical character <user input string> has been entered: quitting program…” to user and output and then terminate program
Otherwise check that user input has only spaces and alphabetical characters
While there are characters in the user input string to check
If a character is nonalphabetical
Display message “Error: Invalid word or phrase: ‘<user input string>’. Re-enter word/phrase: ” to user
If a character is an uppercase letter
Convert letter to lowercase
Call backtrackingWordSearch function for user input
If the phrase is not found in the grid
Display message “The user input word/phrase <user input string> was not found in the table.”

Otherwise the program has found an occurrence of the phrase
Display message “The user input word/phrase was found!”
Call outputPhraseCoordinates function
Call printPhraseFoundTable

BACKTRACKING WORD SEARCH FUNCTION (priority: right, down, left, up) – display all messages to output file

While there are words to search (recursive algorithm operating on each word), search the characters of all words to see if the word is in the table, each word is consecutive to the previous, if current word reaches a dead end, then go back to the end of the previous word and start there (use boolean array to mark a location when all directions have been searched so that we know to go back to yet the next word’s ending; if we reach the first word on backtracking, then the phrase is not in the table)

Display message “Start (0,0) looking for ‘<user input[0]>’
If user input[0] is not found at (0,0)
Display message “ – not found”
Move right of (i,j) for user input[x]
If user input[x] is not found at (i,j)
Display message “ – not found”
Move to the down(or left, up, depending on w of (0,0)
Execute the same above logic for each direction in the direction hierarchy

If all directions at a given [i][j] has been checked, then mark a boolean array noting that, or if a direction does contain the letter being searched but has already been navigated, another boolean array will mark it so we know it is already a traveled dead end.

STRING PARSE FUNCTION – deletes leading or additional white spaces

PRINT TABLE FUNCTION – simple loop to print the 2D array from the input file

PRINT PHRASE FOUND TABLE FUNCTION – prints 2D array, capitalizing all positions where the word/phrase was found

OUTPUT PHRASE COORDINATES FUNCTION – print the coordinates of the first and last letter of each word in the phrase to user and output file

CREATE TABLE FUNCTION – creates the 2D array from input file

3. **Decomposition Diagram** (Used to break program down into components visually. Can have as many components as needed. Defines functionality that will solve the problem – does NOT define a flow )

Main

- Input
  - User file name
    - File validation
  - File Data
    - File data edits
      - Format:
        - Rows, column
        - Letter grid
- Process
  - Create table
  - Perform backtracking word search
- Output
  - Welcome message
  - Input error messages
  - Print user input table
  - Print phrase found table
  - Print phrase word coordinates

- End message

## 4. Test Strategy

File Testing (exist, empty)
Valid data
Invalid data

## 5. Test Plan Version 1

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
|---|---|---|---|---|---|---|
| File Testing | 1 | File does not exist | | | | |
| File Testing | 2 | File exists but empty | | | | |
| Valid data | 3 | Valid data from input file | | | | |
| Invalid data | 4 | Invalid number of rows or columns from input file | | | | |
| Invalid data | 5 | Invalid character in input table | | | | |
| Invalid data | 6 | Invalid user input for phrase | | | | |
| Valid data | 7 | Quit program nonalphabetical entered by user | | | | |
| Valid data | 8 | Find a phrase, restart loop | | | | |
| Valid data | 9 | Do not find a phrase or word | | | | |
| | | | | | | |

## 6. Initial Algorithm

Data: Object Definitions

Class Backtracking
Data:
String* userStringPtr
Vector<bool> allDirectionsSearched

Actions:
parseString

printPhraseCoord

Class Table
Data:
Char** arrayTable2D //2D array

Actions:
buildTable
printTable
printPhraseTable
destroyTable

MAIN FUNCTION
Display message “Welcome to the Backtracking Word Search Test Program” to user
Display message “Enter output file name: ” to user
Read and use the user entered output file name
If output file cannot be used
Display message “file <user output file name> cannot be opened – program terminated” to user
Display message “Welcome to the Backtracking Word Search Test Program” to output file
Display message “Testing Default Scenario” to user and output file
Create a 4x4 character table and test functionality
Call printTable function
Display message “Testing File Data” to user and output file
Display message “Enter file name for character table data: ” to user
Read user entered input data file name
Display message “File name for character table data: <input file name>” to output file
Perform file validation
If cannot open

Display message “file <user input file name> cannot be opened or does not exist – program terminated”

If file exists but is empty

Display message “file <user input file name> contains no data – program terminated”

If number of rows is invalid (<= 0)

Display message “Error: Invalid number of rows: ” <row> “. Program terminated.”

If number of columns is invalid (<= 0)

Display message “Error: Invalid number of columns: ” <col> “. Program terminated.”

Read in all characters from the input file into a 2D dynamic array While not end of file

Display message <row> “x”<col> “ 2D array (word search character table) created from user file…building table…” to user and output file

If a character from the table is invalid (non-alphabetical)

Display message “Error: Invalid character in table: ” <char> “in position ” <[i][j]> “. Program terminated.” to user and output file

If a character from the table is uppercase

Display message “Warning: Uppercase letter in table input: ” <char> “in position ” [i][j] “. Converted to lower case.” to user and output file

If EOF reached

Display message “User input table read successfully…program launch is complete: ready to search words. User table: ” To user and output

Call printTable

Display message “Enter a letter, word, or phrase to search (alphabetical characters and spaces only). Enter a single non alphabetical character to quit the program.” to user

While user input is not a single alphabetical character

Display message “Enter a word or phrase to search, or enter a single nonalphabetical character to quit: ”

Get user input and store it in a string

Display message “User search phrase: ” <user input string> to output file

Call string parse function to delete leading and additional white spaces

If string size is 1 and contains a nonalphabetical

Display message “A single nonalphabetical character <user input string> has been entered: quitting program…” to user and output and then terminate program
Otherwise check that user input has only spaces and alphabetical characters
While there are characters in the user input string to check
If a character is nonalphabetical
Display message “Error: Invalid word or phrase: ‘<user input string>’. Re-enter word/phrase: ” to user
If a character is an uppercase letter
Convert letter to lowercase
Call backtrackingWordSearch function for user input
If the phrase is not found in the grid
Display message “The user input word/phrase <user input string> was not found in the table.”
Otherwise the program has found an occurrence of the phrase
Display message “The user input word/phrase was found!”
Call outputPhraseCoordinates function
Call printPhraseFoundTable

## 7. Test Plan Version 2

| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
|---|---|---|---|---|---|---|
| File Testing | 1 | File does not exist | Wrong file name | “file not found” | | |
| File Testing | 2 | File exists but empty | Empty file | “file is empty” | | |
| Valid data | 3 | Valid data from input file | Valid test file | “test table created…user input table created” | | |
| Invalid data | 4 | Invalid number of rows or columns from input file | Test file with incorrect number of rows and columns | “invalid row or column value for table” | | |
| Invalid | 5 | Invalid | Input $ into | “non | | |

| | | | | | | |
|---|---|---|---|---|---|---|
| data | | character in input table | table | alphabetical char in table…” | | |
| Invalid data | 6 | Invalid user input for phrase | Input test$ | “re-enter a phrase, invalid char found” | | |
| Valid data | 7 | Quit program nonalphabetical entered by user | Enter ` | “` entered; program quitting” | | |
| Valid data | 8 | Find a phrase, restart loop | Search “test phrase” | Phrase found; output coordinates; output table | | |
| Valid data | 9 | Do not find a phrase or word | “test phrase not here” | Phrase not found | | |
| | | | | | | |

Part 1 ends here!!!!!!

## 8. Code

Copy and paste your code here. MAKE SURE TO COMMENT YOUR CODE!

### Main function:

```
// This file contains the 'main' function. Program execution begins and ends there.
//Author: Demetrius E Johnson
//Purpose: create a program that uses the backtracking algorithm to perform a word search
on a character grid
//Date Created: 8/12/21
//Date Modified: 8/19/21



#include "Table.h"
#include <iostream>
#include <sstream>
#include<string>
#include <fstream>
using namespace std;


void searchSequence(string search, Table& grid, ofstream& of);

int main()
{
	string userInputFile;
	string userOutputFile;
```

```
	string testInputFile = "testGrid.dat";

	cout << "Welcome to the Backtracking Word Search Test Program\n";
	cout << "Enter output file name: ";
	cin >> userOutputFile;
	ofstream outputFile(userOutputFile);

	//output file not opened sucessfully case:
	if (!outputFile.good()) {
		cout << "file " << userOutputFile << " cannot be opened – program
terminated...\n";
		system("pause");
		return 1;
	}

	outputFile << "Welcome to the Backtracking Word Search Test Program\n";
	outputFile << "Output file: " << userOutputFile << endl;

	//now build 4x4 test table using input test file "testGrid.dat" and test
backtracking program functionality
	Table cGrid;
	char** testTable = cGrid.getTableReference();
	ifstream testInput(testInputFile);

	//ensure test files open:
	if (!testInput.is_open()) {

		cout << "ERROR: The test input file" << testInputFile << " could not be
opened or found in proper system files location successfully...Program Terminated...\n";
		outputFile << "ERROR: The test input file" << testInputFile << " could not
be opened or found in proper system files location successfully...Program
Terminated...\n";
		system("pause");
		return 1;
	}

	//if return is true, then table was built successfully; otherwise return is false;
exit program --> return 1 for program terminated with errors
	if (cGrid.buildTable(testInput, outputFile)) {

		cout << "Test character table loaded successfully...Test table:\n\n";
		outputFile << "Test character table loaded successfully...Test table:\n\n";
	}
	else {
		system("pause");
		return 1;
	}

	//now call functions that test print functionality and wordsearch functionality:
	cGrid.printTable(outputFile);

	//Call test phrases to search using backtrack algorithm

	cout << "\nTesting Default Scenario...searching for words/phrases:\n'test' and
'TEST', 'test test', 'Test TEST', and 'test negative'.\n";
	outputFile << "\nTesting Default Scenario...searching for words/phrases:\n'test'
and 'TEST', 'test test', 'Test TEST', and 'test negative'.\n";
	//run tests:
```

```
	searchSequence("test", cGrid, outputFile);
	searchSequence("TEST", cGrid, outputFile);
	searchSequence("test test", cGrid, outputFile);
	searchSequence("Test TEST", cGrid, outputFile);
	searchSequence("test negative", cGrid, outputFile);

	cout << "Default Test Scenario completed: success...\n\n";
	outputFile << "Default Test Scenario completed: success...\n\n";

	//now get user input file:

	cout << "Enter an input file name for character table: ";
	cin >> userInputFile;
	outputFile << "File name for user input character table: " << userInputFile <<
endl;
	ifstream inputFile(userInputFile);

	//input file not open successfully case:
	if (!inputFile.good()) {
		cout << "file " << userInputFile << " cannot be opened or does not exist -
program terminated...\n";
		outputFile << "file " << userInputFile << " cannot be opened or does not
exist - program terminated...\n";
		system("pause");
		return 1;
	}

	//empty input file case:
	while (inputFile.peek() == ' ' || inputFile.peek() == '\n')
{ inputFile.ignore(); } //ignore leading white spaces and newlines until we reach a char
or EOF
	if (inputFile.peek() == EOF) {

		cout << "file " << userInputFile << " contains no data - program
terminated...\n";
		outputFile << "file " << userInputFile << " contains no data - program
terminated...\n";
		system("pause");
		return 1;
	}

	//now build the character table/grid (2D array) from user input file:

	cGrid.destroyTable();
	//clear out the test table
	cout << "Input file openend successfully...attempting to build table...\n";
	outputFile << "Input file openend successfully...attempting to build table...\n";

	//if return is true, then table was built successfully; otherwise return is false;
exit program --> return 1 for program terminated with errors
	if (cGrid.buildTable(inputFile, outputFile)) {

		cout << "User input file " << cGrid.getNumRow() << "x" << cGrid.getNumCol()
<< " table loaded successfully...Program Launch Success - ready to search words.\nUser
table:\n\n";
		outputFile << "User input file " << cGrid.getNumRow() << "x" <<
cGrid.getNumCol() << " table loaded successfully...Program Launch Success - ready to
search words.\nUser table:\n\n";
```

```
		cGrid.printTable(outputFile);
		cout << endl;
		outputFile << endl;
	}
	else {
		system("pause");
		return 1;
	}
	char** inputCharTable = cGrid.getTableReference();	//get reference to the new
table
	int numberofRows = cGrid.getNumRow();				//store row
dimension from table
	int numberofCols = cGrid.getNumCol();				//store column
dimension from table

	string userSearchInput = " ";

	//ensure cin is clear so that we can use getline function with no issues
	cin.seekg(0, ios::end);
	cin.clear();

	//do the following loop until user inputs a single character that is
nonalphabetical:
	do {
		//start search prompts:
		cout << "Enter a word or phrase to search, or enter a single
nonalphabetical character to quit:\n";
		cout << "Search word or phrase: ";
		getline(cin, userSearchInput);
			//cannot simply use cin; need to use getline so that input doesn't
stop at ws char and the entire input from user is acquired
		outputFile << "Search word or phrase: " << userSearchInput << endl;
//output searched word/phrase to output file so we know the word being searched for
		cGrid.inputSearchVal(userSearchInput);
		//add word to grid search value
		userSearchInput = cGrid.getSearchVal();
		//now get the search key that is fully corrected for addtional ws char and
capital letters changed to lowercase
		if (userSearchInput.size() == 1 && !isalpha(userSearchInput[0]) ||
userSearchInput.size() == 0) {	//check case where user wants to quit program (if size
== 0, user has entered only ws character(s))
			break;
		}
		while (cGrid.is_validWord() == false) {
		//while search value is invalid, loop and ask for valid input

			cout << "ERROR: Invalid word or phrase: '" << userSearchInput <<
"'.\nRe-eneter word/phrase: ";
			outputFile << "ERROR: Invalid word or phrase: '" << userSearchInput
<< "'.\nRe-eneter word/phrase: ";
			getline(cin, userSearchInput);
			cGrid.inputSearchVal(userSearchInput);
				//add word to grid
			userSearchInput = cGrid.getSearchVal();
				//now get search key that is fully corrected for
additional ws char and capital letters changed to lowercase
```

```
                    if (userSearchInput.size() == 1 && !isalpha(userSearchInput[0] ||
userSearchInput.size() == 0)) { break; } //make sure user can still enter a single
nonalpha char (or ws only) to exit program
                }
                cout << "Search word/phrase is valid.\nNumber of words in phrase is: " <<
cGrid.getNumWordsInSearchPhrase() << endl;
                outputFile << "Search word/phrase is valid. Number of words in phrase is: "
<< cGrid.getNumWordsInSearchPhrase() << endl;

                //now search for valid word/phrase and see if it is in the table:
                if (cGrid.wordSearch(outputFile)) {

                        //if word is found, the two below functions need to execute to
output coordinates of each word and the capitalized words of phrase table
                        cGrid.printPhraseCoord(outputFile);
                        cGrid.printPhraseFoundTable(outputFile);
                }
                //re-print table for next word to search:
                cout << endl;
                outputFile << endl;
                cGrid.printTable(outputFile);
                cout << endl;
                outputFile << endl;



        } while (userSearchInput.size() != 1 || isalpha(userSearchInput[0])); //both need
to be false in order for loop to break; false OR false == false

        //while loop above exited; user has entered a single non alphabetical character:
        cout << "\n\nA single nonalphabetical character '" << userSearchInput << "' has
been entered: quitting program...\n";
        outputFile << "\n\nA single nonalphabetical character " << userSearchInput << "has
been entered: quitting program...\n";

        cout << "\n~Thank you for running the Word Search Backtracking Program written by
Demetrius Johnson!\n";
        outputFile << "\n~Thank you for running the Word Search Backtracking Program
written by Demetrius Johnson!\n";

        //close files
        testInput.close();
        outputFile.close();
        inputFile.close();

        system("pause");
        return 0;
}


//Author: Demetrius E Johnson
//Purpose: condense the initiation of a search, and if necessary print out of coordinates
and table of a found phrase/word
//Date Created: 8/19/21
//Date Modified: 8/19/21
void searchSequence(string search, Table& grid, ofstream& of) {

        grid.inputSearchVal(search);
```

```
	if (grid.wordSearch(of)) {

		grid.printPhraseCoord(of);
		grid.printPhraseFoundTable(of);
	}
}
```

# Table Class:

## TABLE.H:

```
#ifndef TABLE_H
#define TABLE_H
#include <iostream>
#include <sstream>
#include<string>
#include <fstream>
#include <vector>
using namespace std;
class Table
{
private:
	int numRow = 0;
	int numCol = 0;
	int numWordsInPhrase = 0;
	//use this bool table to determine if a given coordinate has already been used in
finding another word
	bool** usedCoordinates;
	//use this vector and associated functions to store the start and ending
coordinates of all words in a found phrase; each found word will take up four vector
elements (xy, xy)
	vector<int> foundCoord;
	void addCoord(int x, int y);
	void deleteLetterCoord(void);
	//I could have used a boolean array for all my bool variables and simply did a
mapping but I figured it looks easier this way
	bool validWord = false;
	bool userPhraseFound = false;
	bool lastWordFound_right = false;
	bool lastWordFound_down = false;
	bool lastWordFound_left = false;
	bool lastWordFound_up = false;
	//use the below two bool variables to control when a coordinate should be deleted
from coordinate vector
	bool addedFirstLetter = false;
	bool addedLastLetter = false;
	//user the char** arrays to store character tables
	char** arrayTable_2D;
	char** arrayTable_2D_copy;
	//use below strings to store the user search phrase: one with spaces, and one
without
	string userSearchVal;
	string userSearchVal_noSpaces;
```

```
	//this private function will only be used by the inputSearchVal function to set the number of words found in the search phrase or get a given word from the phrase
	void setNumWordsInSearchPhrase(string searchKey);
	//these private functions used to check and set if all directions for a given element in the table have been searched
	void setAllDir_searched(bool val);
	//these private functions used to check a given direction (right, down, left, up)
	bool search_allDir(int& x, int& y, int& z, int& wordCount, ofstream& of);
	bool search_right(int& x, int& y, int& z, int& wordCount, ofstream& of);
	bool search_down(int& x, int& y, int& z, int& wordCount, ofstream& of);
	bool search_left(int& x, int& y, int& z, int& wordCount, ofstream& of);
	bool search_up(int& x, int& y, int& z, int& wordCount, ofstream& of);
	//use these functions to maintain the last word found:
	void foundRight(void);
	void foundDown(void);
	void foundLeft(void);
	void foundUp(void);

public:

	//primary functions/algorithms/set functions:
	bool buildTable(ifstream& inputFile, ofstream& outputFile); //return true == table build successful; return false == table not built succesfully
	bool wordSearch(ofstream& of);
	void inputSearchVal(string val);
	bool is_validWord(void);
	//print functions:
	void printTable(ofstream& of);
	void printPhraseFoundTable(ofstream& of);
	void printPhraseCoord(ofstream& of);
	//get functions:
	string getSearchVal(void);
	string getSearchVal_noSpaces(void);
	int getNumRow(void);
	int getNumCol(void);
	int getNumWordsInSearchPhrase(void);
	char** getTableReference(void);
	string getWordFromPhrase(int wordNum);
	//reset function:
	void resetPhraseTable(void);
	void resetBoolTable(void);
	//destructors
	void destroyTable(void);
	~Table();

};
#endif
```

## TABLE.CPP:

```
#include "Table.h"

//Author: Demetrius E Johnson
//Purpose: add coordinates to coordiante vector in order to keep track of the first and last letter of each found word
//Date Created: 8/19/21
//Date Modified: 8/19/21
void Table::addCoord(int x, int y) {
```

```
	foundCoord.push_back(x);
	foundCoord.push_back(y);
}
//Author: Demetrius E Johnson
//Purpose: delete the coordinate of a word (first and last letter) in the event that it
is not longer apart of the solution set for found words in a phrase
//Date Created: 8/19/21
//Date Modified: 8/19/21
void Table::deleteLetterCoord(void) {

	foundCoord.pop_back();
	foundCoord.pop_back();
}
//Author: Demetrius E Johnson
//Purpose: build several 2d arrays that are associated with the table input from the user
input file
//Date Created: 8/12/21
//Date Modified: 8/19/21
bool Table::buildTable(ifstream& inputFile, ofstream& outputFile) {

	//first, check to ensure a table is not already present; call destroy function to
clear the table if necessary:
	destroyTable();

	//now build the character table/grid (2D array)

	stringstream ss;			//use this to input an integer stored in a
string into an int
	string lineParse;			//use this to parse each line from the input
file




	cout << endl << endl;
	outputFile << endl << endl;

	while (inputFile.peek() == ' ' || inputFile.peek() == '\n')
{ inputFile.ignore(); } //if necessary: ignore leading white spaces or newlines to make
input file more flexible by this program

	//acquire size of the table; the first 2 characters are numbers in the data file
giving dimensions of the table; we wills store dimensions in numRow and numCol:

	getline(inputFile, lineParse);	//store current line from input file into
lineParse
	ss << lineParse;			//output to stream
	ss >> numRow >> numCol;			//convert char/string to integer values
//side note: istream::operator>> only extracts characters; it does not also discard them;
use cin.ignore() function to clear buffer if necessary
	ss.clear();				//clear bad bits set for this
stream in case of any bad bits set from outputting from the stream into a variable in the
previous line
								//inputFile should now be
pointing to the next line of the input file (first row of characters in the table)
```

```
	//next set of if statements will check the value of the numRow and numCol aquired
from the input table to see if they are valid:
	if (numRow <= 0) { //check rows

		cout << "ERROR: number of rows: " << numRow << " is less than or equal to
zero.\n"
			<< "Program terminated.\n";
		outputFile << "ERROR: number of rows: " << numRow << " is less than or
equal to zero.\n"
			<< "Program terminated.\n";
		return false;
	}
	else if (numCol <= 0) {//check columns

		cout << "ERROR: number of columns: " << numCol << " is less than or equal
to zero.\n"
			<< "Program terminated.\n";
		outputFile << "ERROR: number of columns: " << numCol << " is less than or
equal to zero.\n"
			<< "Program terminated.\n";
		return false;
	}
	else { //otherwise: number of rows and columns is valid; we can proceed with
attempting to build input character table

		cout << "Number of rows(" << numRow << ") and number of columns(" << numCol
<< ")" << " is valid.\n";
		outputFile << "Number of rows(" << numRow << ") and number of columns(" <<
numCol << ")" << " is valid.\n";

	}

	//we reach this point if rows and column values from the user input file are
valid; now we must allocate the 2D array of the appropriate size:
	arrayTable_2D = new char* [numRow];
				//double char ptr that points to an array of single
char ptrs
	for (int i = 0; i < numRow; i++) { arrayTable_2D[i] = new char[numCol]; }
	//allocate each char pointer to point to an array of char

	//create a copy of the table for later use in the printPhraseFoundTable function:
	arrayTable_2D_copy = new char* [numRow];
	for (int i = 0; i < numRow; i++) { arrayTable_2D_copy[i] = new char[numCol]; }

	//create bool table so we know which char in the table have been used when
searching a phrase:
	usedCoordinates = new bool* [numRow];
	for (int i = 0; i < numRow; i++) { usedCoordinates[i] = new bool[numCol]; }


	//below loop: load all characters from input file character grid into the 2D
array:

	for (int i = 0; i < numRow; i++) {

		getline(inputFile, lineParse);
	//get the current line of characters from the table (getline function stops at the
newline char and discards it when it reads it)
```

```
		for (int j = 0; j < numCol; j++) {

			arrayTable_2D[i][j] = lineParse[j];
	//each line is of length j; there are i rows of length j (columns); so we need to acquire all elements from the string (characters)
			//before leaving loop, check to ensure that the character just added is alphabetical and that it is lowercase:
			if (!isalpha(arrayTable_2D[i][j])) {

				cout << "ERROR: Invalid character in table input: '" << arrayTable_2D[i][j] << "' in position [" << i << "][" << j << "]. Program Terminated...\n";
				outputFile << "ERROR: Invalid character in table input: '" << arrayTable_2D[i][j] << "' in position [" << i << "][" << j << "]. Program Terminated...\n";
				return false;
			}
			//if element added is alphabetical, convert it to lowercase if necessary:
			if (isupper(arrayTable_2D[i][j])) {

				cout << "WARNING: Uppercase letter in table input: '" << arrayTable_2D[i][j] << "' in position [" << i << "][" << j << "]. Converted to lowercase...\n";
				outputFile << "WARNING: Uppercase letter in table input: '" << arrayTable_2D[i][j] << "' in position [" << i << "][" << j << "]. Converted to lowercase...\n";
				arrayTable_2D[i][j] = tolower(arrayTable_2D[i][j]);
			}
			arrayTable_2D_copy[i][j] = arrayTable_2D[i][j]; //add char to copy table
		}
	}
	//2D array should now be fully loaded


	return true;
}
//Author: Demetrius E Johnson
//Purpose: deallocate all dynamic arrays so as to reset them for use in building another table
//Date Created: 8/12/21
//Date Modified: 8/19/21
void Table::destroyTable(void) {

	//first check to ensure the current table instance (object) does not already have a table loaded
	if (arrayTable_2D != nullptr) {

		//deallocate dynamic 2d array:
		for (int i = 0; i < numRow; i++) { delete[] arrayTable_2D[i]; }
//deallocate char* dynamic arrays
		delete[] arrayTable_2D;
			//deallocate char** dynamic array

		//repeat above process to deallocate the copy version and bool version:

		//deallocate dynamic 2d array:
```

```
			for (int i = 0; i < numRow; i++) { delete[] arrayTable_2D_copy[i]; }
		//deallocate char* dynamic arrays
			delete[] arrayTable_2D_copy;
					//deallocate char** dynamic array

			//deallocate dynamic 2d array:
			for (int i = 0; i < numRow; i++) { delete[] usedCoordinates[i]; }
		//deallocate char* dynamic arrays
			delete[] usedCoordinates;

			//reset numRow and numCol and set array to null ptr; this allow the object
to effectively be reinitialzed to its start state so that it can be reused for a new
table
			numRow = 0;
			numCol = 0;
			arrayTable_2D = nullptr;
			arrayTable_2D_copy = nullptr;
			usedCoordinates = nullptr;
		}
}
//Author: Demetrius E Johnson
//Purpose: receive and properly parse user input for search phrase so that we can apply
algorithms that need clean, single spaced input
//Date Created: 8/12/21
//Date Modified: 8/19/21
void Table::inputSearchVal(string val) {

		//default: word is assumed to be a valid input:
		validWord = true;

		//clear the strings so that whenever the function is called, no old data will
affect the new value to input
		userSearchVal.clear();
		userSearchVal_noSpaces.clear();

		//set user search val to original user input, ignoring leading spaces and extra
spaces between words:
		for (int i = 0; i < val.size(); i++) {

			//all leading spaces will be skipped until we hit a non ws char:
			if (val[i] != ' ') {

				userSearchVal.push_back(tolower(val[i]));//add char to userSearchVal,
ensuring it is lowercase if it is alphabetical

				//if a whitespace is the next char (and not the last char) add it to
the searchVal; additional spaces will be skipped when for loop starts again:
				if (((i + 1) < (val.size())) && ((i + 1) != (val.size() - 1))) {

					if (val[i + 1] == ' ') { userSearchVal.push_back(val[i +
1]); }
				}
			}
		}
		if (userSearchVal.size() != 0 && userSearchVal[userSearchVal.size() - 1] == ' ')
{ userSearchVal.pop_back(); } //there may be a single trailing ws char at the end of the
final userSearchVal
```

```
	setNumWordsInSearchPhrase(userSearchVal);
                                                                        //now call
function to store number of words in the phrase -- this will be useful for the search
algorithm

	//now, set no-space version of the user input to be used in backtrack algorithm
for the word search, while also checking if all values are alphabetical and lowercase:
	for (int i = 0; i < userSearchVal.size(); i++) {

		//check to see if the value is alphabetical:
		if (isalpha(userSearchVal[i])) {

			//add lowercase value to no-space version of the search value
			userSearchVal_noSpaces.push_back(tolower(userSearchVal[i]));

		}
		else if (userSearchVal[i] != ' ') {//otherwise, value is not alphabetical
or a space; we can simply return and exit function - no need to continue

			validWord = false; //set this to false; now calling the function
that returns this value will properly tell if the newly added search phrase/word is valid
			return;
		}
	}
}

//Author: Demetrius E Johnson
//Purpose: use this in order to calculate how many words were entered by the user's
phrase - this will be used for various functions
//Date Created: 8/12/21
//Date Modified: 8/19/21
void Table::setNumWordsInSearchPhrase(string searchKey) {

	numWordsInPhrase = 0;	//reset words in phrase counter
	stringstream ss;
	string currentWord;
	ss << searchKey;		//output search key string into ss to be parsed
	while (ss.good()) {

		ss >> currentWord;		//input a word into currentWord -- ss will stop
inputting characters into currentWord when a ws is reached)
		numWordsInPhrase++;		//word added; increment counter
	}
}
//Author: Demetrius E Johnson
//Purpose: set all directions at a given location in the table to true or false - this
sets all or resets all directions that have been searched at a given position in the
table
//Date Created: 8/19/21
//Date Modified: 8/19/21
void Table::setAllDir_searched(bool val) {

	//reset all values to false:
	lastWordFound_right = val;
	lastWordFound_down = val;
	lastWordFound_left = val;
	lastWordFound_up = val;
}
```

```
//Author: Demetrius E Johnson
//Purpose: search all directions relative to a given position in the table - uses
recursion - this is a major part to the backtracking algorithm functioning of this
program
//Date Created: 8/17/21
//Date Modified: 8/19/21
bool Table::search_allDir(int& x, int& y, int& z, int& wordCount, ofstream& of) {

	//use these ints in order to track original values for the current function call
	int x_original = x;
	int y_original = y;
	int z_original = z;
	int wordCount_original = wordCount;
	int vectorSize_original = foundCoord.size(); // track original size of vector so
that when we need to backtrack, all added cooridantes are deleted

	//search right:
	bool wordFound = search_right(x, y, z, wordCount, of);

	if (wordFound == true && z == userSearchVal.size()) { //initial base case: last
word in phrase was found right

		return true;
	}
	if (wordFound == true) { wordFound = search_allDir(x, y, z, wordCount, of); }
//recursively restart search since word was found; need to search for next word
	if (wordFound == true && z == userSearchVal.size()) { //base case: last word in
phrase was found right

		return true;
	}

	//otherwise search down:

	//we need to reset values relative to their original position for when the
function was called, and delete the last word coordinates found since it was a dead end
	x = x_original;
	y = y_original;
	z = z_original;
	wordCount = wordCount_original;
	//need to delete the last coordinates added since at this point it means the last
added word was not a valid path and we had to backtrack
	if (vectorSize_original == foundCoord.size() - 4) {//word was added - delete both
cooridantes
		deleteLetterCoord();
		deleteLetterCoord();
	}
	else if (vectorSize_original == foundCoord.size() - 2) { deleteLetterCoord(); }
//only the first letter of a word was added - delete only a single coordinate
	vectorSize_original = foundCoord.size(); //set new original size

	wordFound = search_down(x, y, z, wordCount, of);

	if (wordFound == true && z == userSearchVal.size()) { //base case: last word in
phrase was found down

		return true;
	}
```

```
	if (wordFound == true) { wordFound = search_allDir(x, y, z, wordCount, of); }
//recursively restart search since word was found; need to search for next word
	if (wordFound == true && z == userSearchVal.size()) { //base case: last word in
phrase was found right

		return true;
	}

	//otherwise search left:

	//we need to reset values relative to their original position for when the
function was called
	x = x_original;
	y = y_original;
	z = z_original;
	wordCount = wordCount_original;
	//need to delete the last coordinates added since at this point it means the last
added word was not a valid path and we had to backtrack
	if (vectorSize_original == foundCoord.size() - 4) {//word was added - delete both
cooridantes
		deleteLetterCoord();
		deleteLetterCoord();
	}
	else if (vectorSize_original == foundCoord.size() - 2) { deleteLetterCoord(); }
//only the first letter of a word was added - delete only a single coordinate
	vectorSize_original = foundCoord.size(); //set new original size

	wordFound = search_left(x, y, z, wordCount, of);

	if (wordFound == true && z == userSearchVal.size()) { //base case: last word in
phrase was found left

		return true;
	}
	if (wordFound == true) { wordFound = search_allDir(x, y, z, wordCount, of); }
//recursively restart search since word was found; need to search for next word
	if (wordFound == true && z == userSearchVal.size()) { //base case: last word in
phrase was found right

		return true;
	}

	//otherwise search up:

	//we need to reset values relative to their original position for when the
function was called
	x = x_original;
	y = y_original;
	z = z_original;
	wordCount = wordCount_original;
	//need to delete the last coordinates added since at this point it means the last
added word was not a valid path and we had to backtrack
	if (vectorSize_original == foundCoord.size() - 4) {//word was added - delete both
cooridantes
		deleteLetterCoord();
		deleteLetterCoord();
	}
```

```
	else if (vectorSize_original == foundCoord.size() - 2) { deleteLetterCoord(); }
//only the first letter of a word was added - delete only a single coordinate
	vectorSize_original = foundCoord.size(); //set new original size

	wordFound = search_up(x, y, z, wordCount, of);

	if (wordFound == true && z == userSearchVal.size()) { //base case: last word in
phrase was found up

		return true;
	}
	if (wordFound == true) { wordFound = search_allDir(x, y, z, wordCount, of); }
//recursively restart search since word was found; need to search for next word
	if (wordFound == true && z == userSearchVal.size()) { //final base case: last word
in phrase was found right

		return true;
	}

	//otherwise word or phrase was not found:
//need to delete the last coordinates added since at this point it means the last added
word was not a valid path and we had to backtrack
	if (vectorSize_original == foundCoord.size() - 4) {//word was added - delete both
cooridantes
		deleteLetterCoord();
		deleteLetterCoord();
	}
	else if (vectorSize_original == foundCoord.size() - 2) { deleteLetterCoord(); }
//only the first letter of a word was added - delete only a single coordinate
	vectorSize_original = foundCoord.size(); //set new original size

	cout << "search: all directions searched at (" << x << "," << y << ").\n";
	return false; // base case: dead end; phrase not found from the current (passed
in) starting position in the table - return false
}

//Author: Demetrius E Johnson
//Purpose: For the below 4 functions: search a given location relative to the current
position we are looking at in the table - this is a major part to the backtracking
algorithm functioning of this program
//Date Created: 8/17/21
//Date Modified: 8/19/21
bool Table::search_right(int& x, int& y, int& z, int& wordCount, ofstream& of) {

	string currentWord;                                          //use to print
current word if found

	//use the below variables to maintain original positions in the event that word is
not found:
	int x_original = x;
	int y_original = y;
	int originalPos = z;

	//reset these variables so we correctly represent that no coordinates of the word
have been added to coordinate vector
	addedFirstLetter = false;
	addedLastLetter = false;
```

```
	//z refers to position in the user phrase/string we are searching, current word refers to current word in the phrase we are searching
	// x and y are set according to table coordinates relative to the current element we are checking in the table

	cout << "search: currPos<right>: (" << x << "," << y + 1 << ").\n";
	of << "search: currPos<right>: (" << x << "," << y + 1 << ").\n";

	while (z < userSearchVal.size() && x < numRow && y + 1 < numCol && arrayTable_2D[x][y + 1] == userSearchVal[z]) {

		cout << "search: right: '" << userSearchVal[z] << "' found at (" << x  << "," << y + 1 << ").\n";
		of << "search: right: '" << userSearchVal[z] << "' found at (" << x  << "," << y + 1 << ").\n";

		//use the below if statements to add coordinate of start letter and last letter of each word:
		//add first letter:
		if (z - 1 >= 0 && userSearchVal[z - 1] == ' ') {

			addCoord(x, y + 1);
			addedFirstLetter = true;
		}
		//add last letter:
		if (z + 1 < userSearchVal.size() && userSearchVal[z + 1] == ' ') {

			addCoord(x, y + 1);
			addedLastLetter = true;
		}
		else if (z + 1 == userSearchVal.size()) {
			addCoord(x, y + 1);
			addedLastLetter = true;

		} //case: we have reached last letter of the entire phrase

		//use below statement to set true in the bool table that a given coordinate is now taken if it is not already:
		//case: coordinate already used; exit function
		if (usedCoordinates[x][y + 1] == true) {

			//maintain original positions since word was not found:
			cout << "search: overlap...backtracking: (" << x_original << "," << y_original << ").\n";
			of << "search: overlap...backtracking: (" << x_original << "," << y_original << ").\n";
			x = x_original;
			y = y_original;
			z = originalPos;

			return false;
		}
		//otherwise: coordinate now used in the search, set it to true so there are no potential overlap or loops later in search
		else { usedCoordinates[x][y + 1] = true; }
```

```
			y++;	//move right in the table (change column, maintain current row) for
the next search
			z++;	//move to next char in search phrase to check against

			//case: we have reached the end of a word in the phrase:
			if (z < userSearchVal.size() && userSearchVal[z] == ' ') {

				currentWord = getWordFromPhrase(wordCount);
				cout << "search: <" << currentWord << "> found.\n";
				of << "search: <" << currentWord << "> found.\n";
				wordCount++;  //move to next word to output from the phrase
				z++;				//skip the space - move to first letter in next
word
				foundRight(); //set flags for which direction the last word was found
				return true;
			}
			//case: we have reached the end of the phrase - phrase has been found in
the table:
			if (z == userSearchVal.size()) {

				currentWord = getWordFromPhrase(wordCount);
				cout << "search: <" << currentWord << "> found.\n";
				of << "search: <" << currentWord << "> found.\n";
				foundRight(); //set flags for which direction the last word was
found
				return true;
			}
		}

		//maintain original positions since word was not found:
		cout << "search: ...backtracking: (" << x_original << "," << y_original << ").\n";
		of << "search: ...backtracking: (" << x_original << "," << y_original << ").\n";
		x = x_original;
		y = y_original;
		z = originalPos;

		//

		return false;
}
bool Table::search_down(int& x, int& y, int& z, int& wordCount, ofstream& of) {

		string currentWord;											//use to
print current word if found

		//use the below variables to maintain original positions in the event that word is
not found:
		int x_original = x;
		int y_original = y;
		int originalPos = z;

		//reset these variables so we correctly represent that no coordinates of the word
have been added to coordinate vector
		addedFirstLetter = false;
		addedLastLetter = false;
```

```
	//z refers to position in the user phrase/string we are searching, current word refers to current word in the phrase we are searching
	// x and y are set according to table coordinates relative to the current element we are checking in the table

	cout << "search: currPos<down>: (" << x + 1 << "," << y << ").\n";
	of << "search: currPos<down>: (" << x + 1 << "," << y << ").\n";

	while (z < userSearchVal.size() && x + 1 < numRow && y < numCol && arrayTable_2D[x + 1][y] == userSearchVal[z]) {

		cout << "search: down: '" << userSearchVal[z] << "' found at (" << x + 1 << "," << y << ").\n";
		of << "search: down: '" << userSearchVal[z] << "' found at (" << x + 1 << "," << y << ").\n";

		//use the below if statements to add coordinate of start letter and last letter of each word:
		//add first letter:
		if (z - 1 >= 0 && userSearchVal[z - 1] == ' ') {

			addCoord(x + 1, y);
			addedFirstLetter = true;
		}
		//add last letter:
		if (z + 1 < userSearchVal.size() && userSearchVal[z + 1] == ' ') {

			addCoord(x + 1, y);
			addedLastLetter = true;
		}
		else if (z + 1 == userSearchVal.size()) {

			addCoord(x + 1, y);
			addedLastLetter = true;
		} //case: we have reached last letter of the entire phrase

		//use below statement to set true in the bool table that a given coordinate is now taken if it is not already:
		//case: coordinate already used; exit function
		if (usedCoordinates[x + 1][y] == true) {

			//maintain original positions since word was not found:
			cout << "search: overlap...backtracking: (" << x_original << "," << y_original << ").\n";
			of << "search: overlap...backtracking: (" << x_original << "," << y_original << ").\n";
			x = x_original;
			y = y_original;
			z = originalPos;

			return false;
		}
		//otherwise: coordinate now used in the search, set it to true so there are no potential overlap or loops later in search
		else { usedCoordinates[x + 1][y] = true; }

		x++;	//move down in the table (change row, maintain current column) for the next search
```

```
			z++;	//move to next char in search phrase to check against

			//case: we have reached the end of a word in the phrase:
			if (z < userSearchVal.size() && userSearchVal[z] == ' ') {

				currentWord = getWordFromPhrase(wordCount);
				cout << "search: <" << currentWord << "> found.\n";
				of << "search: <" << currentWord << "> found.\n";
				wordCount++; //move to next word to output from the phrase
				z++;		//skip the space - move to first letter in next word
				foundDown(); //set flags for which direction the last word was found
				return true;
			}
			//case: we have reached the end of the phrase - phrase has been found in
the table:
			if (z == userSearchVal.size()) {

				currentWord = getWordFromPhrase(wordCount);
				cout << "search: <" << currentWord << "> found.\n";
				of << "search: <" << currentWord << "> found.\n";
				foundDown(); //set flags for which direction the last word was found
				return true;
			}
		}

		//maintain original positions since word was not found:
		cout << "search: ...backtracking: (" << x_original << "," << y_original << ").\n";
		of << "search: ...backtracking: (" << x_original << "," << y_original << ").\n";
		x = x_original;
		y = y_original;
		z = originalPos;

		return false;
}

bool Table::search_left(int& x, int& y, int& z, int& wordCount, ofstream& of) {

		string currentWord;							//use to print
current word if found

		//use the below variables to maintain original positions in the event that word is
not found:
		int x_original = x;
		int y_original = y;
		int originalPos = z;

		//reset these variables so we correctly represent that no coordinates of the word
have been added to coordinate vector
		addedFirstLetter = false;
		addedLastLetter = false;

		//z refers to position in the user phrase/string we are searching, current word
refers to current word in the phrase we are searching
		// x and y are set according to table coordinates relative to the current element
we are checking in the table

		cout << "search: currPos<left>: (" << x << "," << y - 1 << ").\n";
		of << "search: currPos<left>: (" << x << "," << y - 1 << ").\n";
```

```
	while (z < userSearchVal.size() && x < numRow && y - 1 < numCol && y - 1 >= 0 &&
arrayTable_2D[x][y - 1] == userSearchVal[z]) {

		cout << "search: left: '" << userSearchVal[z] << "' found at (" << x << ","
<< y - 1 << ").\n";
		of << "search: left: '" << userSearchVal[z] << "' found at (" << x << ","
<< y - 1 << ").\n";

		//use the below if statements to add coordinate of start letter and last
letter of each word:
		//add first letter:
		if (z - 1 >= 0 && userSearchVal[z - 1] == ' ') {

			addCoord(x, y - 1);
			addedFirstLetter = true;
		}
		//add last letter:
		if (z + 1 < userSearchVal.size() && userSearchVal[z + 1] == ' ') {

			addCoord(x, y - 1);
			addedLastLetter = true;
		}
		else if (z + 1 == userSearchVal.size()) {

			addCoord(x, y - 1);
			addedLastLetter = true;
		} //case: we have reached last letter of the entire phrase

		//use below statement to set true in the bool table that a given coordinate
is now taken if it is not already:
		//case: coordinate already used; exit function
		if (usedCoordinates[x][y - 1] == true) {

			//maintain original positions since word was not found:
			cout << "search: overlap...backtracking: (" << x_original << "," <<
y_original << ").\n";
			of << "search: overlap...backtracking: (" << x_original << "," <<
y_original << ").\n";
			x = x_original;
			y = y_original;
			z = originalPos;

			return false;
		}
		//otherwise: coordinate now used in the search, set it to true so there are
no potential overlap or loops later in search
		else { usedCoordinates[x][y - 1] = true; }

		y--;	//move left in the table (change column, maintain current row)
		z++;	//move to next char in search phrase to check against

		//case: we have reached the end of a word in the phrase:
		if (z < userSearchVal.size() && userSearchVal[z] == ' ') {

			currentWord = getWordFromPhrase(wordCount);
			cout << "search: <" << currentWord << "> found.\n";
			of << "search: <" << currentWord << "> found.\n";
```

```
                        wordCount++; //move to next word to output from the phrase
                        z++;           //skip the space - move to first letter in next word
                        foundLeft();//set flags for which direction the last word was found
                        return true;
                }
                //case: we have reached the end of the phrase - phrase has been found in
the table:
                if (z == userSearchVal.size()) {

                        currentWord = getWordFromPhrase(wordCount);
                        cout << "search: <" << currentWord << "> found.\n";
                        of << "search: <" << currentWord << "> found.\n";
                        foundLeft();//set flags for which direction the last word was found
                        return true;
                }
        }

        //maintain original positions since word was not found:
        cout << "search: ...backtracking: (" << x_original << "," << y_original << ").\n";
        of << "search: ...backtracking: (" << x_original << "," << y_original << ").\n";
        x = x_original;
        y = y_original;
        z = originalPos;

        return false;
}
bool Table::search_up(int& x, int& y, int& z, int& wordCount, ofstream& of) {

        string currentWord;                                                     //use
to print current word if found

                //use the below variables to maintain original positions in the event that
word is not found:
        int x_original = x;
        int y_original = y;
        int originalPos = z;

        //reset these variables so we correctly represent that no coordinates of the word
have been added to coordinate vector
        addedFirstLetter = false;
        addedLastLetter = false;


        //z refers to position in the user phrase/string we are searching, current word
refers to current word in the phrase we are searching
        // x and y are set according to table coordinates relative to the current element
we are checking in the table

        cout << "search: currPos<up>: (" << x - 1 << "," << y << ").\n";
        of << "search: currPos<up>: (" << x - 1 << "," << y << ").\n";

        while (z < userSearchVal.size() && x - 1 < numRow && x - 1 >= 0 && y < numCol &&
arrayTable_2D[x - 1][y] == userSearchVal[z]) {

                cout << "search: up: '" << userSearchVal[z] << "' found at (" << x - 1  <<
"," << y  << ").\n";
                of << "search: up: '" << userSearchVal[z] << "' found at (" << x - 1  <<
"," << y << ").\n";
```

```
            //use the below if statements to add coordinate of start letter and last
letter of each word:
            //add first letter:
            if (z - 1 >= 0 && userSearchVal[z - 1] == ' ') {

                  addCoord(x - 1, y);
                  addedFirstLetter = true;
            }
            //add last letter:
            if (z + 1 < userSearchVal.size() && userSearchVal[z + 1] == ' ') {

                  addCoord(x - 1, y);
                  addedLastLetter = true;
            }
            else if (z + 1 == userSearchVal.size()) {

                  addCoord(x - 1, y);
                  addedLastLetter = true;
            } //case: we have reached last letter of the entire phrase

            //use below statement to set true in the bool table that a given coordinate
is now taken if it is not already:
            //case: coordinate already used; exit function
            if (usedCoordinates[x - 1][y] == true) {

                  //maintain original positions since word was not found:
                  cout << "search: overlap...backtracking: (" << x_original << "," <<
y_original << ").\n";
                  of << "search: overlap...backtracking: (" << x_original << "," <<
y_original << ").\n";
                  x = x_original;
                  y = y_original;
                  z = originalPos;

                  return false;
            }
            //otherwise: coordinate now used in the search, set it to true so there are
no potential overlap or loops later in search
            else { usedCoordinates[x - 1][y] = true; }

            x--;   //move up in the table (change row, maintain current column)
            z++;   //move to next char in search phrase to check against

            //case: we have reached the end of a word in the phrase:
            if (z < userSearchVal.size() && userSearchVal[z] == ' ') {

                  currentWord = getWordFromPhrase(wordCount);
                  cout << "search: <" << currentWord << "> found.\n";
                  of << "search: <" << currentWord << "> found.\n";
                  wordCount++; //move to next word to output from the phrase
                  z++;           //skip the space - move to first letter in next word
                  foundUp(); //set flags for which direction the last word was found
                  return true;
            }
            //case: we have reached the end of the phrase - phrase has been found in
the table:
            if (z == userSearchVal.size()) {
```

```
                        currentWord = getWordFromPhrase(wordCount);
                        cout << "search: <" << currentWord << "> found.\n";
                        of << "search: <" << currentWord << "> found.\n";
                        foundUp(); //set flags for which direction the last word was found
                        return true;
                }
        }

        //maintain original positions since word was not found:
        cout << "search: ...backtracking: (" << x_original << "," << y_original << ").\n";
        of << "search: ...backtracking: (" << x_original << "," << y_original << ").\n";
        x = x_original;
        y = y_original;
        z = originalPos;

        return false;
}

//Author: Demetrius E Johnson
//Purpose: create a program that uses the backtracking algorithm to perform a word search
on a character grid
//Date Created: 8/18/21
//Date Modified: 8/18/21
void Table::foundRight(void) {

        lastWordFound_right = true;
        lastWordFound_down = false;
        lastWordFound_left = false;
        lastWordFound_up = false;

}
void Table::foundDown(void) {

        lastWordFound_right = false;
        lastWordFound_down = true;
        lastWordFound_left = false;
        lastWordFound_up = false;
}
void Table::foundLeft(void) {

        lastWordFound_right = false;
        lastWordFound_down = false;
        lastWordFound_left = true;
        lastWordFound_up = false;

}
void Table::foundUp(void) {

        lastWordFound_right = false;
        lastWordFound_down = false;
        lastWordFound_left = false;
        lastWordFound_up = true;
}

//below are simple return functions that return user phrase, numRow and numCol in table,
num words in search phrase, and reference to the 2d array table
string Table::getSearchVal(void) { return userSearchVal; }
```

```
string Table::getSearchVal_noSpaces(void) { return userSearchVal_noSpaces; }
int Table::getNumRow(void) { return numRow; }
int Table::getNumCol(void) { return numCol; }
int Table::getNumWordsInSearchPhrase(void) { return numWordsInPhrase; }
char** Table::getTableReference(void) { return arrayTable_2D; }

//Author: Demetrius E Johnson
//Purpose: returns a word from the phrase relative to the number passed in: for example,
if wordNum == 2, then the second word in the phrase will be returned (if possible)
//Date Created: 8/12/21
//Date Modified: 8/19/21
string Table::getWordFromPhrase(int wordNum) {

	if (wordNum > numWordsInPhrase || wordNum <= 0) { return ""; } //invalid word
number; return null string
	else {
		stringstream ss;
		string word;
		ss << userSearchVal; //output the user phrase to ss to be parsed word by
word
		for (int i = 0; i < wordNum; i++) { ss >> word; }
		return word;  //word (number position of a word in the phrase) is valid.
	}
}
bool Table::is_validWord(void) { return validWord; }
//Author: Demetrius E Johnson
//Purpose: prints the table (contents of 2d array storing the table) passed in from the
input file and built from the buildTable function
//Date Created: 8/12/21
//Date Modified: 8/19/21
void Table::printTable(ofstream& of) {

	for (int i = 0; i < numRow; i++) { //move to each row

		for (int j = 0; j < numCol; j++) { //print each row

			cout << arrayTable_2D[i][j] << " ";
			of << arrayTable_2D[i][j] << " ";
		}

		//move to next row:
		cout << endl;
		of << endl;
	}
}
//Author: Demetrius E Johnson
//Purpose: prints the character table, but capitalizing each word from the found user
phrase in the table
//Date Created: 8/12/21
//Date Modified: 8/19/21
void Table::printPhraseFoundTable(ofstream& of) {

	cout << "Phrase Found Table for: '" << userSearchVal << "':\n\n";
	of << "Phrase Found Table for: '" << userSearchVal << "':\n\n";


	//remember from foundCoord vector that every four positions stores the xy xy start
and end letter of a word in the phrase
```

```
	//also, x1 and x2 are at i and i+2 while y1 and y2 are at i+1 and i+3
	//remember getWordFromPhrase function takes only values > 0; the integer it takes refers to the first, second, third....etc... word from the user search value phrase
	//also remember that every four positions in the vector, starting from the front, coorespond to the xy xy coordinates of start letter and last letter of each word

	//use the below variables to acquire the start coordinates of each word, and to calculate the differences in their positions
	int x1, y1, x2, y2;
	int xDiff, yDiff;

	//first, capitalize all words at the proper locations in the table, each loop will capitalize a word in the table:
	//remember every 4 elements in the foundCoord vector represnts 2 coordinates (first and last letter of a word)
	for (int i = 0; i < foundCoord.size(); i += 4) {

		x1 = foundCoord[i];
		x2 = foundCoord[i + 2];

		y1 = foundCoord[i + 1];
		y2 = foundCoord[i + 3];

		xDiff = x2 - x1;
		yDiff = y2 - y1;

		//case: go from x1 to x2, y (column) stays the same - this capitalizes a word found from up to down
		if (xDiff >= 0 && yDiff == 0) {

			for (int j = x1; j <= x2; j++) {

				arrayTable_2D_copy[j][y1] = toupper(arrayTable_2D_copy[j][y1]);

			}
		}
		//case: go from y1 to y2, x (row) stays the same - this capitalizes a word found from left to right
		if (xDiff == 0 && yDiff >= 0) {

			for (int j = y1; j <= y2; j++) {

				arrayTable_2D_copy[x1][j] = toupper(arrayTable_2D_copy[x1][j]);

			}
		}
		//case: go from x2 to x1, y (column) stays the same - this capitalizes a word found from down to up
		if (xDiff <= 0 && yDiff == 0) {

			for (int j = x2; j <= x1; j++) {

				arrayTable_2D_copy[j][y1] = toupper(arrayTable_2D_copy[j][y1]);

			}
```

```
			}
			//case: go from y2 to y1, x (row) stays the same - this capitalizes a word
found from right to left
			if (xDiff == 0 && yDiff <= 0) {

				for (int j = y2; j <= y1; j++) {

					arrayTable_2D_copy[x1][j] =
toupper(arrayTable_2D_copy[x1][j]);

				}
			}
		}

		//now print table:

		for (int i = 0; i < numRow; i++) { //move to each row

			for (int j = 0; j < numCol; j++) { //print each row

				cout << arrayTable_2D_copy[i][j] << " ";
				of << arrayTable_2D_copy[i][j] << " ";
			}

			//move to next row:
			cout << endl;
			of << endl;
		}

		resetPhraseTable(); //now reset table for the next function call

		cout << endl;
		of << endl;
}

//Author: Demetrius E Johnson
//Purpose: prints the coordinates of the first and last letter of each word from the user
phrase that was found in the table
//Date Created: 8/12/21
//Date Modified: 8/19/21
void Table::printPhraseCoord(ofstream& of) {

		cout << "Phrase Coordinates for: '" << userSearchVal << "'.\n";
		of << "Phrase Coordinates for: '" << userSearchVal << "'.\n";

		//remember getWordFromPhrase function takes only values > 0; the integer it takes
refers to the first, second, third....etc... word from the user search value phrase
		//also remember that every four positions in the vector, starting from the front,
coorespond to the xy xy coordinates of start letter and last letter of each word

		int nextCoord = 0;

		for (int i = 0; i < numWordsInPhrase; i++) {

			cout << "word coordinate: " << getWordFromPhrase(i + 1) << ": <" <<
foundCoord[nextCoord] << "," << foundCoord[nextCoord + 1] << ">" << " to <" <<
foundCoord[nextCoord + 2] << "," << foundCoord[nextCoord + 3] << ">.\n";
```

```
            of << "word coordinate: " << getWordFromPhrase(i + 1) << ": <" <<
foundCoord[nextCoord] << "," << foundCoord[nextCoord + 1] << ">" << " to <" <<
foundCoord[nextCoord + 2] << "," << foundCoord[nextCoord + 3] << ">.\n";
            nextCoord += 4; //move to start of next coordinates in the phrase vector;
every 4 elements represents a set of coordinates
      }
      cout << "End coordinates.\n";
      of << "End coordinates.\n";
}
//Author: Demetrius E Johnson
//Purpose: part of the backtracking algorithm, this function initiates the word search,
searching at every location in the table using backtracking when necessary
//Date Created: 8/12/21
//Date Modified: 8/19/21
bool Table::wordSearch(ofstream& of) {

      //search order: right, down, left, up.

      int currentWord = 1; //track current word in the phrase that we are searching
      int phrasePos = 0;    //track the position (char) in the phrase we are searching
      int x;                        //track current row position in table relative to a
word found from the phrase
      int y;                        //track current column position in the table relative
to a word found from the phrase
      foundCoord.clear();  //make sure coordinate vector is clear before starting search

      cout << "Start search at (0,0).\n";
      of << "Start search at (0,0).\n";

      //use below loops to begin going to every location in the table (remeber boolean
&& and || are evaluated left to right):
      //start search at every location, and if start location is a match, then proceed
with back tracking algorithm search for user phrase
      for (int i = 0; i < numRow; i++) {

            x = i; //set x initially relative to next table search position

            for (int j = 0; j < numCol; j++) {

                  y = j;                                    //set y initially relative
to next table search position
                  currentWord = 1;                  //reset current word to 1 (the
first word to be searched)
                  phrasePos = 0;                          //reset position searched
from user phrase back to start
                  foundCoord.clear();               //make sure coordinate vector is
clear before starting search from a new position
                  resetBoolTable();                 //make sure bool table is reset
before starting search at a new position

                  //if current position is not in the table we can output a message
and move to next position in the table to search; otherwise we begin search for the rest
of phrase:
                  if (userSearchVal[phrasePos] != arrayTable_2D[x][y]) {

                        cout << "search: currPos: (" << x << "," << y << "): '" <<
userSearchVal[phrasePos] << "' not found.\n";
```

```
                        of << "search: currPos: (" << x << "," << y << "): '" <<
userSearchVal[phrasePos] << "' not found.\n";
                    }
                    else {

                        cout << "search: currPos: (" << x << "," << y << "): '" <<
arrayTable_2D[x][y] << "' found.\n";
                        of << "search: currPos: (" << x << "," << y << "): '" <<
arrayTable_2D[x][y] << "' found.\n";
                        addCoord(x, y);                                 //add
coordinate of first letter of potentially found word from the phrase
                        usedCoordinates[x][y] = true; //coordinate used in the search,
set it to true so there are no potential overlap or loops later in search
                        phrasePos++;

                        //case: we have reached the end of the phrase - phrase has
been found in the table; phrase was just a single character:
                        if (userSearchVal.size() == 1) {

                            cout << "search: <" << getWordFromPhrase(currentWord)
<< "> found.\n";
                            of << "search: <" << getWordFromPhrase(currentWord) <<
"> found.\n";
                            addCoord(x, y);                                 //add
coordinate of last letter of found word from the phrase
                            userPhraseFound = true;
                            cout << "search: SUCCESS!: '" << userSearchVal << "'
found!\n";
                            of << "search: SUCCESS!: '" << userSearchVal << "'
found!\n";
                            return true;
                        }
                        //case: current word in the phrase is only a single character;
need to skip the space and move to next word to initiate remaining search:
                        if (userSearchVal[phrasePos] == ' ') {

                            cout << "search: <" << getWordFromPhrase(currentWord)
<< "> found.\n";
                            of << "search: <" << getWordFromPhrase(currentWord) <<
"> found.\n";
                            addCoord(x, y);                                 //add
coordinate of last letter of found word from the phrase
                            currentWord++;                                  //move
to next word to output from the phrase
                            phrasePos++;                            //skip the
space - move to first letter in next word
                            foundRight();                           //set flags
for which direction the last word was found
                        }

                        //now continue to search for the current word and the rest of
the phrase from the current position:

                        userPhraseFound = search_allDir(x, y, phrasePos, currentWord,
of);
                        if (userPhraseFound == true) {
```

```
                                    cout << "search: SUCCESS!: '" << userSearchVal << "'
found!\n";
                                    of << "search: SUCCESS!: '" << userSearchVal << "'
found!\n";
                                    return true;
                             }
                      }
               }//otherwise loop iteration will end and j will increment and move to next
character in the row to start search again
       }//otherwise loop iteration will end and i will increment and move to the next row
in the table to begin starting searches from those elements

       cout << "search: FAILED!: '" << userSearchVal << "' not found!\n";
       of << "search: FAILED!: '" << userSearchVal << "' not found!\n";

       userPhraseFound = false;
       return false; //if we reach here, all possible paths (state space tree, but cut
off from efficient backtracking) checked; word/phrase is not in table
}
//Author: Demetrius E Johnson
//Purpose: resets the phrase found table - this table needs to be reset to default all-
lowercase table after the table has been printed so that it can be re-capitalized based
on the next found user phrase
//Date Created: 8/12/21
//Date Modified: 8/19/21
void Table::resetPhraseTable(void) {

       for (int i = 0; i < numRow; i++) { //move to each row

              for (int j = 0; j < numCol; j++) { //reset row

                     arrayTable_2D_copy[i][j] = arrayTable_2D[i][j];
              }
       }
}
//Author: Demetrius E Johnson
//Purpose: reset the bool table - this will reset the table whenever backtracking occurs
so that all positions in the table will be cleared and available for use if search
algorithm finds a match during another search position
//Date Created: 8/12/21
//Date Modified: 8/19/21
void Table::resetBoolTable(void) {

       for (int i = 0; i < numRow; i++) { //move to each row

              for (int j = 0; j < numCol; j++) { //reset row

                     usedCoordinates[i][j] = false;
              }
       }
}
//Author: Demetrius E Johnson
//Purpose: call the destroy function so that when a Table object is destroyed, all
dynamically allocated memory is deallocated if it has not been already
//Date Created: 8/12/21
//Date Modified: 8/19/21
Table::~Table() {
```

```
	//call destroy table function, which will delete all dynamically allocated memory
from the 2D array:
	destroyTable();

}
```

# 9. Test Plan Version 3

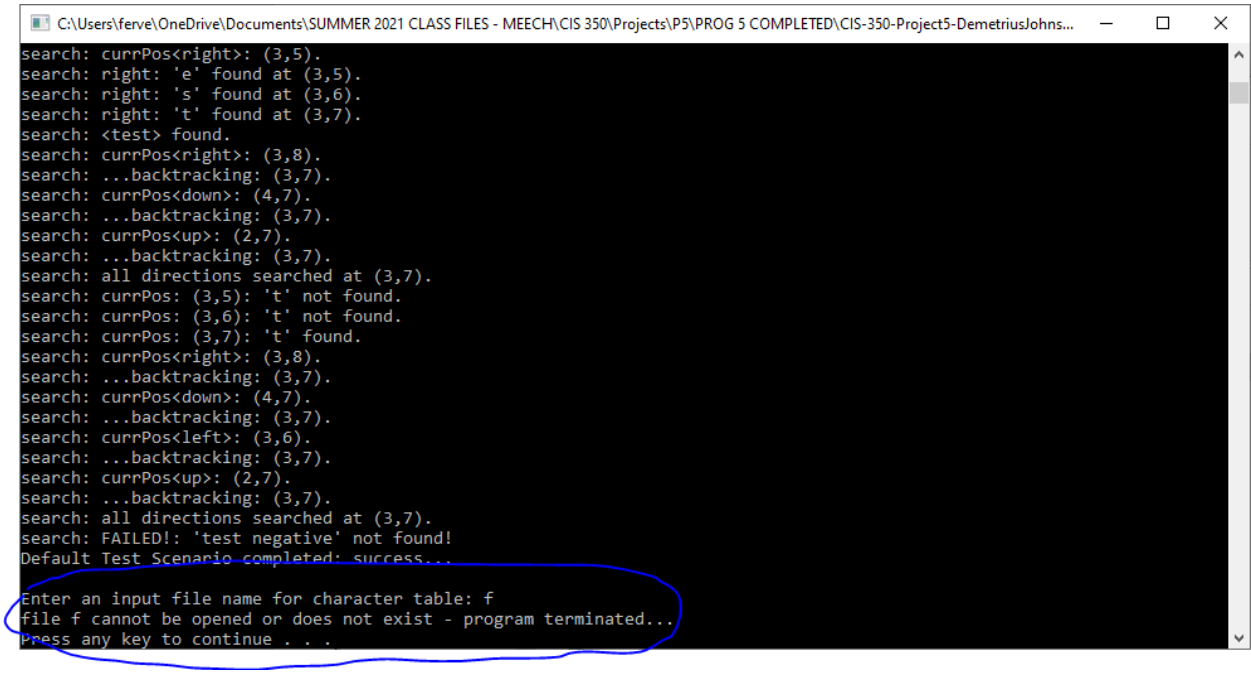
| Test Strategy | Test Number | Description | Input | Expected Output | Actual Output | Pass/Fail |
|---|---|---|---|---|---|---|
| File Testing | 1 | File does not exist | Wrong file name | "file not found" | What was the actual output from your code | pass |
| File Testing | 2 | File exists but empty | Empty file | "file is empty" | See screenshot | pass |
| Valid data | 3 | Valid data from input file | Valid test file | "test table created…user input table created" | See screenshot | pass |
| Invalid data | 4 | Invalid number of rows or columns from input file | Test file with incorrect number of rows and columns | "invalid row or column value for table" | See screenshot | pass |
| Invalid data | 5 | Invalid character in input table | Input $ into table | "non alphabetical char in table…" | See screenshot | pass |
| Invalid data | 6 | Invalid user input for phrase | Input test$ | "re-enter a phrase, invalid char found" | See screenshot | pass |
| Valid data | 7 | Quit program nonalphabetical entered by user | Enter ` | "` entered; program quitting" | See screenshot | pass |
| Valid data | 8 | Find a phrase, restart loop | Search "test phrase" | Phrase found; output coordinates; output table | See screenshot | pass |
| Valid data | 9 | Do not find a phrase or word | "test phrase | Phrase not found | See screenshot | pass |

| | | | | | | |
|---|---|---|---|---|---|---|
| | | | not here” | | | |
| Valid data | 10 | 4 additional tests: testing every direction of search | Capital letters, spaces, etc in user input | Valid search grid, and req1 requirements | See screenshot | pass |
| Valid data | 11 | Testing req1.dat + additional tests | Capital letters, spaces, etc in user input | Valid search grid, and req1 requirements | See screenshot | pass |
| Valid data | 12 | Testing req4.dat + additional tests | Capital letters, spaces, etc in user input | Valid search grid, and req4 requirements | See screenshot | pass |
| Valid data | 13 | Testing req5.dat + additional tests | Capital letters, spaces, etc in user input | Valid search grid, and req5 requirements | See screenshot | pass |
| Valid data | 14 | Testing req6.dat + additional tests | Capital letters, spaces, etc in user input | Valid search grid, and req6 requirements | See screenshot | pass |

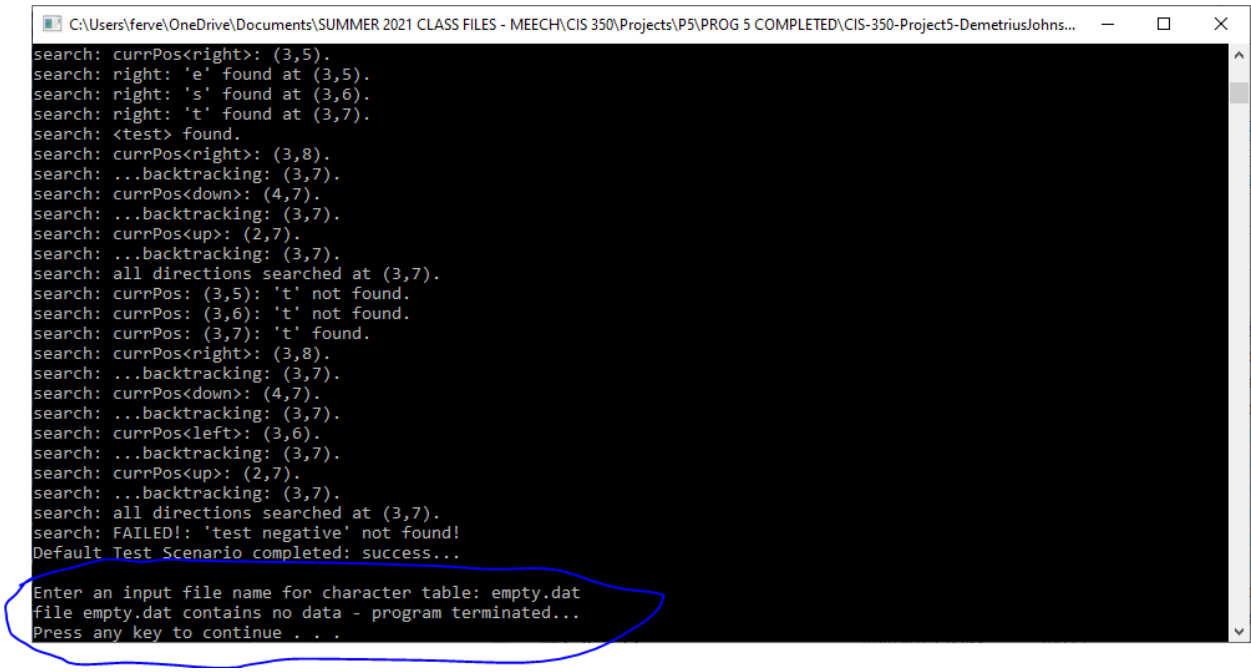
## 10. Screenshots

Screenshots of your testing goes here. YOU MUST HAVE A SCREENSHOT FOR EVERY TEST CASE (unless the output goes to a file in which case the screenshots are a sample of output and specifically any output that does not get sent to the output file – e.g. exceptions). A screenshot may picture multiple test cases. For each screen shot caption it with a list of the test cases are depicted in it.

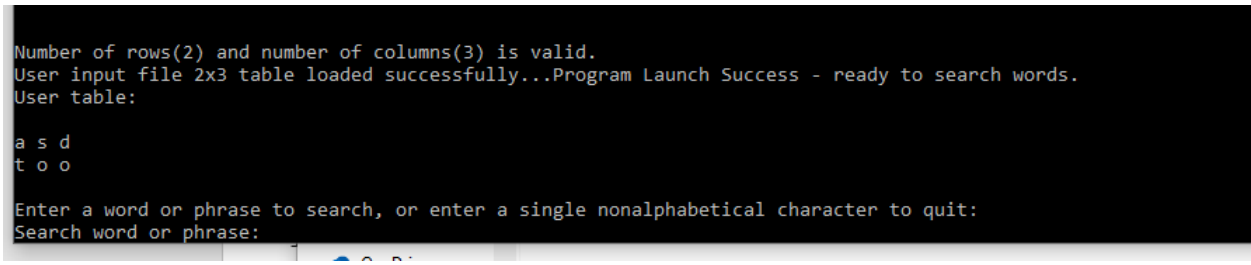
TEST 1: file not found error



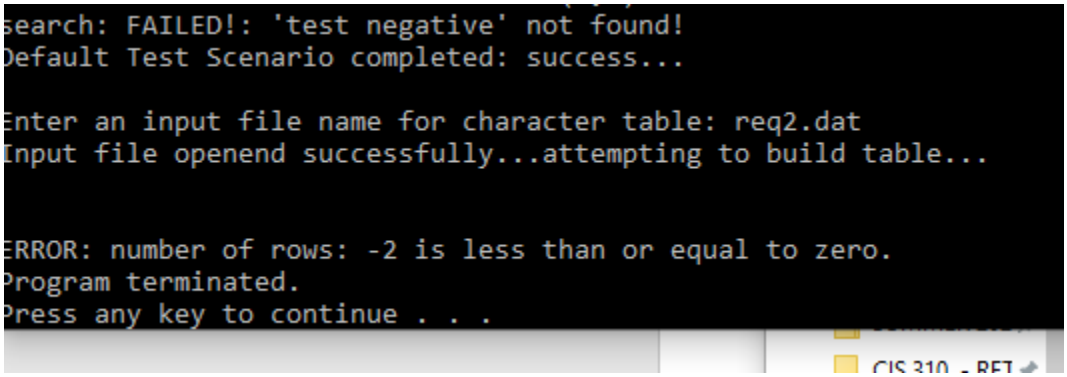

TEST 2: empty file case

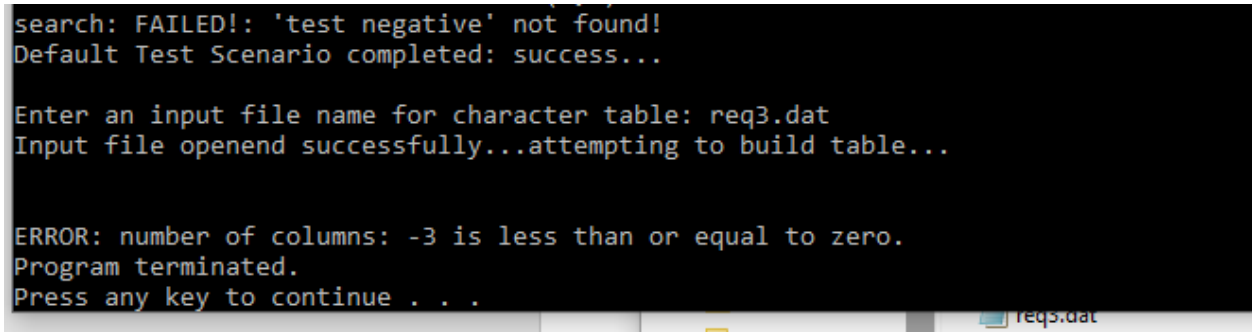



TEST 3: valid test file and user file

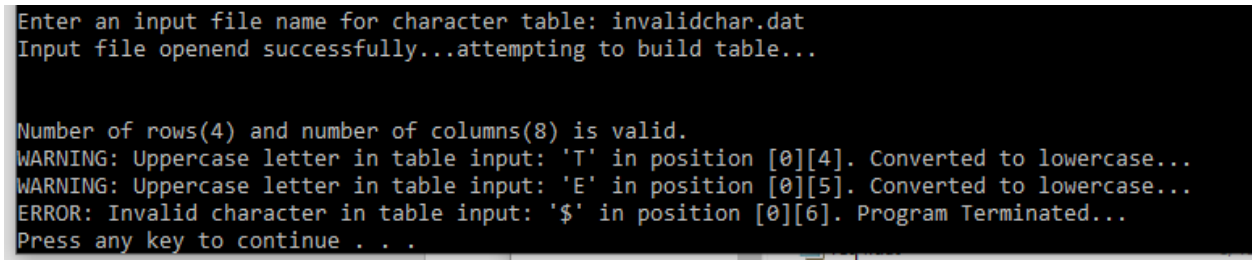


TEST 4:  invalid number of rows or columns (req2.dat and req3.dat)

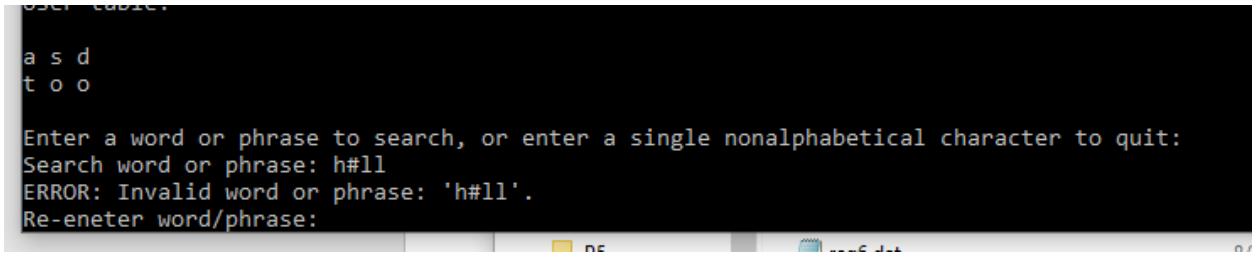


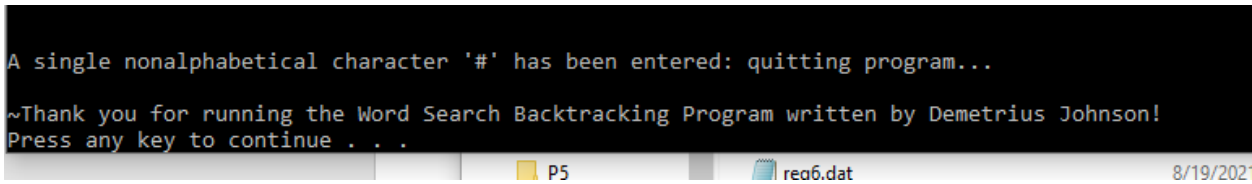

TEST 5: invalid character in table and uppercase letter warning



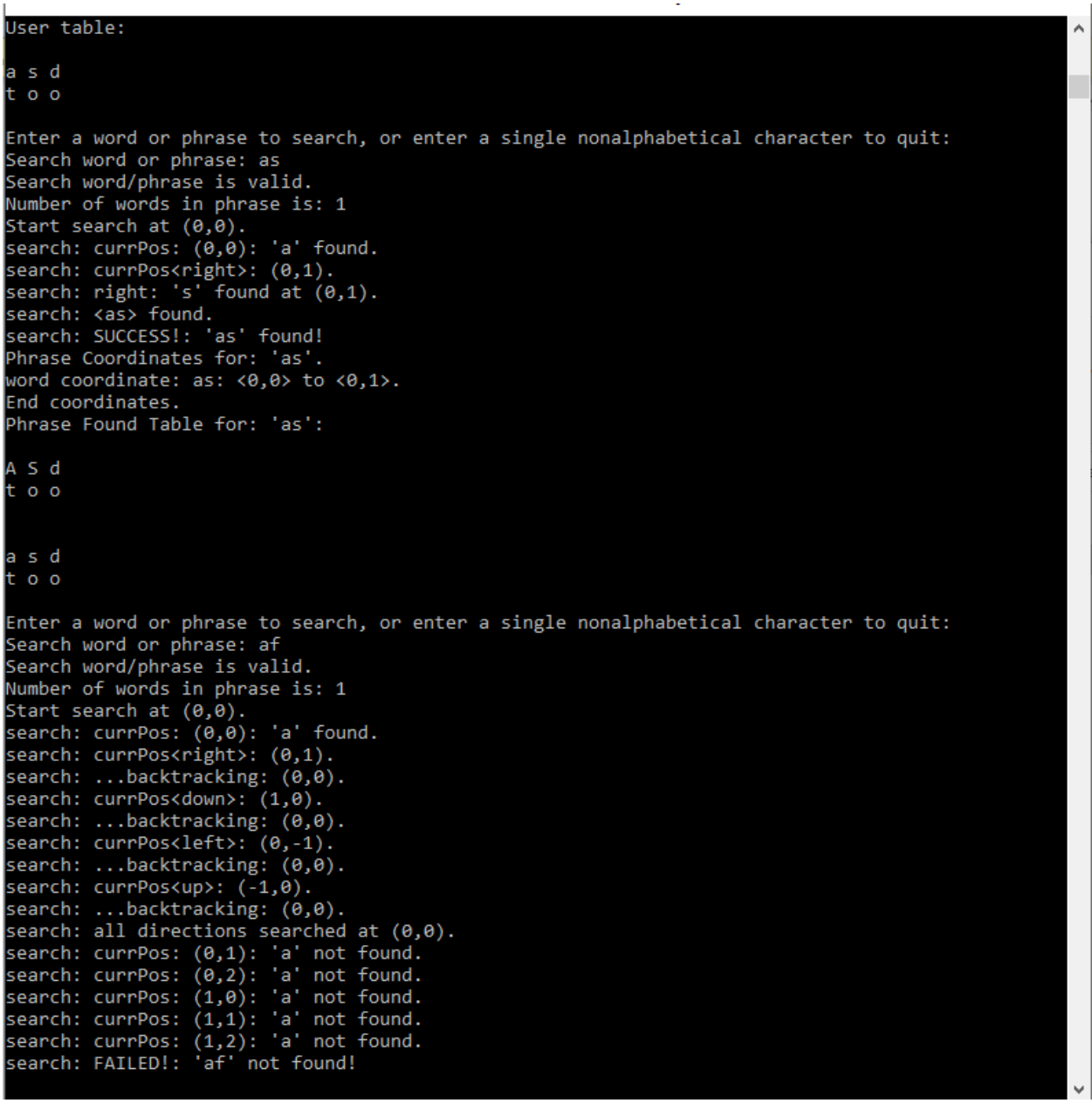

TEST 6: invalid user input



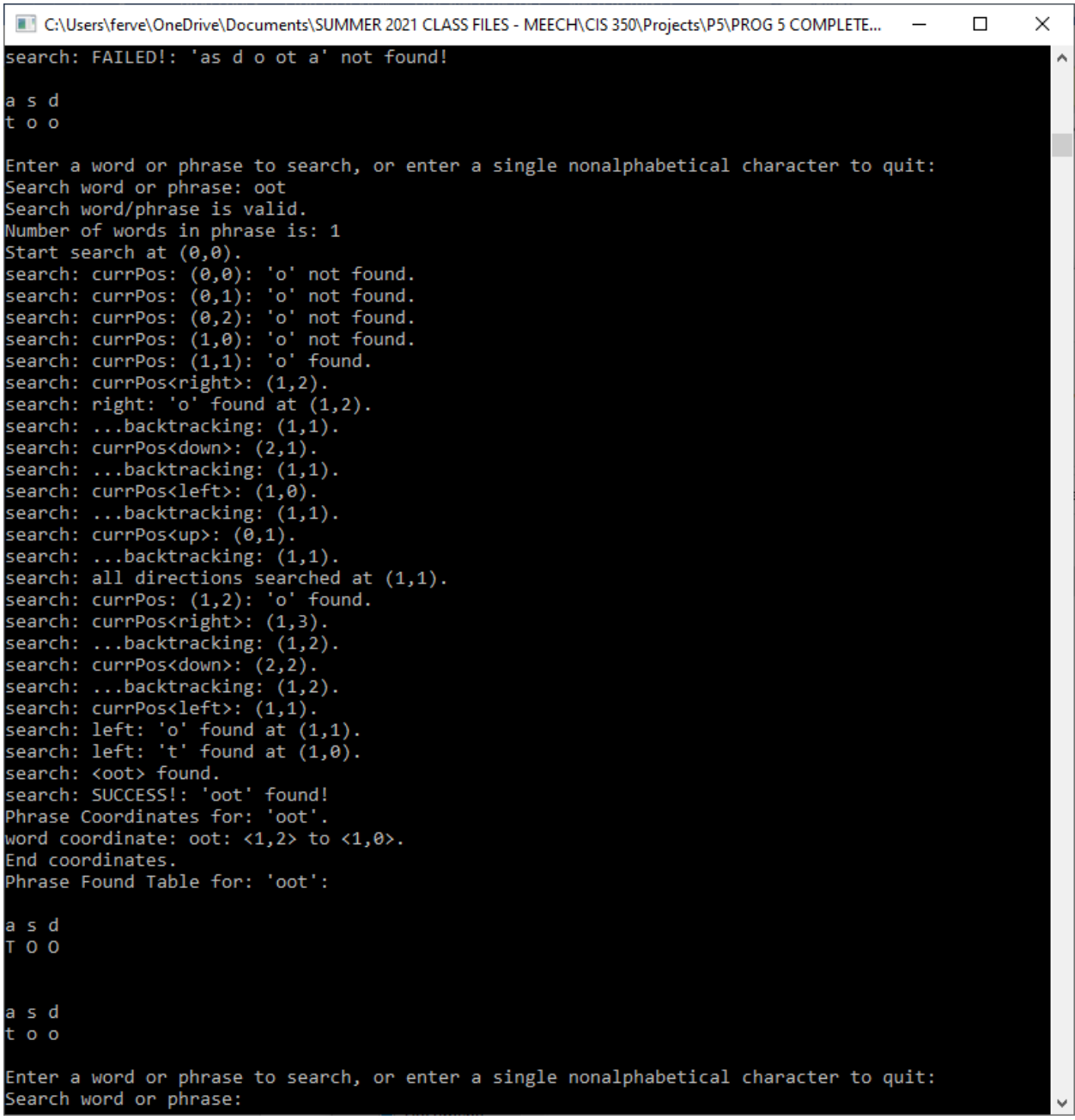

TEST 7: quit program by entering a single, nonalphabetical input, also shows

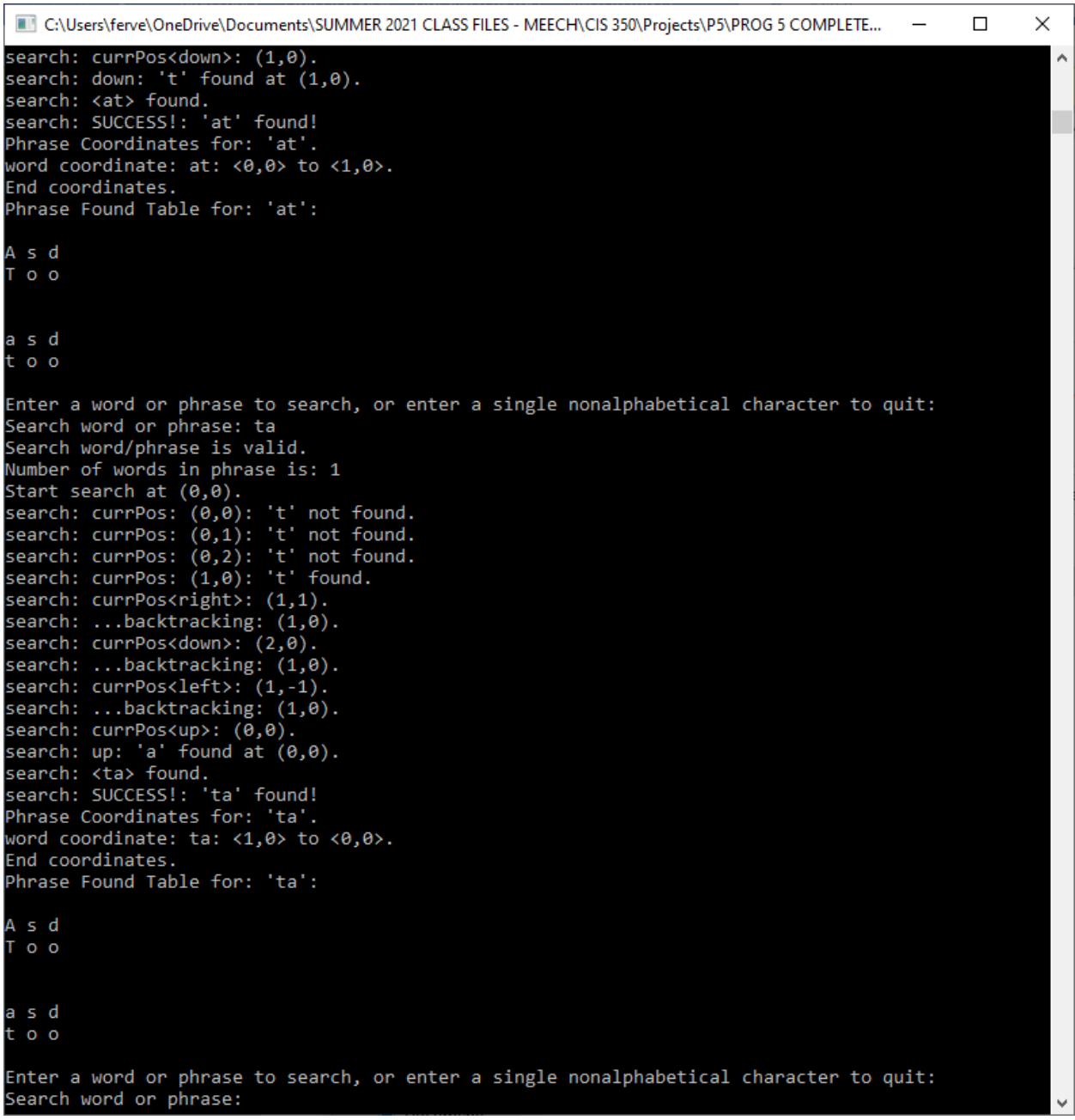


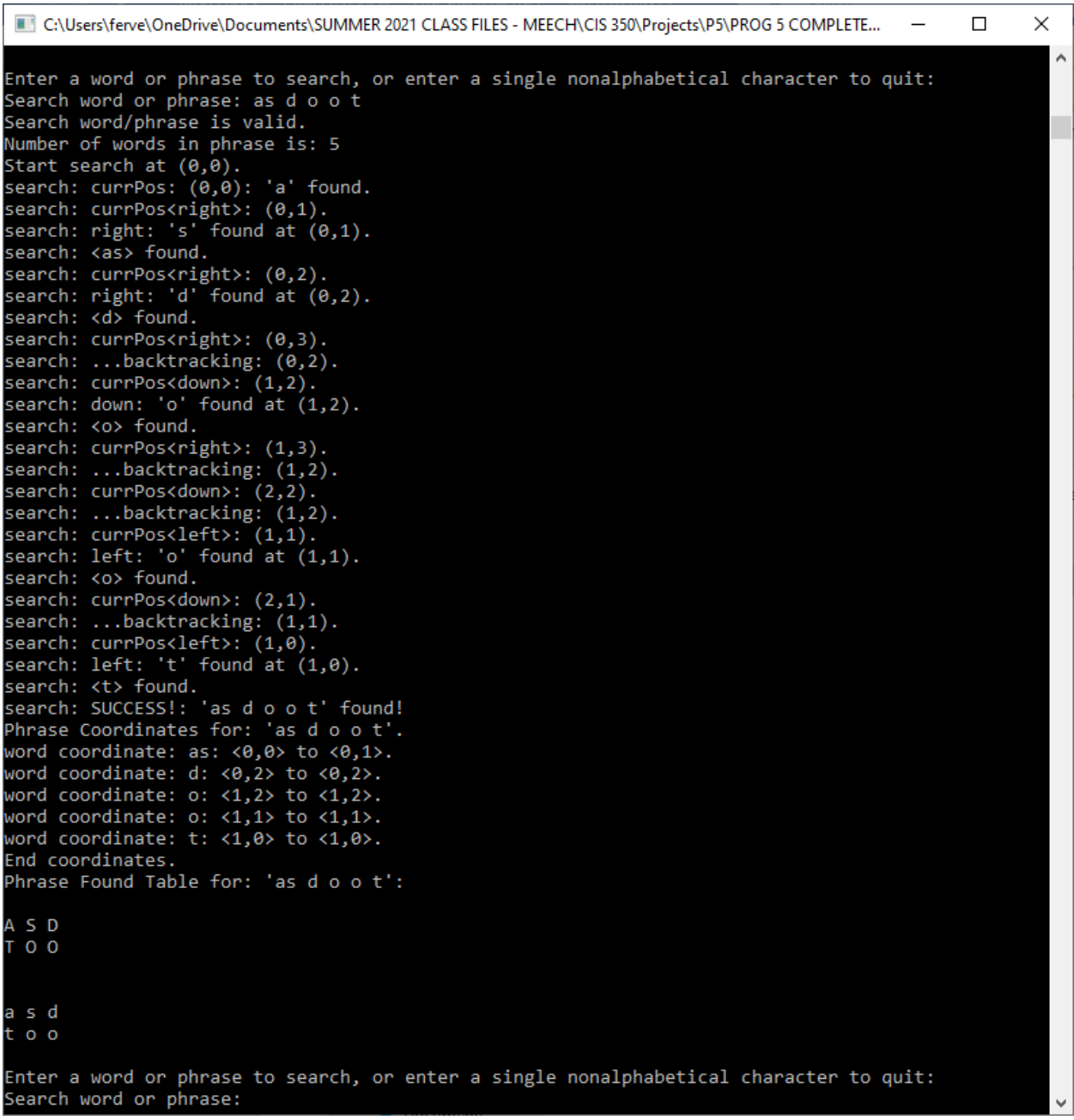

TESTS 8 and 9: user phrase found and not found loop



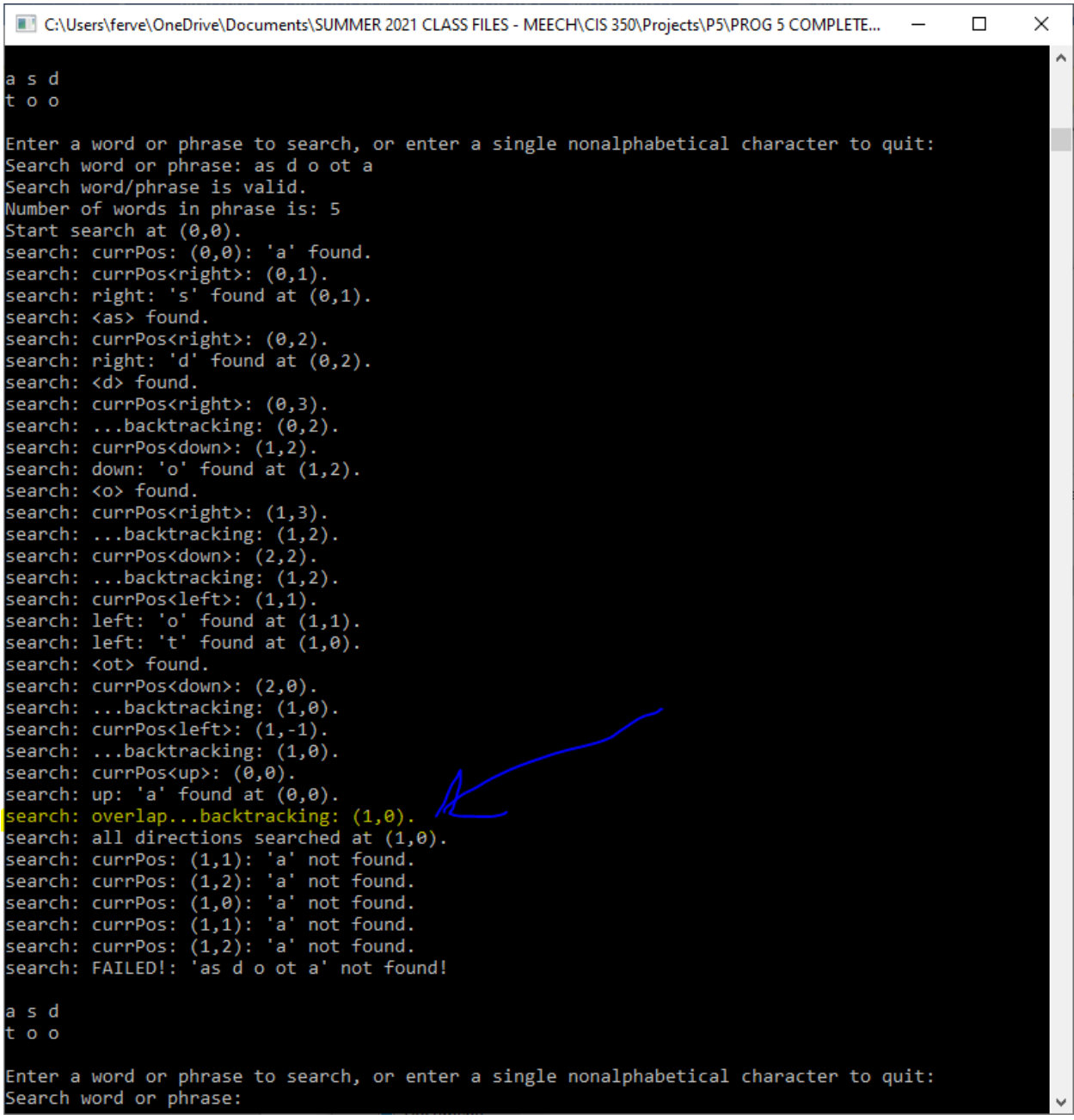
TEST 10: show finding various directions of found phrases


C:\Users\ferve\OneDrive\Documents\SUMMER 2021 CLASS FILES - MEECH\CIS 350\Projects\P5\PROG 5 COMPLETE...
search: FAILED!: 'as d o ot a' not found!
a s d
t o o
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase: oot
Search word/phrase is valid.
Number of words in phrase is: 1
Start search at (0,0).
search: currPos: (0,0): 'o' not found.
search: currPos: (0,1): 'o' not found.
search: currPos: (0,2): 'o' not found.
search: currPos: (1,0): 'o' not found.
search: currPos: (1,1): 'o' found.
search: currPos<right>: (1,2).
search: right: 'o' found at (1,2).
search: ...backtracking: (1,1).
search: currPos<down>: (2,1).
search: ...backtracking: (1,1).
search: currPos<left>: (1,0).
search: ...backtracking: (1,1).
search: currPos<up>: (0,1).
search: ...backtracking: (1,1).
search: all directions searched at (1,1).
search: currPos: (1,2): 'o' found.
search: currPos<right>: (1,3).
search: ...backtracking: (1,2).
search: currPos<down>: (2,2).
search: ...backtracking: (1,2).
search: currPos<left>: (1,1).
search: left: 'o' found at (1,1).
search: left: 't' found at (1,0).
search: <oot> found.
search: SUCCESS!: 'oot' found!
Phrase Coordinates for: 'oot'.
word coordinate: oot: <1,2> to <1,0>.
End coordinates.
Phrase Found Table for: 'oot':
a s d
T O O
a s d
t o o
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase:

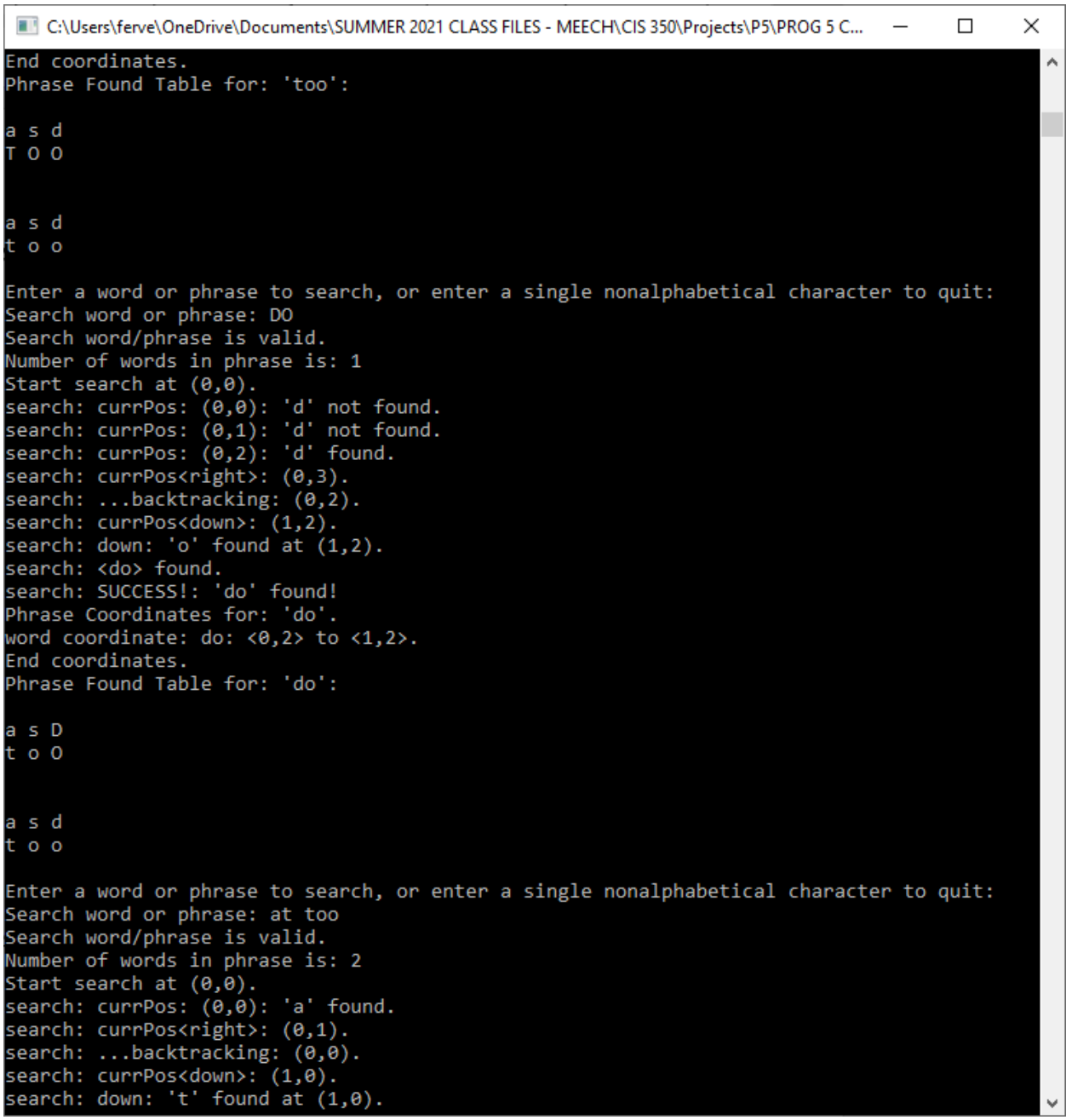

C:\Users\ferve\OneDrive\Documents\SUMMER 2021 CLASS FILES - MEECH\CIS 350\Projects\P5\PROG 5 COMPLETE...
search: currPos<down>: (1,0).
search: down: 't' found at (1,0).
search: <at> found.
search: SUCCESS!: 'at' found!
Phrase Coordinates for: 'at'.
word coordinate: at: <0,0> to <1,0>.
End coordinates.
Phrase Found Table for: 'at':
A s d
T o o
a s d
t o o
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase: ta
Search word/phrase is valid.
Number of words in phrase is: 1
Start search at (0,0).
search: currPos: (0,0): 't' not found.
search: currPos: (0,1): 't' not found.
search: currPos: (0,2): 't' not found.
search: currPos: (1,0): 't' found.
search: currPos<right>: (1,1).
search: ...backtracking: (1,0).
search: currPos<down>: (2,0).
search: ...backtracking: (1,0).
search: currPos<left>: (1,-1).
search: ...backtracking: (1,0).
search: currPos<up>: (0,0).
search: up: 'a' found at (0,0).
search: <ta> found.
search: SUCCESS!: 'ta' found!
Phrase Coordinates for: 'ta'.
word coordinate: ta: <1,0> to <0,0>.
End coordinates.
Phrase Found Table for: 'ta':
A s d
T o o
a s d
t o o
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase:

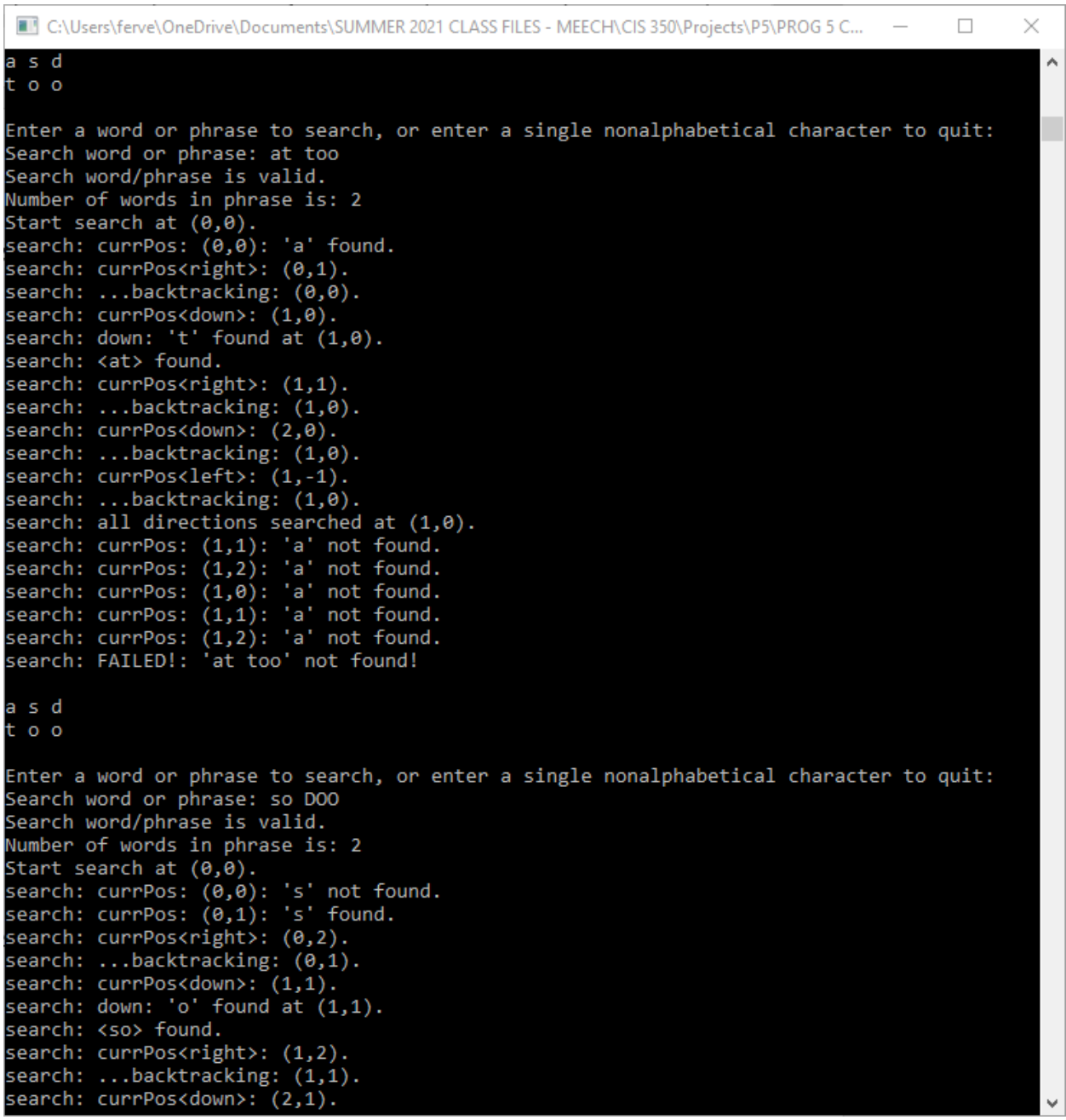


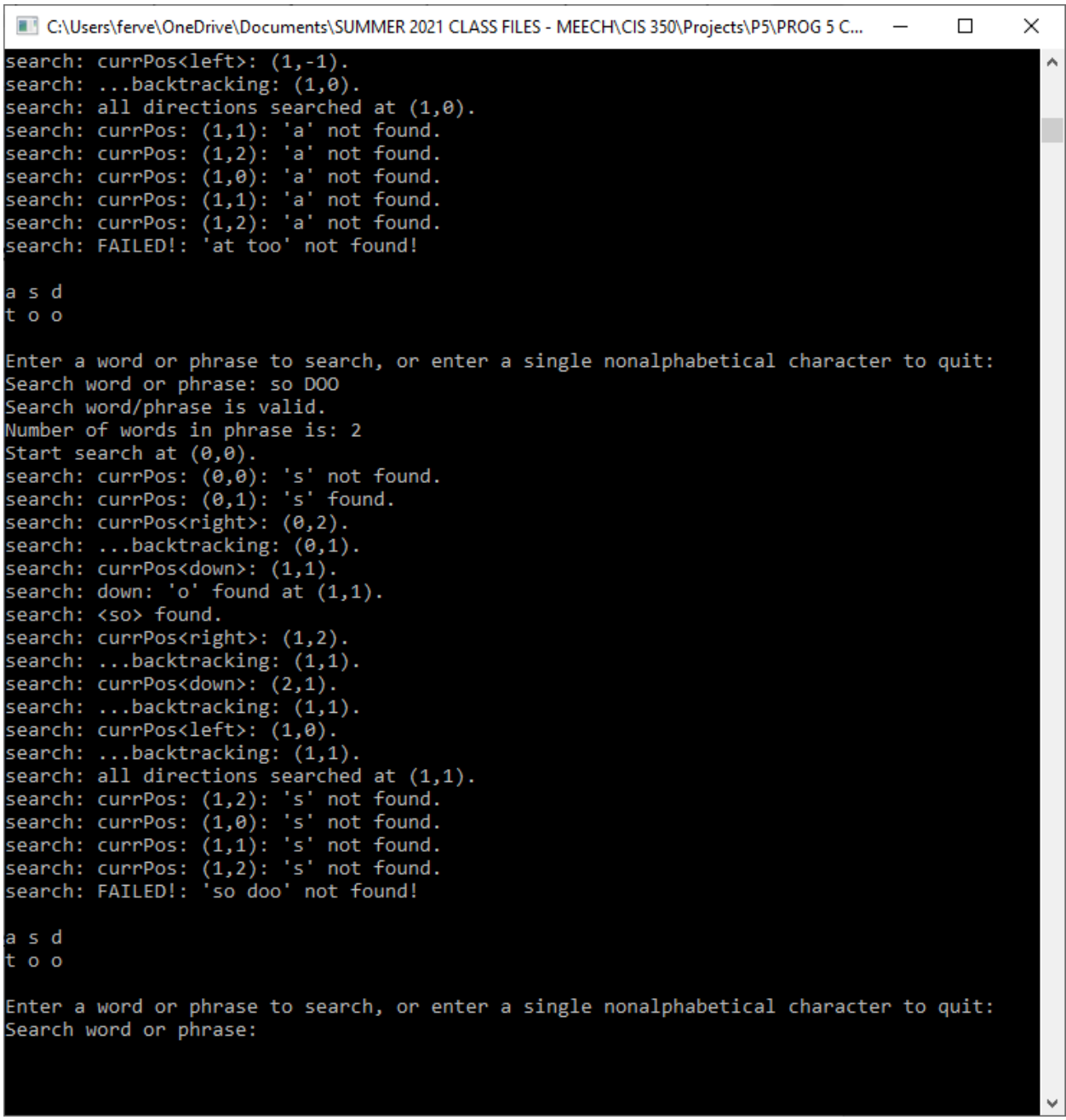
TEST 11: show overlapping is accounted for when searching for a phrase



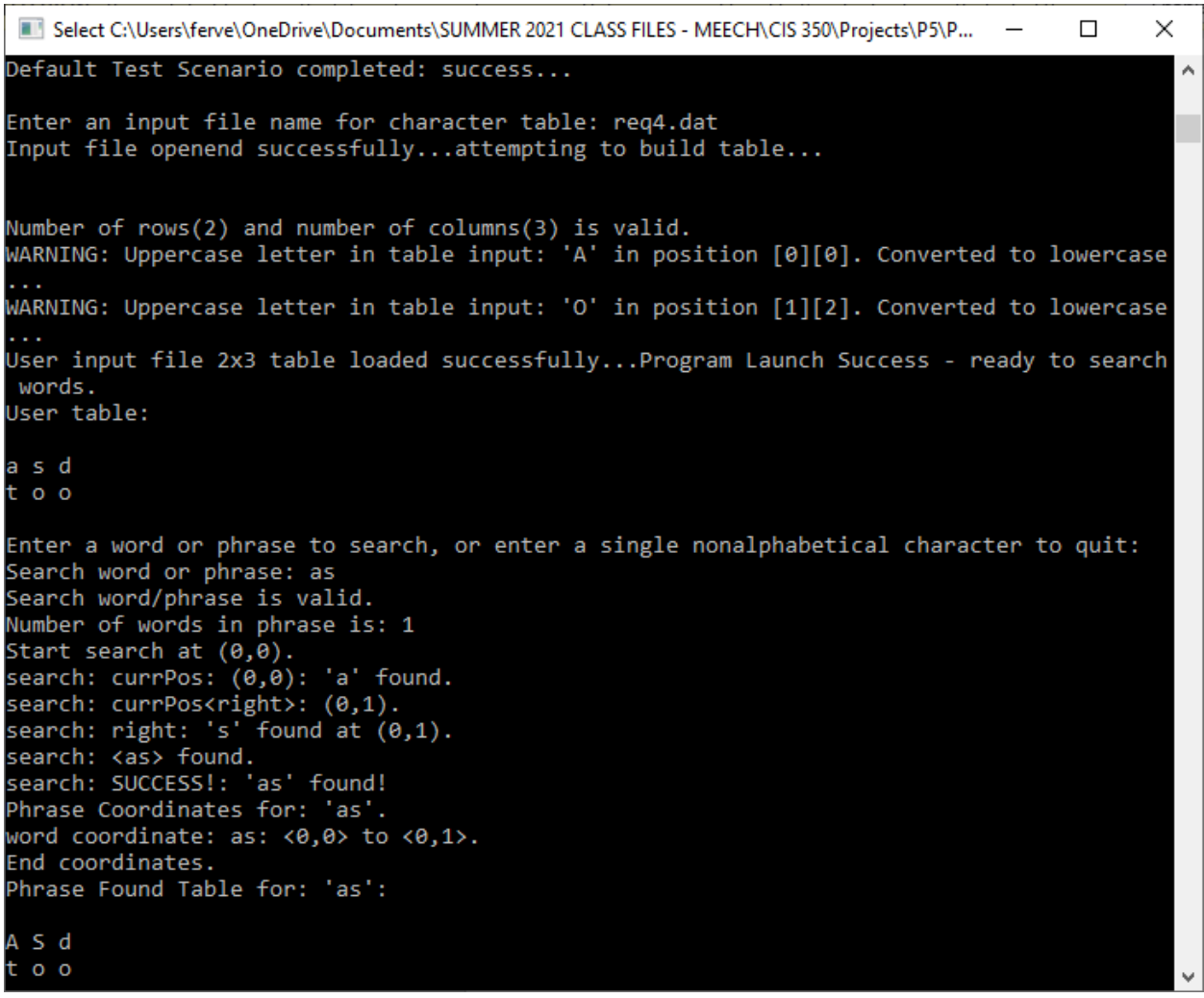
TEST 12: req1.dat


C:\Users\ferve\OneDrive\Documents\SUMMER 2021 CLASS FILES - MEECH\CIS 350\Projects\P5\PROG 5 C...
End coordinates.
Phrase Found Table for: 'too':
a s d
T O O
a s d
t o o
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase: DO
Search word/phrase is valid.
Number of words in phrase is: 1
Start search at (0,0).
search: currPos: (0,0): 'd' not found.
search: currPos: (0,1): 'd' not found.
search: currPos: (0,2): 'd' found.
search: currPos<right>: (0,3).
search: ...backtracking: (0,2).
search: currPos<down>: (1,2).
search: down: 'o' found at (1,2).
search: <do> found.
search: SUCCESS!: 'do' found!
Phrase Coordinates for: 'do'.
word coordinate: do: <0,2> to <1,2>.
End coordinates.
Phrase Found Table for: 'do':
a s D
t o O
a s d
t o o
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase: at too
Search word/phrase is valid.
Number of words in phrase is: 2
Start search at (0,0).
search: currPos: (0,0): 'a' found.
search: currPos<right>: (0,1).
search: ...backtracking: (0,0).
search: currPos<down>: (1,0).
search: down: 't' found at (1,0).

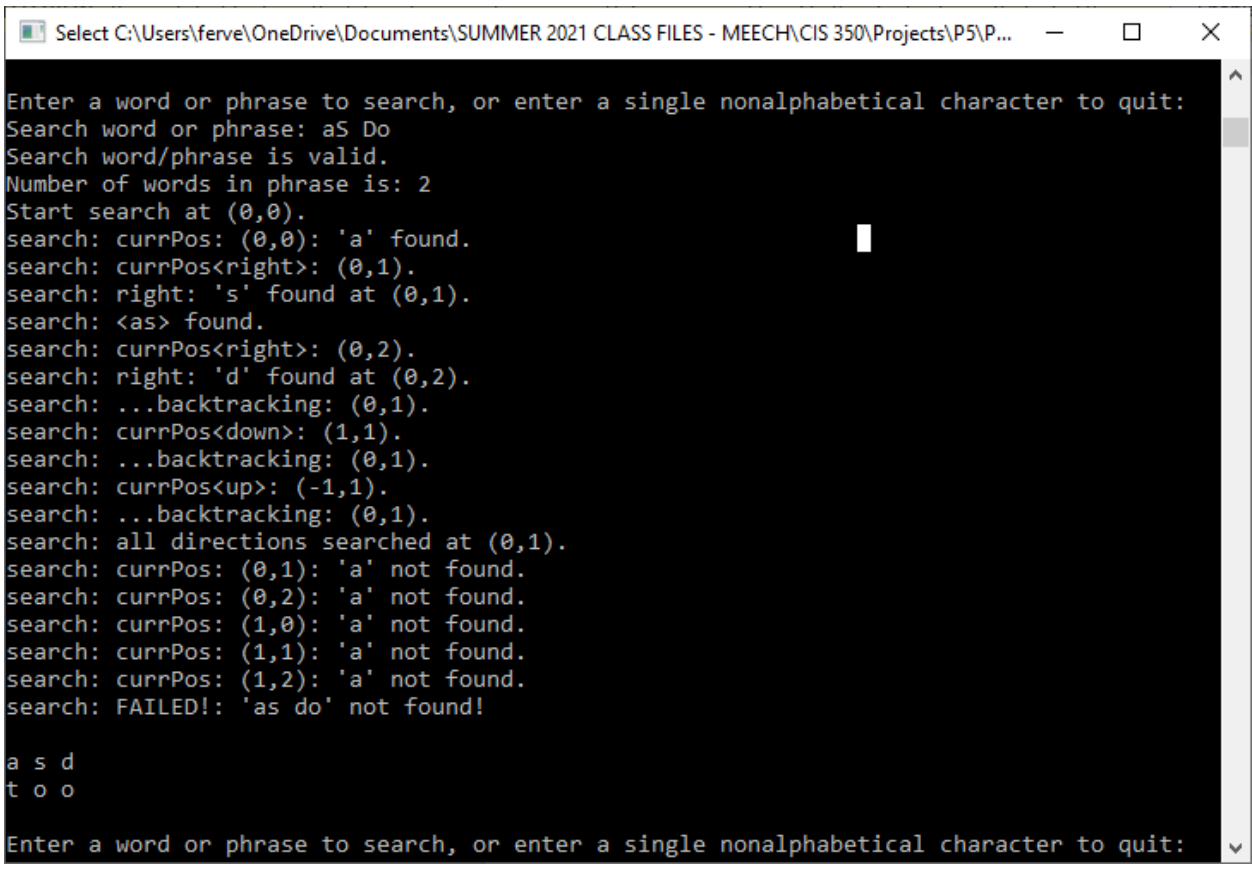

C:\Users\ferve\OneDrive\Documents\SUMMER 2021 CLASS FILES - MEECH\CIS 350\Projects\P5\PROG 5 C...
a s d
t o o
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase: at too
Search word/phrase is valid.
Number of words in phrase is: 2
Start search at (0,0).
search: currPos: (0,0): 'a' found.
search: currPos<right>: (0,1).
search: ...backtracking: (0,0).
search: currPos<down>: (1,0).
search: down: 't' found at (1,0).
search: <at> found.
search: currPos<right>: (1,1).
search: ...backtracking: (1,0).
search: currPos<down>: (2,0).
search: ...backtracking: (1,0).
search: currPos<left>: (1,-1).
search: ...backtracking: (1,0).
search: all directions searched at (1,0).
search: currPos: (1,1): 'a' not found.
search: currPos: (1,2): 'a' not found.
search: currPos: (1,0): 'a' not found.
search: currPos: (1,1): 'a' not found.
search: currPos: (1,2): 'a' not found.
search: FAILED!: 'at too' not found!
a s d
t o o
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase: so DOO
Search word/phrase is valid.
Number of words in phrase is: 2
Start search at (0,0).
search: currPos: (0,0): 's' not found.
search: currPos: (0,1): 's' found.
search: currPos<right>: (0,2).
search: ...backtracking: (0,1).
search: currPos<down>: (1,1).
search: down: 'o' found at (1,1).
search: <so> found.
search: currPos<right>: (1,2).
search: ...backtracking: (1,1).
search: currPos<down>: (2,1).



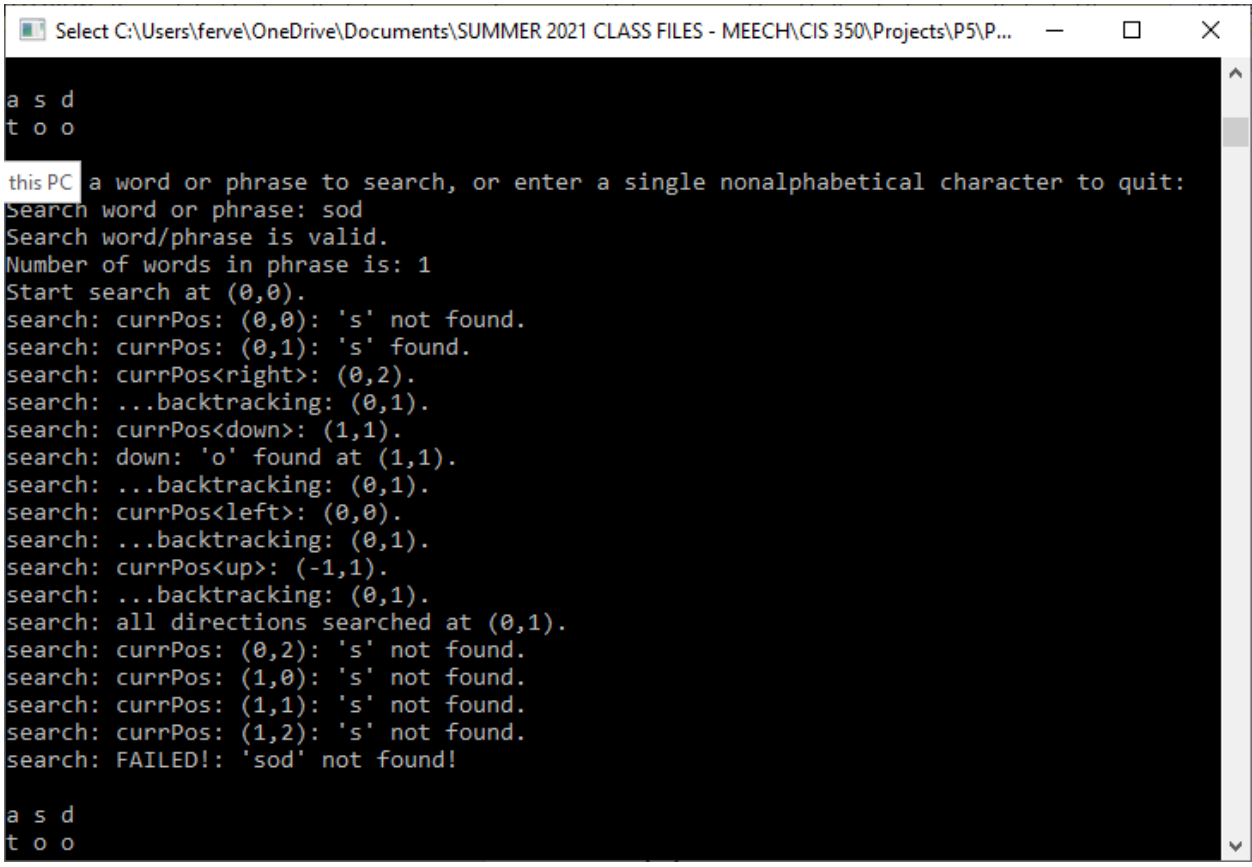

TEST 13: req4.dat – valid with capital letters – also notice capital user input sent to lower case
(search for “as”)



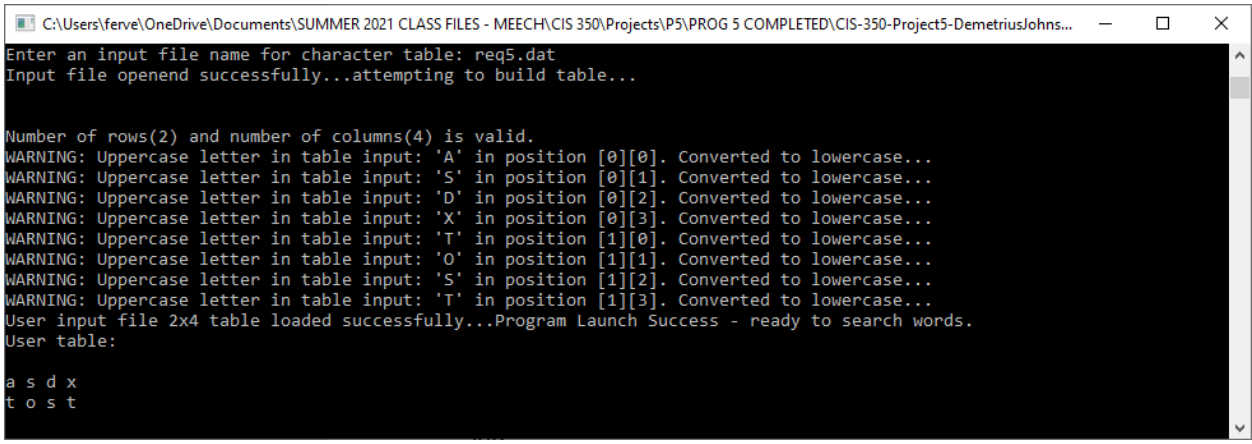

(search for “aS Do”)



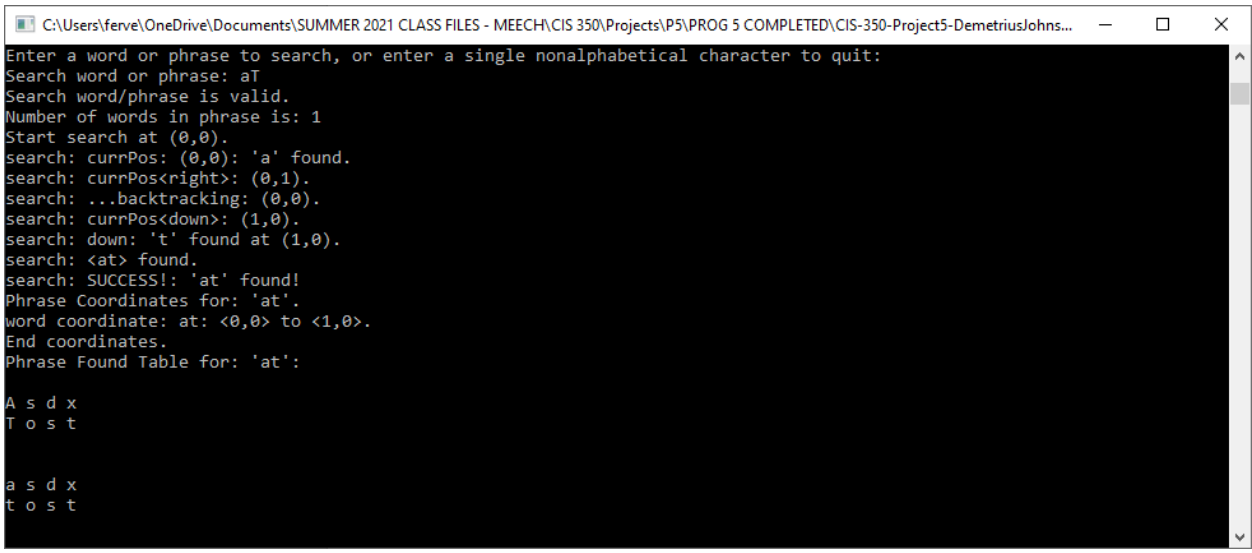
(search for “sod”)



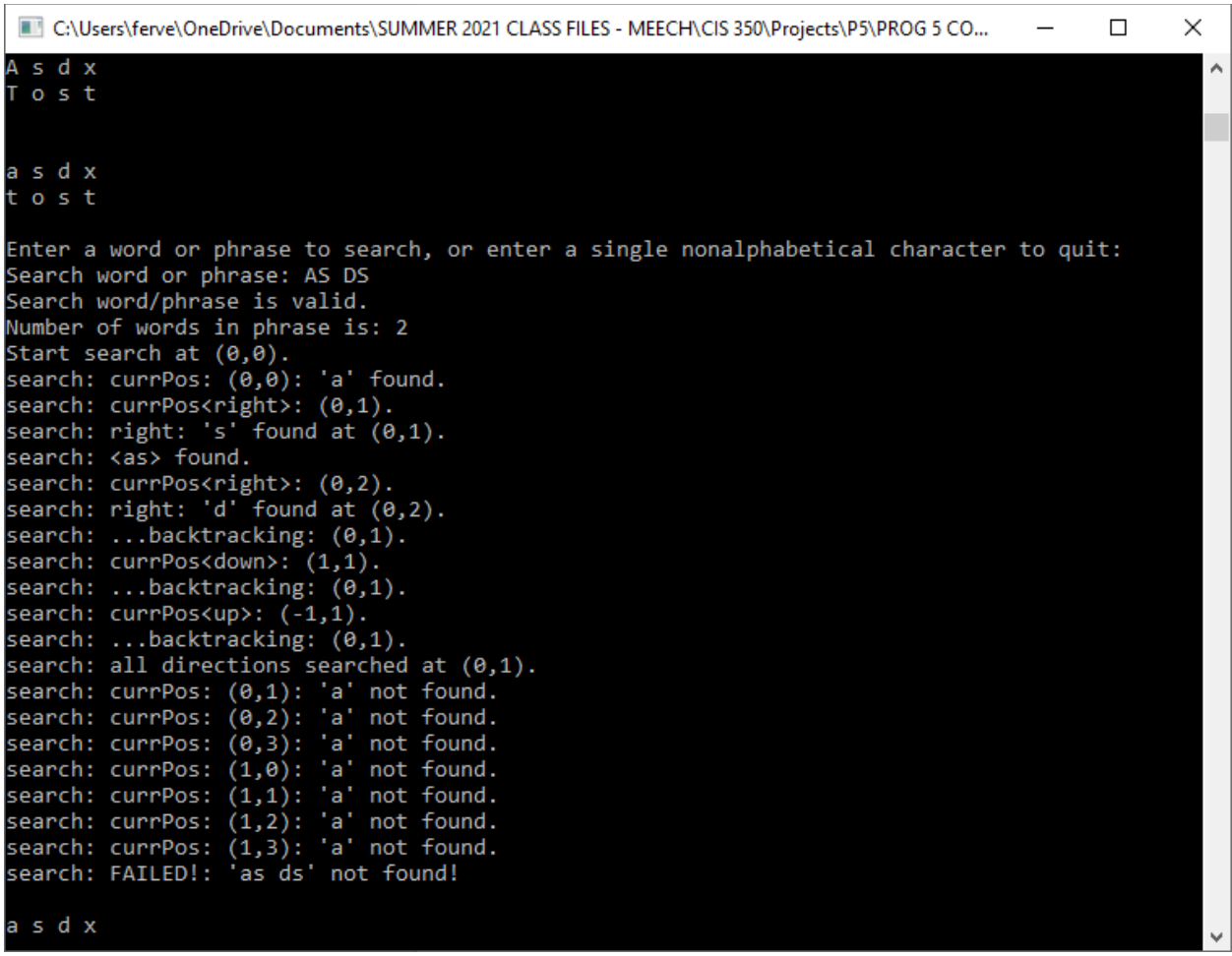

TEST 14: req5.dat – valid with all capital letters
(table built successful with warning message for cap letters)



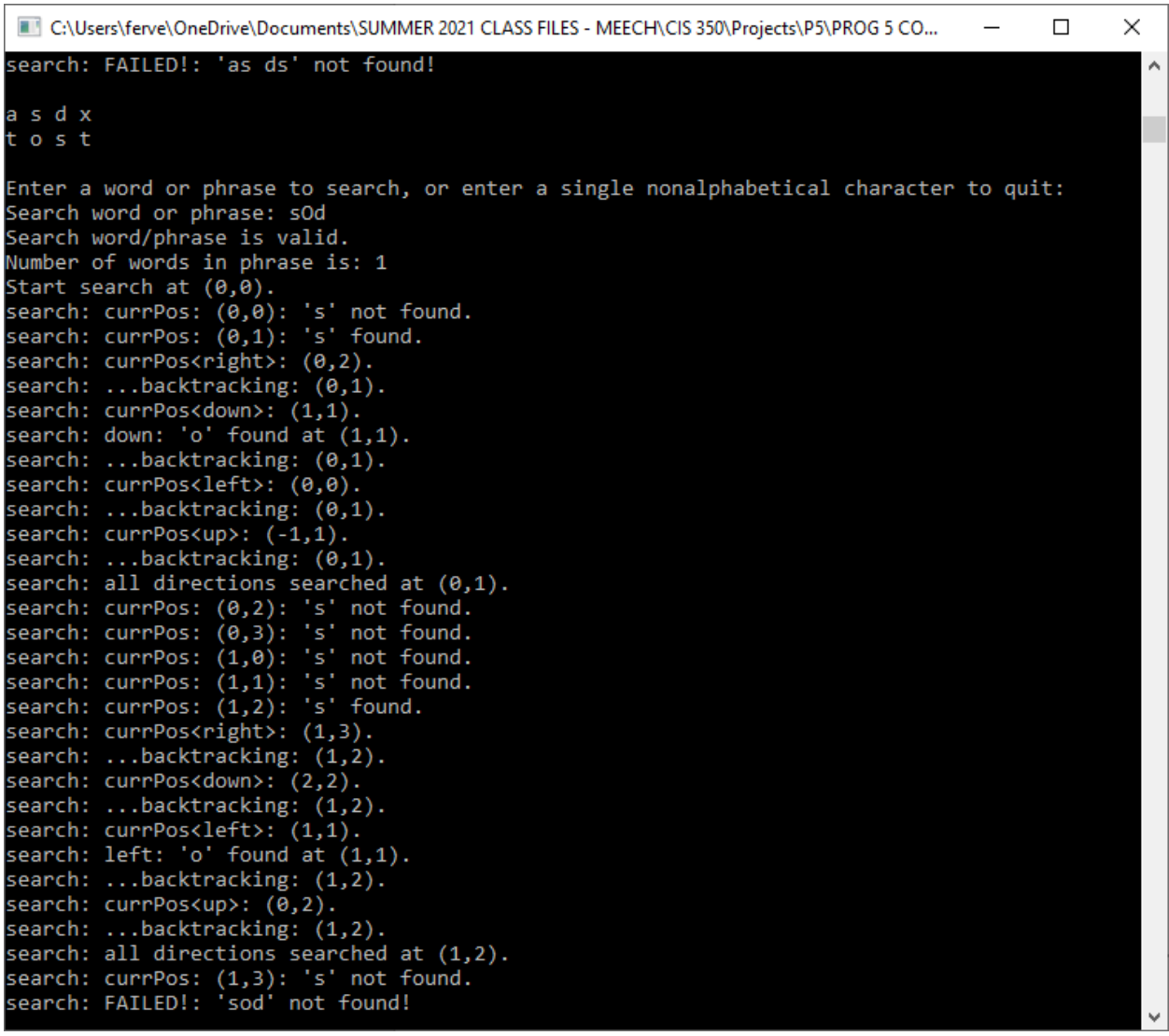

(search for “aT”)



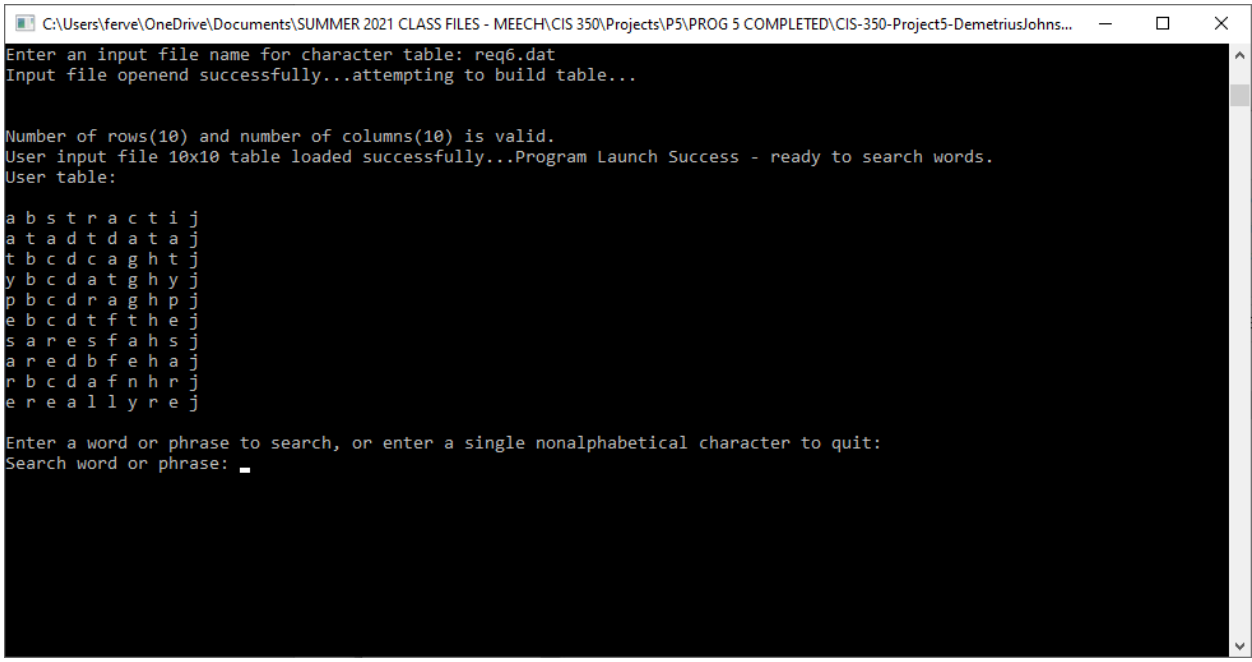

(search for “AS DS”)



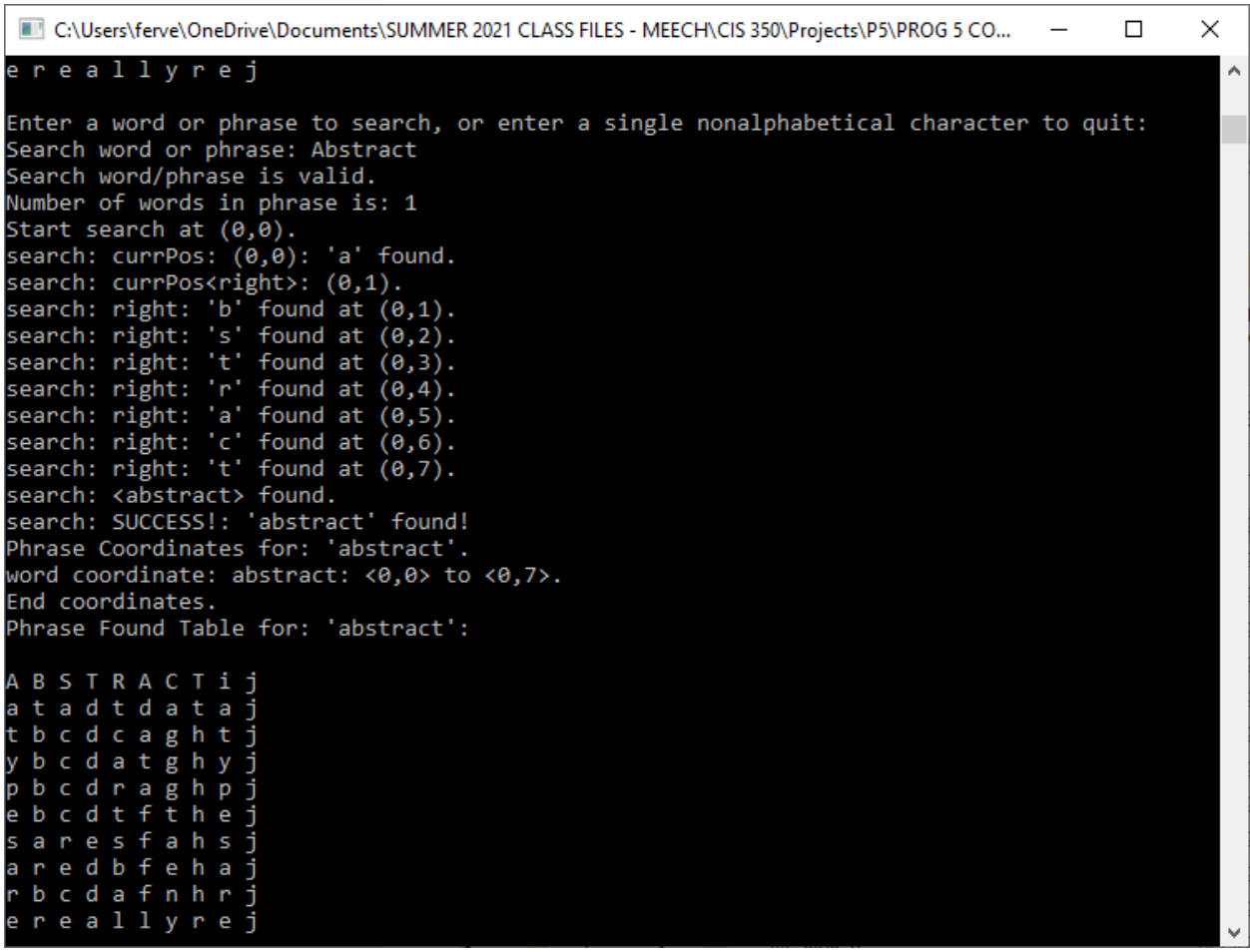

(search for “sOd”)



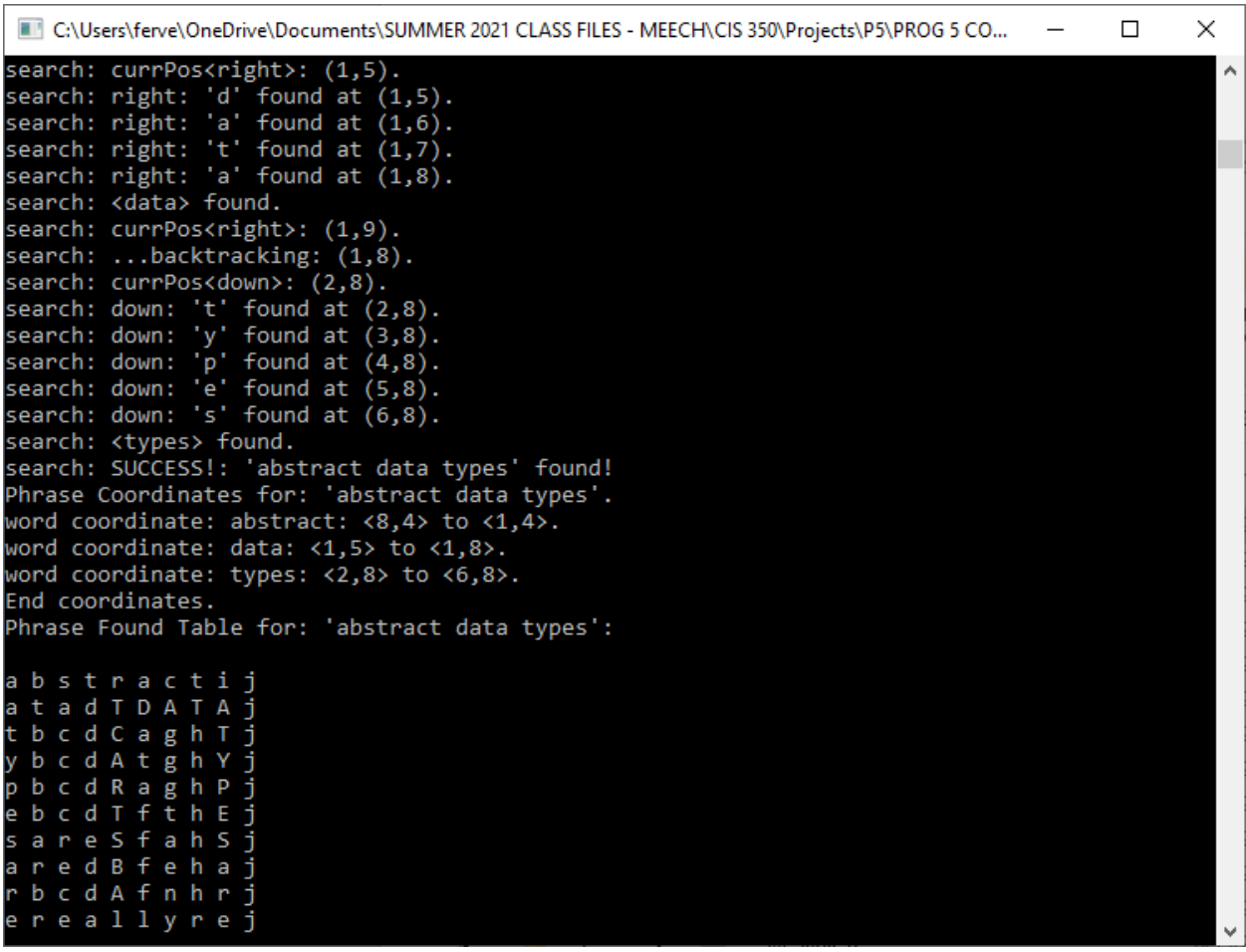

TEST 15: req6.dat – valid large 10x10 grid



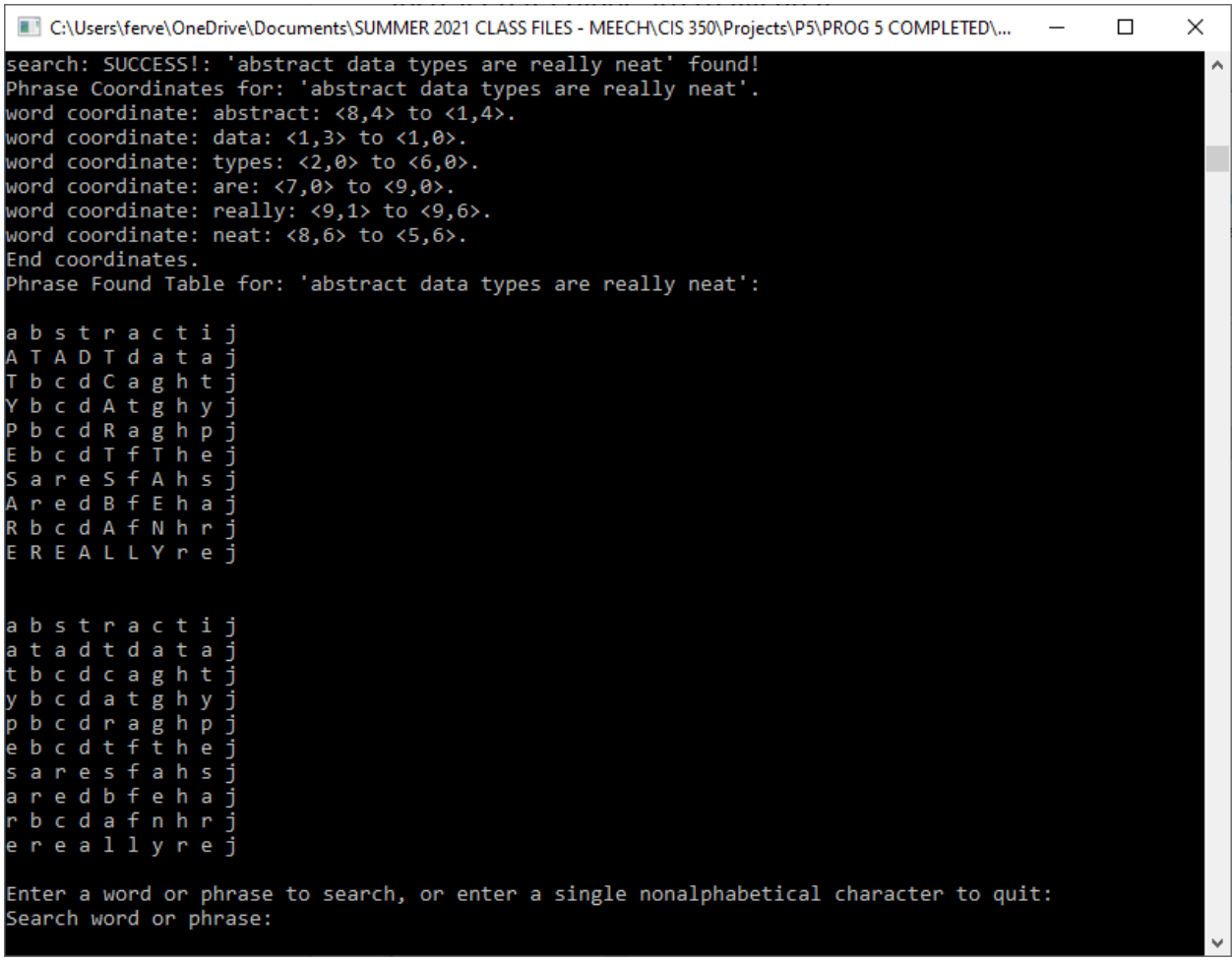

Abstract:



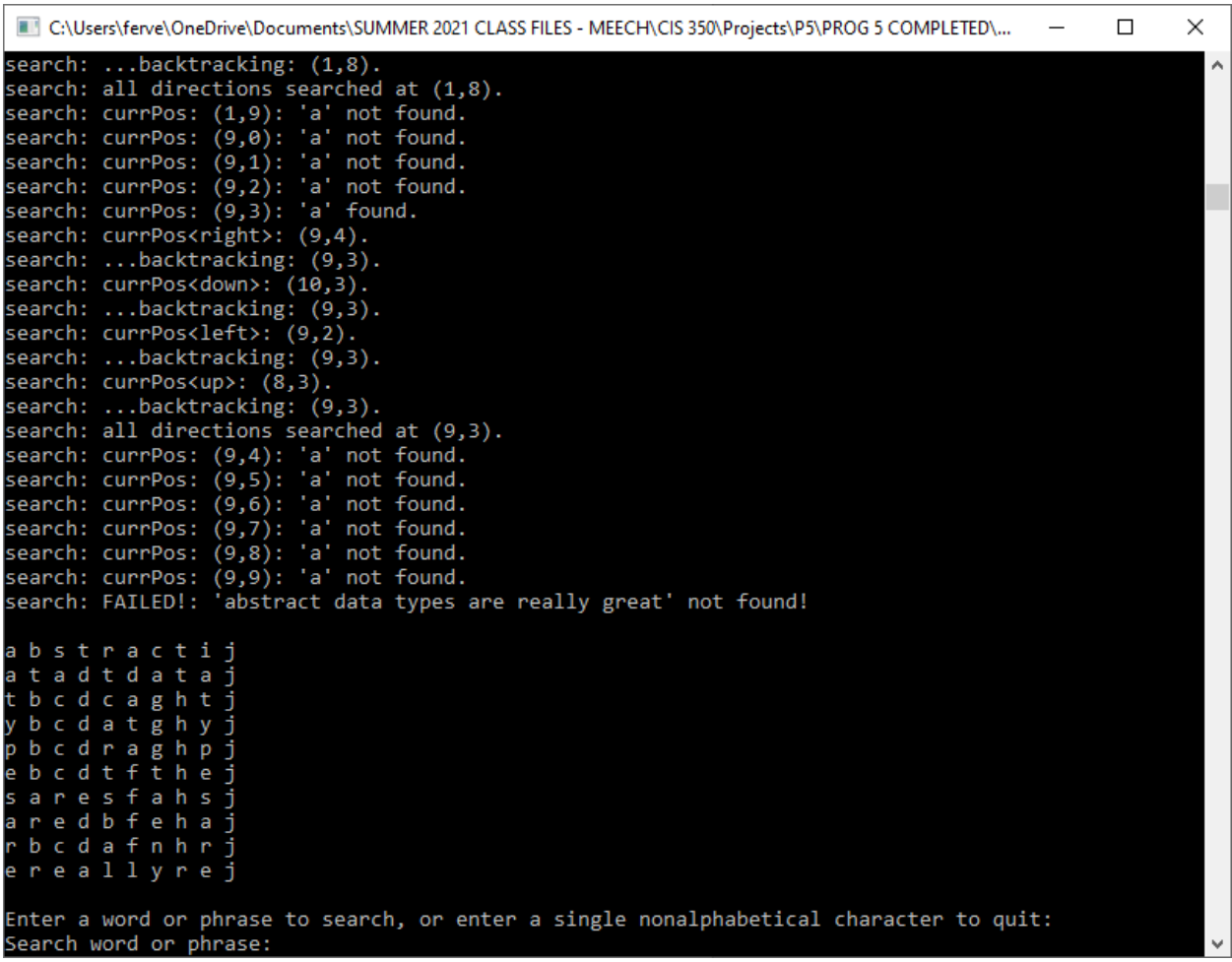

abstract data types:



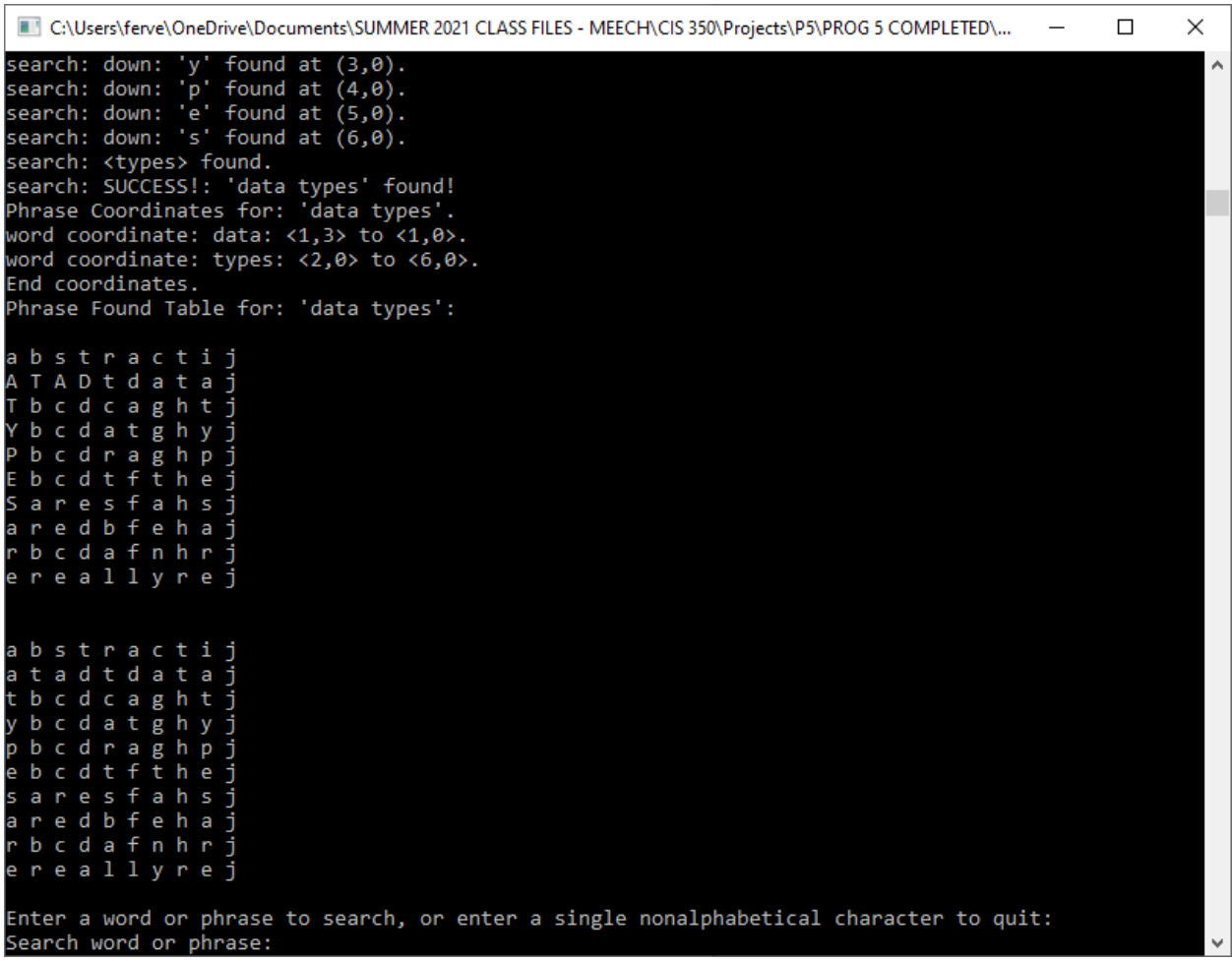

abstract data types are really neat:



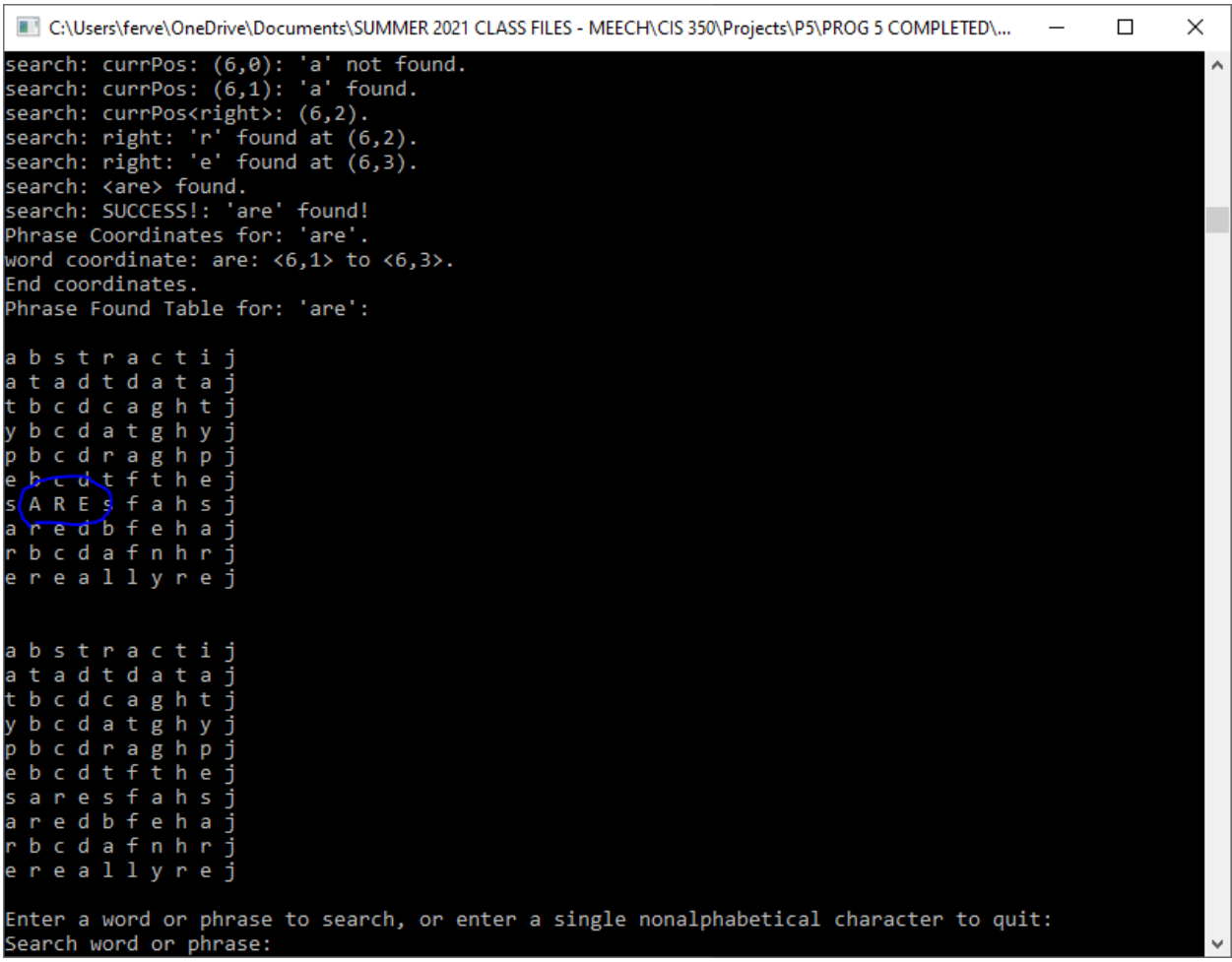

abstract data types are really great:


C:\Users\ferve\OneDrive\Documents\SUMMER 2021 CLASS FILES - MEECH\CIS 350\Projects\P5\PROG 5 COMPLETED\...
search: ...backtracking: (1,8).
search: all directions searched at (1,8).
search: currPos: (1,9): 'a' not found.
search: currPos: (9,0): 'a' not found.
search: currPos: (9,1): 'a' not found.
search: currPos: (9,2): 'a' not found.
search: currPos: (9,3): 'a' found.
search: currPos<right>: (9,4).
search: ...backtracking: (9,3).
search: currPos<down>: (10,3).
search: ...backtracking: (9,3).
search: currPos<left>: (9,2).
search: ...backtracking: (9,3).
search: currPos<up>: (8,3).
search: ...backtracking: (9,3).
search: all directions searched at (9,3).
search: currPos: (9,4): 'a' not found.
search: currPos: (9,5): 'a' not found.
search: currPos: (9,6): 'a' not found.
search: currPos: (9,7): 'a' not found.
search: currPos: (9,8): 'a' not found.
search: currPos: (9,9): 'a' not found.
search: FAILED!: 'abstract data types are really great' not found!
a b s t r a c t i j
a t a d t d a t a j
t b c d c a g h t j
y b c d a t g h y j
p b c d r a g h p j
e b c d t f t h e j
s a r e s f a h s j
a r e d b f e h a j
r b c d a f n h r j
e r e a l l y r e j
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase:

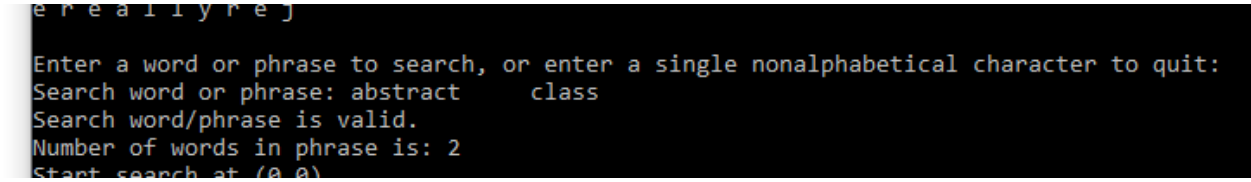
data types:

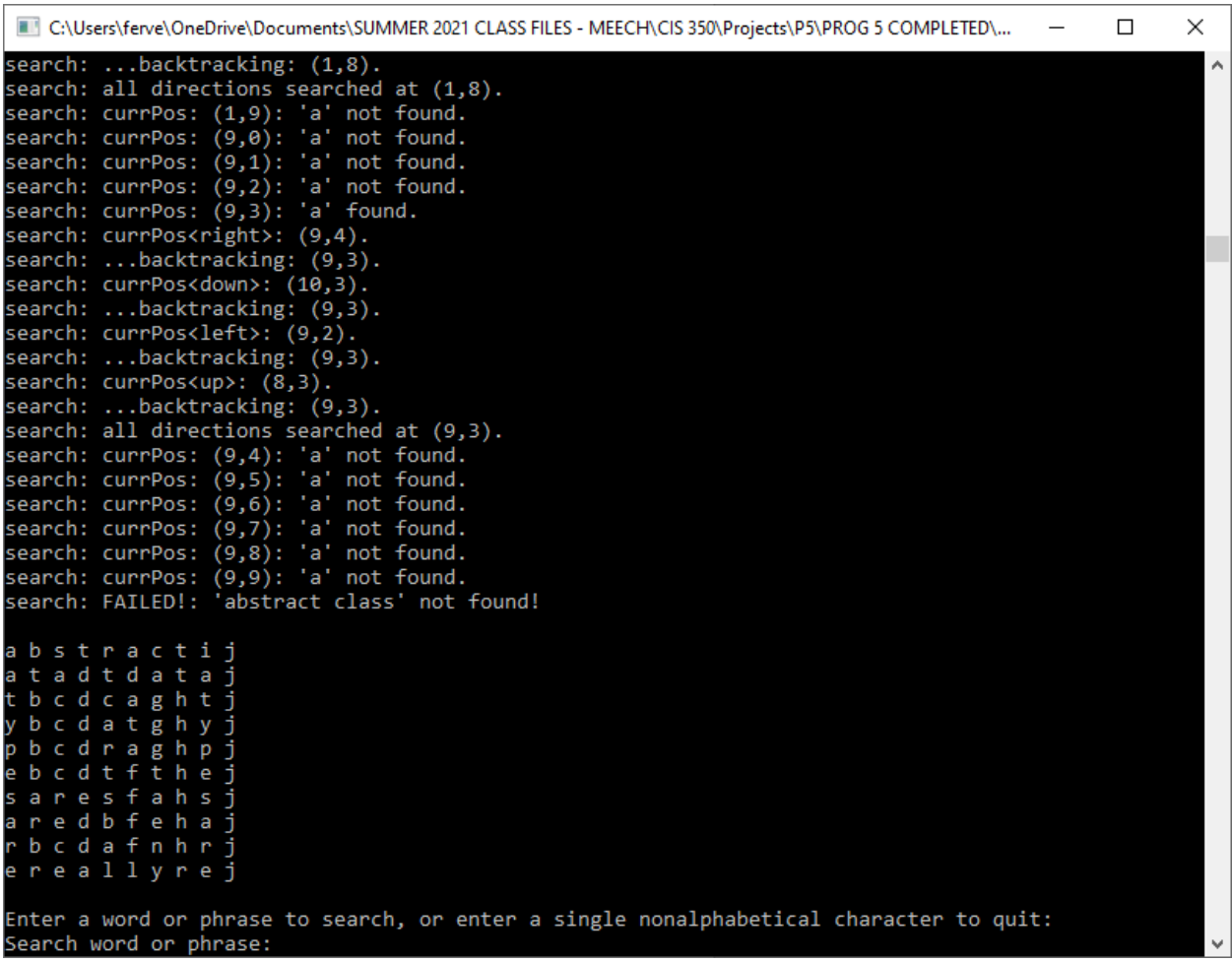

C:\Users\ferve\OneDrive\Documents\SUMMER 2021 CLASS FILES - MEECH\CIS 350\Projects\P5\PROG 5 COMPLETED\...
search: down: 'y' found at (3,0).
search: down: 'p' found at (4,0).
search: down: 'e' found at (5,0).
search: down: 's' found at (6,0).
search: <types> found.
search: SUCCESS!: 'data types' found!
Phrase Coordinates for: 'data types'.
word coordinate: data: <1,3> to <1,0>.
word coordinate: types: <2,0> to <6,0>.
End coordinates.
Phrase Found Table for: 'data types':
a b s t r a c t i j
A T A D t d a t a j
T b c d c a g h t j
Y b c d a t g h y j
P b c d r a g h p j
E b c d t f t h e j
S a r e s f a h s j
a r e d b f e h a j
r b c d a f n h r j
e r e a l l y r e j
a b s t r a c t i j
a t a d t d a t a j
t b c d c a g h t j
y b c d a t g h y j
p b c d r a g h p j
e b c d t f t h e j
s a r e s f a h s j
a r e d b f e h a j
r b c d a f n h r j
e r e a l l y r e j
Enter a word or phrase to search, or enter a single nonalphabetical character to quit:
Search word or phrase:

Are:




abstract<5 spaces>class <- test ignoring multiple spaces between words:






# 11. Error Log

Any issues you had while testing your code are recorded in the error log as you perform testing of the "completed" code – that is, when you run through all of the test cases in the test plan.

| Error Type | Cause of Error | Solution to Error |
|---|---|---|
| Log 2 types of errors:<br>Logic<br>Runtime | What specifically caused the error to occur | What did you do/change to fix the error |
| Major runtime error: could not find phrase "abstract data types are really neat" | I used if-else statements where I should have only used if statements; nesting with if-else caused my program to jump when it wasn't supposed to. | I had to spend an additional 4 hours because my program almost worked perfectly until that final test case; I had to make all the if statements independent of each other in my backtracking functions and figure out a way to |

| | | |
|---|---|---|
| | | delete old coordinates that needed to be removed because of back tracking |

Do not list any syntax errors or errors detected in unit testing as you build your program.

## 12. Status

Program is **100%** fully functional with no defects; it acts exactly as expected and up to specifications.