CS 121 Homework 6

John Ingram

11/27/19

**Problem one**

Given the word find puzzle below locate and print out all four letter words.

**Code:**

//

// This is an adapted version of the instructor provided "find\_word" program

// Modified By John Ingram

//

#include <iostream>

#include <fstream>

#include <iomanip>

#include <string>

using namespace std;

string reverseit(string line);

string getVertical(unsigned int col);

int main(void)

{

string horizontal\_word = "";

string reversed\_horizontal\_word = "";

string vertical\_word = "";

string reversed\_vertical\_word = "";

string puzzle\_line = "";

string puzzle\_low = "";

string dictionary\_word = "";

unsigned int j, k;

char ch;

bool found = false;

ifstream dictionary;

ifstream puzzle;

puzzle.open("./Puzzle.txt");

if (!puzzle)

{

cout << "Unable to open puzzle file" << endl;

system("pause");

return 1;

}

else

{

cout << "Puzzle file opened" << endl;

}

dictionary.open("./Dict.txt");

if (!dictionary)

{

cout << "Unable to open dictionary file" << endl;

system("pause");

return 1;

}

else

{

cout << "Dictionary file opened" << endl << endl;

}

// Get a line from the puzzle

while (!puzzle.eof())

{

puzzle >> puzzle\_line;

cout << endl << puzzle\_line << endl;

// Make sure it is lowercase.

puzzle\_low = "";

for (j = 0; j < puzzle\_line.length(); j++)

{

ch = tolower(puzzle\_line.at(j));

puzzle\_low = puzzle\_low + ch;

}

// Extract 4 characters

for (k = 0; k < puzzle\_low.length() - 3; k++)

{

horizontal\_word = puzzle\_low.substr(k,4);

// Make a reverse copy of the test\_word

reversed\_horizontal\_word = reverseit(horizontal\_word);

// Now go through the whole dictionary to see if these are words

dictionary >> dictionary\_word; // Get one word from the dictionary

while(!dictionary.eof())

{

if (dictionary\_word == horizontal\_word)

{

cout << horizontal\_word << " is a word" << endl;

}

if (dictionary\_word == reversed\_horizontal\_word)

{

cout << reversed\_horizontal\_word << " is a word" << endl;

}

dictionary >> dictionary\_word; // Get next word from the dictionary

} // End dictionary search loop

// Reset the dictionary file to test the next words.

dictionary.clear();

dictionary.seekg(0, ios::beg);

} // End loop for processing one line of the puzzle

} // End loop for testing each line of the puzzle

//

//

// Test columns

//

for(unsigned int i = 0; i < 10; i++)

{

puzzle\_line = getVertical(i);

cout << endl << puzzle\_line << endl;

// Make sure it is lowercase.

puzzle\_low = "";

for (j = 0; j < puzzle\_line.length(); j++)

{

ch = tolower(puzzle\_line.at(j));

puzzle\_low = puzzle\_low + ch;

}

// Extract 4 characters

for (k = 0; k < puzzle\_low.length() - 3; k++)

{

horizontal\_word = puzzle\_low.substr(k,4);

// Make a reverse copy of the test\_word

reversed\_horizontal\_word = reverseit(horizontal\_word);

// Now go through the whole dictionary to see if these are words

dictionary >> dictionary\_word; // Get one word from the dictionary

while(!dictionary.eof())

{

if (dictionary\_word == horizontal\_word)

{

cout << horizontal\_word << " is a word" << endl;

}

if (dictionary\_word == reversed\_horizontal\_word)

{

cout << reversed\_horizontal\_word << " is a word" << endl;

}

dictionary >> dictionary\_word; // Get next word from the dictionary

} // End dictionary search loop

// Reset the dictionary file to test the next words.

dictionary.clear();

dictionary.seekg(0, ios::beg);

} // End loop for processing one line of the puzzle

} // End loop for testing each line of the puzzle

cout << endl << endl << endl;

return 0;

}

//

// Function Reversit

//

string reverseit(string line)

{

string invert\_line = "";

int position = 0;

for (position = line.length() - 1; position >= 0; position--)

invert\_line += line.at(position);

return invert\_line;

}

string getVertical(unsigned int col)

{

ifstream puzzle;

puzzle.open("./Puzzle.txt");

string result = "";

string line = "";

while(!puzzle.eof())

{

puzzle >> line;

result += line[col];

}

puzzle.clear();

puzzle.seekg(0, ios::beg);

return result;

}

Output:

**Problem Two**

* Create a dynamic two dimensional square array of unsigned integers (array\_one). Prompt the user to enter the number of rows (maximum of 50) (Use this for columns too since the array will be square.)
* Pass the array to a function that will initialize the two dimensional array to random numbers between 0 and 4000 using the rand() library function. Here is the kicker: The array cannot have any repeated values!
* Create another dynamic two dimensional array of the same size (array\_transpose)
* Pass both arrays to a function that will generate the transpose of array\_one returning the values in array\_transpose. The transpose swaps the rows and columns of an array. Suppose the square array is a 4 by 4 integer numbers.