CSCE 240 – Programming Assignment Three

Due: 11:59pm on Thursday, March 6

Purpose

Write, test, and use the four functions described below to read a text file containing a word search grid, display the grid, and find words within the grid.

Functions

Function 1 – ReadWordSearch

Implement the *ReadWordSearch* function. This function reads a text file containing the characters into a double-subscripted character array with kSize rows and kSize columns. kSize is a constant variable set in *word_search_functions.h*. This function should return true if the character array is successfully populated, and false if not. Read over the pre-condition and post-condition comments in *word_search_functions.h* for more details.

Function 2 – PrintWordSearch

Implement the *PrintWordSearch* function. This function takes a double-subscripted character array with kSize rows and kSize columns as an argument and outputs the characters at the standard output device (using cout) in a grid with a space between each column. Read over the pre-condition and post-condition comments in *word_search_functions.h* for more details.

Function 3 – FindWordRight

Implement the *FindWordRight* function. This function takes a double-subscripted character array with kSize rows and kSize columns, a string, and two integer references as arguments. The function will locate the first occurrence of the string in the array, written from left to right in a row in the array, and set the integer reference arguments to the starting position of the string. If the string appears in a row of the character array, the function returns true. If the string does not appear in a row in the array, the function returns false. Read over the pre-condition and post-condition comments in *word_search_functions.h* for more details.

Function 4 – FindWordLeft

Implement the *FindWordLeft* function. This function takes a double-subscripted character array with kSize rows and kSize columns, a string, and two integer references as arguments. The function will locate the first occurrence of the string in the array, written from right to left in a row in the array, and set the integer reference arguments to the starting position of the string. If the string appears in a row of the character array, the function returns true. If the string does not appear in a row in the array, the function returns false. Read over the pre-condition and post-condition comments in *word_search_functions.h* for more details.

Function 5 – FindWordDown

Implement the *FindWordDown* function. This function takes a double-subscripted character array with kSize rows and kSize columns, a string, and two integer references as arguments. The function will locate the first occurrence of the string in the array, written down in a column in the array, and set the integer reference arguments to the starting position of the string. If the string appears in a column of the character array, the function returns true. If the string does not appear in a column in the array, the function returns false. Read over the pre-condition and post-condition comments in *word_search_functions.h* for more details.

Function 6 – FindWordUp

Implement the *FindWordUp* function. This function takes a double-subscripted character array with kSize rows and kSize columns, a string, and two integer references as arguments. The function will locate the first occurrence of the string in the array, written up (from bottom to top) in a column in the array, and set the integer reference arguments to the starting position of the string. If the string appears in a column of the character array, the function returns true. If the string does not appear in a column in the array, the function returns false. Read over the pre-condition and post-condition comments in *word_search_functions.h* for more details.

Function 7 – FindWordDiagonal

Implement the *FindWordDiagonal* function. This function takes a double-subscripted character array with kSize rows and kSize columns, a string, and two integer references as arguments. The function will locate the first occurrence of the string in the array, written diagonally from left to right in the array, and set the integer reference arguments to the starting position of the string. If the string appears on a diagonal in the character array, the function returns true. If the string does not appear on a diagonal in the array, the function returns false. Read over the pre-condition and post-condition comments in *word_search_functions.h* for more details.

Example

For the grid below:

```
e c n e i c t p r o
g r a m r g m a u z
u c l m n o n a e b
i a c w s i g r x t
t v s t l t n y y u
a r e o b x i e i o
r b r h b k t m n w
h a l w l b e a l s
c a w e c n e i c s
p r o g r a m q x z
```

FindWordRight should find “program” at position 9,0

FindWordLeft should find “science” at position 8,9

FindWordDown should find “guitar” at position 1,0

FindWordUp should find “meeting” at position 9,6

FindWordDiagonal should find “nine” at position 2,4

Specifications

- All output should be directed to the standard output device using cout.
- The prototypes for the *FindWordRight*, *FindWordLeft*, *FindWordDown*, *FindWordUp*, and *FindWordDiagonal* functions are included in *word_search_functions.h*. Do not change the contents of the header file, except to modify the constant kSize, when testing your functions for different size word search grids. You must implement these functions in *word_search_functions.cc,* and *word_search_functions.cc* must include *word_search_functions.h*
- You will submit *word_search_functions.cc* to the assignment in Blackboard.
- Source files must compile and run on a computer of the instructor’s choosing in the Linux lab (see your course syllabus for additional details).

Testing

A sample text file, *grid.txt*, containing the example word search grid shown above, a sample driver to test your functions *program3.cc*, and a makefile have been provided. You should ensure that your functions find the words in the positions provided in the examples. You should search for words that do not appear in the grid to ensure that the words are not found and your functions do not crash. You are encouraged to test your functions with additional grids and search words.

Your functions will be graded using the included *program3.cc* with modifications to kSize and the *grid.txt* input file.

Grade Breakdown

Style: 1 point

Documentation: 1 point

Clean compilation: 1 point

ReadWordSearch passes instructor’s tests: 1 point

PrintWordSearch passes instructor’s tests: 1 point

FindWordRight passes instructor’s tests: 1 point

FindWordLeft passes instructor’s tests: 1 point

FindWordDown passes instructor’s tests: 1 point

FindWordUp passes instructor’s tests: 1 point

FindWordDiagonal passes instructor’s tests: 1 point

The penalty for late program submissions is 10% per day, with no program being accepted after 3 days.