CS 201

Homework Assignment 0

NOTE: This assignment will not be graded.

In this homework, you will implement a system for solving word puzzles. This system will read a word puzzle, consisting of hidden words, from the input file and search the words provided by the user in that puzzle. The search should be conducted in 8 directions:

1. left-to-right (→)
2. right-to-left (←)
3. top-to-bottom (↓)
4. bottom-to-top (↑)
5. top-left-to-bottom-right (↘)
6. top-right-to-bottom-left (↙)
7. bottom-left-to-top-right (↗)
8. bottom-right-to-top-left (↖)

As an example, suppose that you are given the word puzzle below and you are asked to search the following words in that puzzle: {CIGDEMGUNDUZ, AYNURDAYANIK, BURAKTOSUN, ATATURK, DEMIR, FATIHISLER, DOGAN, CELALCIGIR, SELIMAKSOY}. Your program is supposed to locate the first eight words in that puzzle and to figure out that the last word (SELIMAKSOY) is not in that puzzle.

H	D	Y	X	L	O	Y	K	S	F	H	K	L	G	E
G	H	J	S	R	R	U	O	Z	D	S	W	H	H	G
A	C	I	G	D	E	M	G	U	N	D	U	Z	T	N
D	Y	A	E	A	G	L	G	J	N	E	L	O	S	U
F	G	N	Q	V	D	H	S	M	S	D	B	Z	M	S
T	H	U	U	O	G	Q	O	I	S	N	D	A	F	O
A	J	D	G	R	H	O	Y	C	H	G	H	K	S	T
H	K	A	A	C	D	A	B	G	C	I	R	X	H	K
T	N	I	M	L	E	A	O	Y	S	U	T	Y	R	A
S	J	D	E	O	M	D	Y	Y	T	O	Y	A	Y	R
I	S	R	I	G	I	C	L	A	L	E	C	C	F	U
O	F	E	G	Y	R	D	T	H	N	S	H	U	F	B
H	H	Y	N	T	E	A	D	S	O	I	A	Y	H	B
W	C	I	T	S	A	T	N	A	F	H	K	G	F	F
N	D	A	W	Q	F	G	G	R	S	Q	A	G	U	Z

In this assignment, for solving such puzzles, you will implement a class called "`WordPuzzle`". Each `WordPuzzle` object stores the characters of a puzzle in a fixed 2-D array of characters. The class will have the following functionalities; the details of these functionalities are given below:

1. Load the puzzle from an input file
2. Search a given word in the puzzle
3. Search a list of words in the puzzle
4. Display the solution on the screen

Load puzzle from an input file: The system will load a puzzle from an input file at its declaration. So you will write a constructor for the `WordPuzzle` class. This constructor will take the name of the input file, read the content of the file, and initialize the puzzle from the file. If there is no such file with the specified file name, your system should give a warning message. In this function, you may assume that the size of the `puzzle` array, which is a data member of the `WordPuzzle` class, is consistent with the file contents (e.g., if the size of `puzzle` is 15x20, you may assume that there are 15 lines in the file and each line consists of 20 characters excluding the white characters at its end; you may also assume that there are no white space characters in between these 20 characters).

Search a given word in the puzzle: The system will allow the user to search a word in the puzzle. For that, it uses a main search function. This main search function includes calling 8 different functions, each of which corresponds to a search operation in one of the following directions: left-to-right (→), right-to-left (←), top-to-bottom (↓), bottom-to-top (↑), top-left-to-bottom-right (↘), top-right-to-bottom-left (↙), bottom-left-to-top-right (↗), bottom-right-to-top-left (↖). Each of these 8 functions should return whether or not the given word is found in the puzzle. If the word is found, they should also return the row and column numbers at which the word starts. The prototypes of these functions are provided in the class definition below.

After calling these 8 functions, the main search function should return one of the enumeration values, indicating the direction in which the word is found, specified in the class definition below. It should also return the starting row and column numbers. For instance, in the puzzle example given above, for CIGDEMGUNDUZ, the main search function returns `LEFT_TO_RIGHT` (direction), 2 (starting row number), and 1 (starting column number); for AYNURDAYANIK, it returns `TOP_LEFT_TO_BOTTOM_RIGHT`, 2, and 0; for FATIHISLER, it returns `BOTTOM_RIGHT_TO_TOP_LEFT`, 10, and 13; and for SELIMAKSOY, it returns `NONE`, -1, and -1. Note that this main search function returns these values but does not display anything on the screen. In order to see the results, you should use the `displaySolution` function.

Search a list of words in the puzzle: The system will also allow the user to search a list of words in the puzzle. The word list is given in another file. For that, the second main search function in the system will take the name of the word list file, read the word list from that file, conduct a search operation for each word, and write the results in a new output file. If there is no such file with the specified file name, the system should give a warning message.

In the word list input file, each word is given in a separate line. The result of each word should be written in a separate line and as a tab-separated list into the output file. For example, if the word list input file contains

```
CIGDEMGUNDUZ
AYNURDAYANIK
FATIHISLER
SELIMAKSOY
```

the output file will contain

```
LEFT_TO_RIGHT	2	1
TOP_LEFT_TO_BOTTOM_RIGHT	2	0
BOTTOM_RIGHT_TO_TOP_LEFT	10	13
NONE	-1	-1
```

Display the solution on the screen: The system will allow the user to see the result of search operation on the screen. This function will take a word and its search results (direction, starting row number, and starting column number) and write the characters of a given word one by one (separated with tabs) on the screen together with their corresponding index values. So for each word, you will see 3 lines on the screen: the first line contains the characters of the word, the second line contains the row number of each character, and the third line contains the column number of each character. For the words that do not exist in the puzzle, the system will just give a message on the screen. For example,

for CIGDEMGUNDUZ, you will see

```
C     I     G     D     E     M     G     U     N     D     U     Z
2     2     2     2     2     2     2     2     2     2     2     2
1     2     3     4     5     6     7     8     9     10    11    12
```

for AYNURDAYANIK, you will see

```
A     Y     N     U     R     D     A     Y     A     N     I     K
2     3     4     5     6     7     8     9     10    11    12    13
0     1     2     3     4     5     6     7     8     9     10    11
```

for FATIHISLER, you will see

```
F     A     T     I     H     I     S     L     E     R
10    9     8     7     6     5     4     3     2     1
13    12    11    10    9     8     7     6     5     4
```

and for SELIMAKSOY, you will see
SELIMAKSOY does not exist in the puzzle.

Below is the required public part of the `WordPuzzle` class that you must write in this assignment. The name of the class **must** be **`WordPuzzle`**. The interface for the class must be written in a file called **“`WordPuzzle.h`”** and its implementation must be written in a file called **“`WordPuzzle.cpp`”**. In this assignment, we provide you with the interface file (“`WordPuzzle.h`”). You are not allowed modifying this file. For example, you should not define additional public and private member functions and data members in this class. In other words, the “`WordPuzzle.h`” should remain the same and you must use the “`WordPuzzle.h`” file as it is.

```
// This is the header file of a WordPuzzle solver system. In this header
// file, you will find the definition of WordPuzzle class.
//
// Note that you ARE NOT ALLOWED MODIFYING this class definition.
// Otherwise you will lose a considerable amount of grade even though
// your implementation is totally correct.


#include <string>       // For string declarations and operations
using namespace std;

class WordPuzzle{
public:
     const static int numOfRow = 15;
     const static int numOfColumn = 15;

     // There are 8 directions for which the search operation is done
     enum Direction{     NONE = 0, LEFT_TO_RIGHT, RIGHT_TO_LEFT, TOP_TO_BOTTOM,
                         BOTTOM_TO_TOP, TOP_LEFT_TO_BOTTOM_RIGHT,
                         TOP_RIGHT_TO_BOTTOM_LEFT, BOTTOM_LEFT_TO_TOP_RIGHT,
                         BOTTOM_RIGHT_TO_TOP_LEFT};


     // The constructor takes the puzzle file name as a parameter, reads the
     // content of the file, and initializes the puzzle from that content
     WordPuzzle(string puzzleFileName);


     // This function takes a word as a parameter and searches it from
     // left to right in the puzzle. If it finds the word in the puzzle,
     // it returns true and the numbers of the row and column at which
     // the word starts. Otherwise, it returns false.
     bool searchLeftToRight(string word, int& row, int& column);
```

```
// This function takes a word as a parameter and searches it from
// right to left in the puzzle. If it finds the word in the puzzle,
// it returns true and the numbers of the row and column at which
// the word starts. Otherwise, it returns false.
bool searchRightToLeft(string word, int& row, int& column);

// This function takes a word as a parameter and searches it from
// top to bottom in the puzzle. If it finds the word in the puzzle,
// it returns true and the numbers of the row and column at which
// the word starts. Otherwise, it returns false.
bool searchTopToBottom(string word, int& row, int& column);

// This function takes a word as a parameter and searches it from
// bottom to top in the puzzle. If it finds the word in the puzzle,
// it returns true and the numbers of the row and column at which
// the word starts. Otherwise, it returns false.
bool searchBottomToTop(string word, int& row, int& column);

// This function takes a word as a parameter and searches it from
// top-left to bottom-right (diagonally) in the puzzle. If it finds
// the word in the puzzle, it returns true and the numbers of the row
// and column at which the word starts. Otherwise, it returns false.
bool searchTopLeftToBottomRight(string word, int& row, int& column);

// This function takes a word as a parameter and searches it from
// top-right to bottom-left (diagonally) in the puzzle. If it finds
// the word in the puzzle, it returns true and the numbers of the row
// and column at which the word starts. Otherwise, it returns false.
bool searchTopRightToBottomLeft(string word, int& row, int& column);

// This function takes a word as a parameter and searches it from
// bottom-left to top-right (diagonally) in the puzzle. If it finds
// the word in the puzzle, it returns true and the numbers of the row
// and column at which the word starts. Otherwise, it returns false.
bool searchBottomLeftToTopRight(string word, int& row, int& column);


// This function takes a word as a parameter and searches it from
// bottom-right to top-left (diagonally) in the puzzle. If it finds
// the word in the puzzle, it returns true and the numbers of the row
// and column at which the word starts. Otherwise, it returns false.
bool searchBottomRightToTopLeft(string word, int& row, int& column);


// This function takes a word as a parameter, calls 8 different
// functions, defined above, each of which searches in a particular
// direction. If it finds the word, it returns the direction in which it
// finds the word and the numbers of the row and column at which the word
// starts. If it does not find the word, it returns NONE.
Direction solvePuzzle(string word, int& row, int& column);


// This function takes two parameters (the names of two files). It
// reads a list of words from the first file, searches these words
// in the puzzle and writes the search results in a new output file,
// whose name is provided as the second parameter. Please see the
// homework description for the format of the results.
void solvePuzzle(string wordListFileName, string outputFileName);
```

```
     // This function takes a word and its search results (direction,
     // starting row number, and starting column number) and writes the
     // solution on the screen. The solution contains 3 lines: the first
     // line contains the characters of the word, the second line contains
     // the row number of each character, and the third line contains
     // the column number of each character. For the words that do not
     // exist in the puzzle, the system will just give a message on
     // the screen. Please see the homework description for the detailed
     // format of the solution.
     void displaySolution(string word, Direction dir, int row, int column);


private:
     char puzzle[numOfRow][numOfColumn];

};
```

NOTES ABOUT IMPLEMENTATION:

1. You are provided with sample input files (“puzzle.txt” and “wordlist.txt”). For the “puzzle.txt” input file, the static data members `numOfRow = 15` and `numOfColumn = 15`, as given in the header file. Please be aware that one may change the input files so that they contain different number of lines and characters. However, in that case, one will assign the corresponding values to `numOfRow` and `numOfColumn` data members. Therefore, it is very IMPORTANT FOR YOU to use these constants in your program instead of their exact values (for instance, do not use 15 instead of using `numOfRow` constant).
2. The input files we provide are just sample inputs. Test your program with different input files as well.
3. You ARE NOT ALLOWED modifying the header file provided you with this assignment. As a part of this, your code MUST use a 2-D character array with a fixed size. Do not use dynamic arrays or data structures such as `vector` from the standard library. Similarly, you should keep the words as string variables.
4. You may assume that all characters in the puzzle and the words are upper case letters. Also, you may assume that hidden words are unique.
5. You can use the following code segment to open a text file, read data from that file, and close the file when all data are read.

```
//Declare variables
string filename = "puzzle.txt";
string word;
char ch;
...
//Open a stream for the input file
ifstream inputFile;
inputFile.open( filename.c_str(), ios_base::in );
...
//Read a string from the file stream just like reading it from cin
inputFile >> word;
//Read a character from the file stream just like reading it from cin
inputFile >> ch;
...
//Close the input file stream
inputFile.close();
```

Similarly, you can use the following code segment to open a text file, write data into that file, and close the file when you are done.

```
//Declare variables
string filename = "res.txt";
char ch;
int d;
...
//Open a stream for the output file
ofstream outputFile;
outputFile.open( filename.c_str(), ios_base::out );
...
//Write values into the file stream just like displaying them on the screen
outputFile << ch << “\t” << d << endl;
...
//Close the output file stream
outputFile.close();
```

EXAMPLE TEST CODE:

```
#include "WordPuzzle.h"
#include <iostream>
using namespace std;

int main(){

    WordPuzzle::Direction D;
    int x, y;

    WordPuzzle P1("my_puzzle.txt");

    WordPuzzle P2("puzzle.txt");
    P2.solvePuzzle("my_wordlist.txt","my_res.txt");
    P2.solvePuzzle("wordlist.txt","res.txt");

    D = P2.solvePuzzle("CIGDEMGUNDUZ",x,y);
    P2.displaySolution("CIGDEMGUNDUZ",D,x,y);
    cout << endl;

    D = P2.solvePuzzle("AYNURDAYANIK",x,y);
    P2.displaySolution("AYNURDAYANIK",D,x,y);
    cout << endl;

    D = P2.solvePuzzle("CIGDEMDEMIR",x,y);
    P2.displaySolution("CIGDEMDEMIR",D,x,y);
    cout << endl;

    D = P2.solvePuzzle("FANTASTIC",x,y);
    P2.displaySolution("FANTASTIC",D,x,y);
    cout << endl;

    D = P2.solvePuzzle("SELIMAKSOY",x,y);
    P2.displaySolution("SELIMAKSOY",D,x,y);
    cout << endl;

    D = P2.solvePuzzle("OSCAR",x,y);
    P2.displaySolution("OSCAR",D,x,y);
    cout << endl;

    D = P2.solvePuzzle("WHOISTHAT",x,y);
    P2.displaySolution("WHOISTHAT",D,x,y);
    cout << endl;
    D = P2.solvePuzzle("MOODLE",x,y);
    P2.displaySolution("MOODLE",D,x,y);
    cout << endl;

    D = P2.solvePuzzle("FATIHISLER",x,y);
    P2.displaySolution("FATIHISLER",D,x,y);
    cout << endl;

    D = P2.solvePuzzle("GAME",x,y);
    P2.displaySolution("GAME",D,x,y);
    cout << endl;
    return 0;
}
```

OUTPUT OF THE EXAMPLE TEST CODE:

This test code uses the "`puzzle.txt`" and "`wordlist.txt`" sample input files. You can download them from the Moodle website of this course. For this test code, you will see the following output on the screen. Besides, this test code creates an output file whose name is "`res.txt`". You can see the contents of this output file also downloading it from the course website.

```
ERROR: my_puzzle.txt does not exist ...
ERROR: my_wordlist.txt does not exist ...
C      I      G      D      E      M      G      U      N      D      U      Z
2      2      2      2      2      2      2      2      2      2      2      2
1      2      3      4      5      6      7      8      9      10     11     12

A      Y      N      U      R      D      A      Y      A      N      I      K
2      3      4      5      6      7      8      9      10     11     12     13
0      1      2      3      4      5      6      7      8      9      10     11

CIGDEMDEMIR does not exist in the puzzle

F      A      N      T      A      S      T      I      C
13     13     13     13     13     13     13     13     13
9      8      7      6      5      4      3      2      1

SELIMAKSOY does not exist in the puzzle

O      S      C      A      R
12     11     10     9      8
9      10     11     12     13

W      H      O      I      S      T      H      A      T
13     12     11     10     9      8      7      6      5
0      0      0      0      0      0      0      0      0

M      O      O      D      L      E
4      5      6      7      8      9
8      7      6      5      4      3

F      A      T      I      H      I      S      L      E      R
10     9      8      7      6      5      4      3      2      1
13     12     11     10     9      8      7      6      5      4

G      A      M      E
6      7      8      9
3      3      3      3
```

NOTES:

1. **<u>This assignment will not be graded. This homework is intended solely for practice.</u>**
2. In this assignment, you must have separate interface and implementation files (i.e., separate .h and .cpp files) for your class. You must use the "`WordPuzzle.h`" file that we provide as your interface file. This file can be downloaded from the course website.

   You may test your implementation by using the "`WordPuzzle.h`" file that we provide and writing our own "`driver.cpp`" file. For this reason, your implementation file name MUST BE "`WordPuzzle.cpp`" and it SHOULD NOT include the class header. We also recommend you to write your own driver file to test each of your functions.
3. You are free to write your programs in any environment (you may use either Linux or Windows). On the other hand, we encourage you to test your programs on "`dijkstra.ug.bcc.bilkent.edu.tr`" and because we will expect your future homework programs to compile and run on the `dijkstra` machine.
4. Do not forget to download the necessary files ("`hw1_files.zip`") for this assignment. The zip file includes the interface file ("`WordPuzzle.h`"), the sample driver file ("`driver.cpp`"), the sample input files ("`puzzle.txt`" and "`wordlist.txt`"), and the corresponding output file ("`res.txt`").