Laboratory Exercise 2: Finding words

Assigned: Friday 4 September
Paper homework due: Monday 7 September ***at 2:30pm***[1]
Due: Friday 11 September at 12:00pm

The purpose of this lab is to write some simple algorithms concerning searching in grids, and implement them in C++, making use of a pre-existing class. The first part is a homework for the weekend. We will focus on the second part in lab on Monday.

1 Homework

Review the standards for written work at
`https://piazza.com/hamilton/fall2015/cs111/resources`.

Neatly, using pencil/paper, or a text editor/typesetting software, write two standalone functions as specified. You might want to keep a second copy of your solution handy for the lab period.

```
string substr(const string & original, size_t start, size_t length)
// This function performs the same operation as the C++ string::substr method,
// (though its start and length parameters are not optional).
// If you implement this function correctly, you are free to use the method
// in your project.

size_t distance_to_edge(size_t row, size_t col, size_t D, int dr, int dc);
// Supposing that (row, col) represents a grid location within a DxD matrix
// of cells, return the number of cells between that location and the edge
// when moving toward the edge on the vector represented by dr and dc.
// For example consider the D=8 (8x8) grid below:
//     0  1  2  3  4  5  6  7
//    +--+--+--+--+--+--+--+--+
//  0 |  |  |b |  |  |c |  |  |
//    +--+--+--+--+--+--+--+--+
//  1 |a |  |b |  |c |  |  |  |
//    +--+--+--+--+--+--+--+--+
//  2 |  |a |b |c |  |  |  |  |
//    +--+--+--+--+--+--+--+--+
//  3 |d |d |XX|e |e |e |e |e |
//    +--+--+--+--+--+--+--+--+
//  4 |  |f |g |h |  |  |  |  |
//    +--+--+--+--+--+--+--+--+
//  5 |f |  |g |  |h |  |  |  |
//    +--+--+--+--+--+--+--+--+
```

[1] Be prepared to hand in your homework, and do so before class begins. Absolutely no late homeworks will be accepted.

```
//  6 |  |  |g |  |  |h |  |  |
//    +--+--+--+--+--+--+--+--+
//  7 |  |  |g |  |  |  |h |  |
//    +--+--+--+--+--+--+--+--+

// Starting at position row=3, col=2, there are 8 vectors toward the edges
// with the indicated distance_to_edge:
//
// a (dr=-1, dc=-1)  : distance_to_edge returns 3
// b (dr=-1, dc=0)   : distance_to_edge returns 4
// c (dr=-1, dc=1)   : distance_to_edge returns 4
// d (dr=0 , dc=-1)  : distance_to_edge returns 3
// e (dr=0 , dc=1)   : distance_to_edge returns 6
// f (dr=1 , dc=-1)  : distance_to_edge returns 3
// g (dr=1 , dc=0)   : distance_to_edge returns 5
// h (dr=1 , dc=1)   : distance_to_edge returns 5
//
//
// PRECONDITION
// 0 <= row < D, 0 <= col < D
// -1 <= dr <= 1, -1 <= dc <= 1; dr and dc are not both zero.

// You may use basic functions such as min(a,b) and max(a,b).
// If it helps you better break the problem down into its cases,
// you might also write auxiliary functions, though this is not
// necessary.  You should certainly not write any loops.
```

2 Computer work

1. Login to gemini and create the directory for this assignment. You **must not** do this more than once. The command below copies a directory from my account to yours. That directory contains some scaffolding code for you to start with.

   ```
   cp -rp ~acampbel/wordfind ~/111
   ```

2. Each time you log in to work, change to the correct directory

   ```
   cd ~/111/wordfind
   ```

3. Finish the implementation of `wordfind.cc` so that the program's behavior conforms to the description in section 3.

   ```
   emacs wordfind.cc &
   ```

4. Use good style at all times. Be consistent with use of white space, especially conventions for indentation. Choose appropriate variable names. Never exceed 80 characters per line. Don't use any unnecessary punctuation. Curly braces should follow standard conventions. (Ask about this in lab if you're not sure.) Carefully, succinctly and completely comment

your methods and functions. At the top of your implementation, provide a header block like this one:

```
/*****************************************************************************
*                                                                            *
* THE NAME OF THE FILE                                                       *
* YOUR NAME                                                                  *
* THE NAME OF THE ASSIGNMENT                                                 *
* THE DATE                                                                   *
*                                                                            *
* A short paragraph describing what your code accomplishes...                *
*                                                                            *
*****************************************************************************/
```

5. You've been provided with a makefile. To compile, use the make utility.

   ```
   make
   ```

6. It creates an executable called `wordfind`. You can run wordfind with its input redirected from a file instead of the keyboard. One sample input has been supplied. You can use emacs to write others.

   ```
   ./wordfind < sample.in
   ```

7. When you think you're ready to submit your work, at the shell window type

   ```
   submit
   ```

   This will collect your `wordfind.cc` file and subject it to a battery of tests. Results will be shown. You can always `submit` as many times as you want. We retain all copies, but we grade only the last one before the deadline.

3 Program description

Suppose you have a grid of lowercase letters, and, hidden in those letters is a word you're searching for. It can be horizontal, vertical, or diagonal in any direction.

Write a program in C++ that finds a word in a grid, converting to upper case if found.

The input will consist of a side-length D followed by a grid of $D \times D$ characters comprising the grid, followed by a word W to search for. If W is found in the grid, the output will be the word `FOUND` followed by the grid, with the word indicated in uppercase. If W is not found, the output will be the phrase `NOT FOUND`

3.1 Sample input

```
10
jsdpovkrjs
dfmabebdjd
ncettoswsf
cjolaisjew
dklqdjglou
dlwfksmefk
djuwlfmasj
ajmcolelaj
vjfklwsfai
dkvnbueoqp
meow
```

3.2 Sample output

```
FOUND
jsdpovkrjs
dfmabebdjd
ncettoswsf
cjolaisjeW
dklqdjglOu
dlwfksmEfk
djuwlfMasj
ajmcolelaj
vjfklwsfai
dkvnbueoqp
```

A class `wordgrid` is provided for your use. You can use `#include <wordgrid.h>` in the header.

4 `wordgrid.h`

For your reference, here is the documentation for `wordgrid`.

```
#include <iostream>

using namespace std;

// class wordgrid provides an infrastructure for square (DxD)
// grids of letters.  Each slot is referenced with a row number r,
// 0 <= r < D and a column number c, 0 <= c < D.

class wordgrid {

 public:

  //-wordgrid(size_t d, string data)--------------------------------------
```

```
  wordgrid(size_t d, string data);
  // PRE: second parameter is an long string of lowercase letters,
  //      which is the data for all the rows of the d x d grid.
  //      (data.length() == d * d)
  // POST: initialize the instance with the data.
  //-------------------------------------------------------------------

  //-size_t size() const-----------------------------------------------
  size_t size() const;
  // RETURN: The dimension of the grid (length of one side)
  //-------------------------------------------------------------------

  //-char get(size_t r, size_t c) const --------------------------------
  char get(size_t r, size_t c) const;
  // PRE: r < size() and c < size()
  // RETURN: the character at position (r,c)
  //-------------------------------------------------------------------

  //-void set(size_t r, size_t c, char v)------------------------------
  void set(size_t r, size_t c, char v);
  // PRE: same as get
  // POST: the character at position (r,c) is now v
  //-------------------------------------------------------------------

  //-void output(ostream & ostr) const---------------------------------
  void output(ostream & ostr) const;
  // POST: The contents of the grid are produced on the
  //       output stream in rows terminated by newlines.
  //-------------------------------------------------------------------

 private:
  wordgrid(const wordgrid &);  // disable copy constructor
  char * _data;
  size_t _dim;
  size_t _getLoc(size_t r, size_t c) const;
};

// -ostream & operator<<(ostream & ostr, const wordgrid & wg)----------
// support the stream output operator so that
// statements like 'cout << wg;' works when
// wg is a wordgrid.
ostream & operator<<(ostream & ostr, const wordgrid & wg);
```