CS 121 Homework 6

John Ingram

11/27/19

Problem 1

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];

}

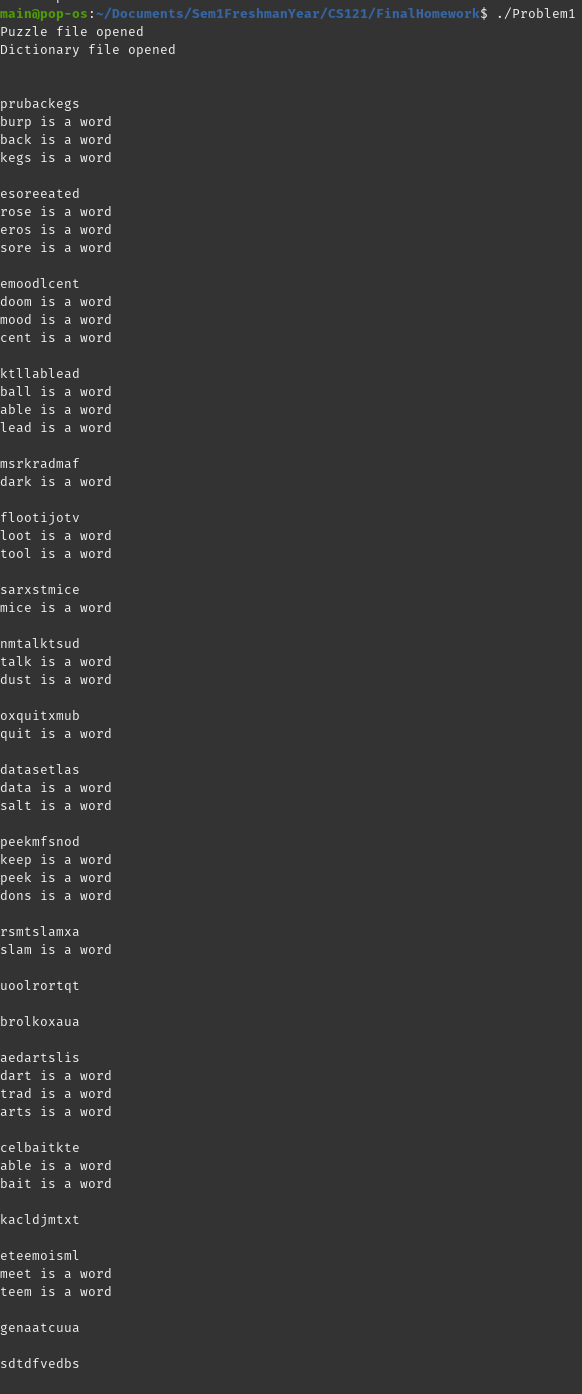
puzzle.clear();

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

return result;

}

Output:



Problem 2

Code:

// This program is a modified version of the instructor

// provided "dynamic\_2d\_arrays\_case2" program

#include <iostream>

#include <fstream>

#include <iomanip>

#include <string>

#include <vector>

#define SKIP2 cout << endl << endl;

using namespace std;

void initialize(unsigned int\*\* array, unsigned int rows, unsigned int cols);

bool contains(vector<unsigned int> array, unsigned int number);

void transpose(unsigned int\*\* sourceArray, unsigned int\*\* transporseArray, unsigned int rows);

void printarray(unsigned int\*\* array, unsigned int rows, unsigned int cols);

vector<unsigned int> usedNumbers;

int main()

{

int j, k;

int rows = 0, columns = 0;

unsigned int old\_numbers[6][10] = { 0 };

unsigned int\*\* table = nullptr;

unsigned int\*\* array\_transpose = nullptr;

// Create 2D Dynamic array with arbitrary rows and columns

cout << "Enter the number of rows and columns for the array " << endl;

cin >> rows;

table = new unsigned int\* [rows];

for (k = 0; k < rows; k++)

table[k] = new unsigned int[rows];

array\_transpose = new unsigned int\* [rows];

for (k = 0; k < rows; k++)

array\_transpose[k] = new unsigned int[rows];

cout << "Initialize and print 2D array table" << endl;

initialize(table, rows, rows);

printarray(table, rows, rows);

SKIP2

cout << "Transposed Array" << endl;

transpose(table, array\_transpose, rows);

printarray(array\_transpose, rows, rows);

cout << endl;

return 0;

}

//

// Initialize a two dimensional array with random positive integers

// This function requires a pointer to unsigned int for formal parameter array

// This function has been modified from it's original to

// ensure that each unsigned integer is unique

//

void initialize(unsigned int\*\* array, unsigned int rows, unsigned int cols)

{

unsigned int i, j, current = rand() % 4000;

for (i = 0; i < rows; i++)

{

for (j = 0; j < cols; j++)

{

//Keep generating numbers till one hasn't already been used

while (contains(usedNumbers, current))

{

current = rand() % 4000;

}

usedNumbers.push\_back(current);

array[i][j] = current;

}

}

}

bool contains(vector<unsigned int> array, unsigned int number)

{

for (unsigned int i = 0; i < array.size(); i++)

{

if (array.at(i) == number) return true;

}

return false;

}

//

// Print the contents of a two dimensional array with variable rows and columns

// The actual array passed to the function must contain pointers to unsigned int

// This function has been modified to support the format requested by the instructor

//

void printarray(unsigned int\*\* array, unsigned int rows, unsigned int cols)

{

unsigned int j, k;

for (j = 0; j < rows; j++)

{

for (k = 0; k < cols; k++) {

cout << setw(4) << array[j][k] << " ";

}

cout << endl;

}

cout << endl;

return;

}

void transpose(unsigned int\*\* sourceArray, unsigned int\*\* transporseArray, unsigned int rows)

{

for (unsigned int r = 0; r < rows; r++)

{

for (unsigned int c = 0; c < rows; c++)

{

transporseArray[r][c] = sourceArray[c][r];

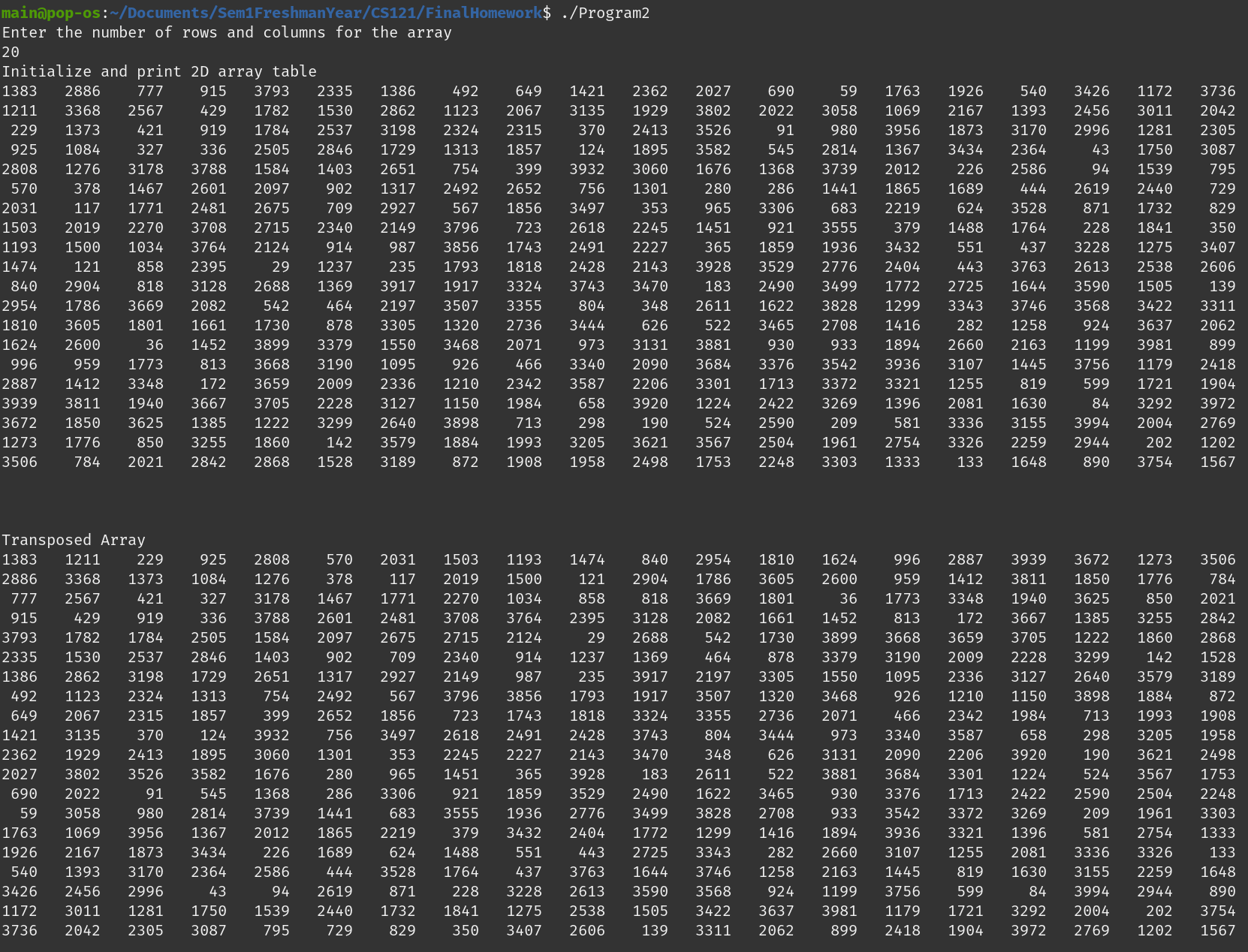
}

}

return;

}

Output:



Problem 3

Code:

//

//

//

#include <iostream>

#include <fstream>

#include <iomanip>

#include <string>

#include <vector>

using namespace std;

struct dateStruct

{

unsigned int month, day, year;

};

struct timeStruct

{

unsigned int hours, minutes, seconds;

};

struct idStruct

{

string prefix = "";

unsigned int number;

string state = "";

};

struct entry

{

idStruct id;

float lat, longit, alt;

dateStruct date;

timeStruct time;

float temp, prcp, wind, press;

};

bool contains(vector<unsigned int> array, unsigned int number);

void listStations(vector<entry> entries);

void stationDate(vector<entry> entries, idStruct id, dateStruct date);

float totalPRCP(vector<entry> entries, dateStruct date);

int main(void)

{

ifstream dataFile;

vector<entry> entries;

string trash;

char tempChar;

dataFile.open("./Data.txt");

if (!dataFile)

{

cout << "Unable to open Data File" << endl;

return 1;

}

getline(dataFile, trash);

while (!dataFile.eof())

{

entry temp;

for (int i = 0; i < 3; i++) // Populate 'Prefix'

{

dataFile >> tempChar;

temp.id.prefix += tempChar;

}

dataFile >> temp.id.number;

for (int i = 0; i < 2; i++) // Populate 'State'

{

dataFile >> tempChar;

temp.id.state += tempChar;

}

dataFile >> temp.lat;

dataFile >> temp.longit;

dataFile >> temp.alt;

dataFile >> temp.date.month;

dataFile >> temp.date.day;

dataFile >> temp.date.year;

dataFile >> temp.time.hours;

dataFile >> temp.time.minutes;

dataFile >> temp.time.seconds;

dataFile >> temp.temp;

dataFile >> temp.prcp;

dataFile >> temp.wind;

dataFile >> temp.press;

entries.push\_back(temp);

}

dataFile.close();

cout << "Data read. Select an operation (1-3): ";

int operation;

cin >> operation;

cout << endl;

if (operation == 1)

{

listStations(entries);

}

if (operation == 2)

{

idStruct tempID;

dateStruct tempDate;

cout << "Enter a station id: ";

for (int i = 0; i < 3; i++) // Populate 'Prefix'

{

cin >> tempChar;

tempID.prefix += tempChar;

}

cin >> tempID.number;

for (int i = 0; i < 2; i++) // Populate 'State'

{

cin >> tempChar;

tempID.state += tempChar;

}

cout << "Enter a Date (MM DD YYYY): ";

cin >> tempDate.month;

cin >> tempDate.day;

cin >> tempDate.year;

stationDate(entries, tempID, tempDate);

}

if (operation == 3)

{

dateStruct tempDate;

cout << "Enter a Date (MM DD YYYY): ";

cin >> tempDate.month;

cin >> tempDate.day;

cin >> tempDate.year;

cout << endl;

cout << "The total percipitation for that day is " << totalPRCP(entries, tempDate) << endl;

}

return 0;

}

void listStations(vector<entry> entries)

{

vector<unsigned int> used;

for (int i = 0; i < entries.size(); i++)

{

if (!contains(used, entries.at(i).id.number))

{

cout << "NWS" << setw(3) << setfill('0') << entries.at(i).id.number << "AL" << endl;

cout << "Latitude:\tLongitude:\tAltitude:" << endl;

cout << setw(9) << setfill(' ') << entries.at(i).lat << "\t";

cout << setw(9) << setfill(' ') << entries.at(i).longit << "\t";

cout << setw(9) << setfill(' ') << entries.at(i).alt << endl;

used.push\_back(entries.at(i).id.number);

}

}

}

bool contains(vector<unsigned int> array, unsigned int number)

{

for (unsigned int i = 0; i < array.size(); i++)

{

if (array.at(i) == number) return true;

}

return false;

}

void stationDate(vector<entry> entries, idStruct id, dateStruct date)

{

cout << "Hours: Minutes: Seconds: Temperature: Percipitation: Wind: Pressure: " << endl;

for (int i = 0; i < entries.size(); i++)

{

if (

entries.at(i).id.number == id.number &&

entries.at(i).id.prefix == id.prefix &&

entries.at(i).id.state == id.state &&

entries.at(i).date.day == date.day &&

entries.at(i).date.month == date.month &&

entries.at(i).date.year == date.year

)

{

cout << setw(6) << entries.at(i).time.hours << " ";

cout << setw(8) << entries.at(i).time.hours << " ";

cout << setw(8) << entries.at(i).time.hours << " ";

cout << setw(12) << entries.at(i).temp << " ";

cout << setw(14) << entries.at(i).prcp << " ";

cout << setw(5) << entries.at(i).wind << " ";

cout << setw(9) << entries.at(i).press << " ";

cout << endl;

}

}

}

float totalPRCP(vector<entry> entries, dateStruct date)

{

float total = 0;

for (int i = 0; i < entries.size(); i++)

{

if (

entries.at(i).date.day == date.day &&

entries.at(i).date.month == date.month &&

entries.at(i).date.year == date.year

)

{

total += entries.at(i).prcp;

}

}

return total;

}

Output:

