**CIS 350/3501**

**Summer 2021**

**Program 5**

**Due: August 10 and August 17, 2021**

**Purpose: Backtracking**

Recursion and in particular backtracking algorithms are a common problem to solve. In this assignment a recursive programming will be written using a backtracking algorithm to find phrases in a puzzle.

"Word search" are a common form of puzzle. These vary in format from puzzle to puzzle, but one common format is as follows: A user is presented with a two-dimensional grid of letters and a list of words. The user must find all of the words from the list within the grid, indicating where they are by drawing ovals around them. Words may be formed in any direction: up, down, left or right (diagonals will not be allowed for this program even though most puzzles do allow them) but all of the letters in a word must be in a straight line.

This idea can be extended to allow phrases to be embedded in the grids. However, requiring an entire phrase to be in a straight line on the board is not practical, as the dimensions of the board would need to be overly large. Therefore, we will still require the letters in each word to be in a straight line, but the program is allowed to change direction between words.

For example, we might be given the following 10x10 grid of letters:

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

and the following phrases:

abstract

abstract data types

abstract data types are really neat

abstract data types are really great

Assume the grid starts in the upper left corner with position (0,0), and that the coordinates are (row, column). Searching would yield:

“abstract” found in two places:

[(0,0) to (0,7)]

[(8,4) to (1,4)]

“abstract data types” found in two places:

[(8,4) to (1,4)], [(1,5) to (1,8)], [(2,8) to (6,8)]

[(8,4) to (1,4)], [(1,3) to (1,0)], [(2,0) to (6,0)]

“abstract data types are really neat” found at:

[(8,4) to (1,4)], [(1,3) to (1,0)], [(2,0) to (6,0)], [(7,0) to (9,0)], [(9,1) to (9,6)], [(8,6) to (5,6)]

“abstract data types are really great” not found

**Implementation**

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 phase.

**Input**

User entered file name for character grid – perform all file error checks.

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 and columns in the grid.

Remaining lines: number of rows lines each containing number of columns characters

For example:

2 3

asd

too

The format of the data will be correct but you need to edit the data for correct values (e.g. values on line one must be greater than zero, grid must contain letters – if these are incorrect print the incorrect value and error message and stop the program. Uppercase letters are to be converted to lowercase letters, warning message written but program continues.

Error: Invalid number of rows: -2. Program terminated.

Error: Invalid number of characters per row: 0. Program terminated.

Error: Invalid character in grid input: $ in position [0] [2]. Program terminated.

Warning: Uppercase letter in grid input: A in position [1] [1]. Converted to lower case.

The user input will be phrases of words, with a single space between each word. Ignore multiple space if they occur – no message – test case!

The program needs to edit the user entered data. Each phrase will be entered on a single line (assumption!).

If data contains non-alphabetic data

Error: Invalid word or phrase: ‘123’. Re-enter word/phrase.

Error: Invalid word or phrase: ‘a,d’. Re-enter word/phrase.

The user may enter either upper- or lower-case letters, but the string should be converted to lower case before searching the grid.

The program will end when the user requests to quit.

For example, valid user input could look like:

too

DO

at too

so DOO

[QUIT] <- whatever way you have user end program

**Output**

If a phrase is not found in the grid, the output should simply state that.

If a phrase is found in the grid, your program must find one occurrence of the phrase, and the output must indicate this fact in two ways:

* Show the coordinates of each word in the phrase as a pair of (row, column). The first (row, column) must be the starting location for the word and the second must be the ending location.
* Show the grid with the letters of the phrase indicated in upper case.

For example, for the 10x10 grid above and the phrase "abstract data types" your output would be:

abstract: (8,4) to (1,4)

data: (1,5) to (1,8)

types: (2,8) to (6,8)

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

**Algorithm**

The search algorithm must be a recursive, backtracking algorithm. Note that you do not need recursion to match the letters within individual words (although you may do this recursively if you prefer). Where the recursion is necessary is when moving from one word to the next, since it is here where you may change direction. To make the program more consistent the following requirements will be used for the recursive process:

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.

When given a choice of directions, the options must be tried in the following order:

* right
* down
* left
* up

Given this ordering and the grid above, the solution for "abstract data" would be [(8,4) to (1,4)], [(1,5) to (1,8)] rather than [(8,4) to (1,4)], [(1,3) to (1,0)], since the "right" direction is tried before the "left" direction.

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. For example, in the 10x10 grid shown above:

For the phrase "abstract data", this is not a valid solution: [(8,4) to (1,4)], [(1,5) to (4,5)]. This solution is not legal because we proceeded right from the "T" of "abstract" to find the "D" in "data", but then proceeded down to find the remaining letters in "data".

For the phrase "abstract data types are", this is not a valid solution: [(8,4) to (1,4)], [(1,3) to (1,0)], [(2,0) to (6,0)], [(7,0) to (7,2)]. This solution is not legal because we proceeded down from the "S" of "types" to find the "A" in "are", but then proceeded right to find the remaining letters in "are".

**Required Data Files/Example Output**

**[You must create an additional four test files]**

Req1.dat

2 3

asd

too

Please enter grid filename:

Req1.dat

Screen:

2 rows of 3 characters

Puzzle Layout

a s d

t o o

Please enter phrase (separate by single spaces):

do

Looking for: do

Phrase contains 1 words

The phrase: do

was found:

abstract: (0,2) to (1,2)

a s D

t o O

Please enter phrase (separate by single spaces):

sad

Looking for: sad

Phrase contains 1 words

The phrase: sad

was not found

then search for:

too

DO

at too

so DOO

output file:

Grid from: req1.dat

2 rows of 3 characters

Puzzle Layout

a s d

t o o

Looking for: do

Phrase contains 1 words

Search:

Start (0,0) look for ‘d’, not found

move right for ‘d’

‘d’ found at (0,2) move right for ‘o’ - not found

‘d’ found at (0,2) move down for ‘o’ - found

The phrase: do

was found:

a s D

t o O

Looking for: sad

Phrase contains 1 words

Search:

Start (0,0) look for ‘s’ - not found

look right for ‘s’

‘s’ found at (0,1) move right for ‘a’ - not found

‘s’ found at (0,1) move down for ‘a’ - not found

‘s’ found at (0,1) move left for ‘a’ - found

(note: “sad” must be left only as direction is set)

‘a’ found at (0,0) move left for ‘d’ - not found

BACKTRACK to ‘s’ at (0,1)

‘s’ found at (0,1) move up for ‘a’ - not found

ALL FOUR DIRECTIONS CHECKED – search for next ‘s’

NO MORE ‘s’ values

The phrase: sad

was not found.

req2.dat (invalid rows)

-2 3

asd

too

req3.dat (invalid columns)

2 -3

asd

too

req4.dat (valid with capital letters)

2 3

Asd

toO

search for:

“as”

“aS Do”

“sod”

req5.dat (valid with all capital letters)

2 4

ASDX

TOST

search for:

“aT”

“AS DS” <- not found! – ‘d’ to the right of ‘as’, cannot go down to ‘s’

“sOd”

req6.dat

10 10

abstractij

atadtdataj

tbcdcaghtj

ybcdatghyj

pbcdraghpj

ebcdtfthej

saresfahsj

aredbfehaj

rbcdafnhrj

ereallyrej

search for:

Abstract

abstract data types

abstract data types are really neat

abstract data types are really great

data types

are

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

Screen:

Please enter grid filename:

Req6.dat

10 rows of 10 characters

Puzzle Layout

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

Please enter phrase (separate by single spaces):

Abstract

Looking for: abstract

Phrase contains 1 words

The phrase: abstract

was found:

abstract: (0,0) to (0,7)

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

Please enter phrase (separate by single spaces):

abstract data types

Looking for: abstract data types

Phrase contains 3 words

The phrase: abstract data types

was found:

abstract: (8,4) to (1,4)

data: (1,5) to (1,8)

types: (2,8) to (6,8)

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

Please enter phrase (separate by single spaces):

abstract data types are really neat

Looking for: abstract data types are really neat

Phrase contains contains 6 words

The phrase: abstract data types are really neat

was found:

abstract: (8,4) to (1,4)

data: (1,3) to (1,0)

types: (2,0) to (6,0)

are: (7,0) to (9,0)

really: (9,1) to (9,6)

neat: (8,6) to (5,6)

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

Please enter phrase (separate by single spaces):

abstract data types are really great

Looking for: abstract data types are really great

Phrase contains 6 words

The phrase: abstract data types are really great

was not found

Please enter phrase (separate by single spaces):

data types

Looking for: data types

Phrase contains 2 words

The phrase: data types

was found:

data: (1,3) to (1,0)

types: (2,0) to (6,0)

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 l h e j

S a r e s f o h s j

a r e d b f o h a j

r b c d a f c h r j

e r e a l l y r e j

Please enter phrase (separate by single spaces):

Are

Looking for: are

Phrase contains 1 words

The phrase: are

was found:

are: (6,1) to (6,3)

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 l h e j

s A R E s f o h s j

a r e d b f o h a j

r b c d a f c h r j

e r e a l l y r e j

Please enter phrase (separate by single spaces):

abstract class

Looking for: abstract class

Phrase contains 2 words

The phrase: abstract class

was not found

Please enter phrase (separate by single spaces):

[QUIT]

Partial file output:

Looking for: abstract class

Phrase contains 2 words

Search:

Look for word 1: abstract

Start (0,0) look for ‘a’ - found

‘a’ found at (0,0) move right for ‘b’ - found

‘b’ found at (0,1) move right for ‘s’ – found

‘s’ found at (0,2) move right for ‘t’ – found

‘t’ found at (0,3) move right for ‘r’ – found

‘r’ found at (0,4) move right for ‘a’ – found

‘a’ found at (0,5) move right for ‘c’ – found

‘c’ found at (0,6) move right for ‘t’ – found

‘t’ found at (0,7) word 1 “abstract” found

Look for word 2: class

Start (0,7) look right for ‘c’ – not found

Start (0,7) look down for ‘c’ – not found

Start (0,7) look left for ‘c’ – not found

Start (0,7) look up for ‘c’ – not found

BACKTRACK to ‘a’ at (0,0)

‘a’ found at (0,0) move down for ‘b’ – not found

‘a’ found at (0,0) move left for ‘b’ – not found

‘a’ found at (0,0) move up for ‘b’ – not found

ALL FOUR DIRECTIONS CHECKED – search for next ‘a’

‘a’ found at (0,5) move right for ‘b’ – not found

‘a’ found at (0,5) move down for ‘b’ – not found

‘a’ found at (0,5) move left for ‘b’ – not found

‘a’ found at (0,5) move up for ‘b’ – not found

ALL FOUR DIRECTIONS CHECKED – search for next ‘a’

‘a’ found at (1,0) move right for ‘b’ – not found

‘a’ found at (1,0) move down for ‘b’ – not found

‘a’ found at (1,0) move left for ‘b’ – not found

‘a’ found at (1,0) move up for ‘b’ – not found

[repeated for all ‘a’s]

ALL FOUR DIRECTIONS CHECKED – search for next ‘a’

‘a’ found at (8,5) move right for ‘b’ – not found

‘a’ found at (8,5) move down for ‘b’ – not found

‘a’ found at (8,5) move left for ‘b’ – not found

‘a’ found at (8,5) move up for ‘b’ – found

‘b’ found at (7,5) move up for ‘s’ – found

‘s’ found at (6,5) move up for ‘t’ – found

‘t’ found at (5,5) move up for ‘r’ – found

‘r’ found at (4,5) move up for ‘a’ – found

‘a’ found at (3,5) move up for ‘c’ – found

‘c’ found at (2,5) move up for ‘t’ – found

‘t’ found at (1,5) word 1 “abstract” found

Look for word 2: class

Start (1,5) look right for ‘c’ – not found

Start (1,5) look down for ‘c’ – not found

Start (1,5) look left for ‘c’ – not found

Start (1,5) look up for ‘c’ – not found

BACKTRACK to ‘a’ at (8,5)

ALL FOUR DIRECTIONS CHECKED – search for next ‘a’

‘a’ found at (9,3) move right for ‘b’ – not found

‘a’ found at (9,3) move down for ‘b’ – not found

‘a’ found at (9,3) move left for ‘b’ – not found

‘a’ found at (9,3) move up for ‘b’ – not found

ALL FOUR DIRECTIONS CHECKED – search for next ‘a’

NO MORE ‘a’ values

The phrase: abstract class

was not found.

‘s’ found at (0,1) move down for ‘a’ - not found

‘s’ found at (0,1) move left for ‘a’ - found

(note: “sad” must be left only as direction is set)

‘a’ found at (0,0) move left for ‘d’ - not found

BACKTRACK to ‘s’

‘s’ found at (0,1) move up for ‘a’ - not found

ALL FOUR DIRECTIONS CHECKED – search for next ‘s’

NO MORE ‘s’ values

The phrase: sad

was not found.