CSCI 2270, Lab 6
Sequence alignment algorithm project in C++
Due 3/1 at 11:55pm by Moodle

How similar are 2 strings of text? It's easy to say that BOBO == BOBO, but harder to describe the similarity between strings like BOBO and BONOBO, even though you can tell they share 4 letters in the same order (in two different ways). How might we quantify this similarity? And could we extend that quantification to longer text strings, which would be hard for us to match up by hand?

This method is sometimes called the "longest common subsequence" or LCS problem. It relies on the inductive observation that the best match to BOBO and BONOBO must depend on finding the best match at the beginning of the 2 strings (for instance, that first "B" should match) and continuing this match through to the best match for the rest of the 2 strings after that (OBO and ONOBO).

This handy insight points to a way to solve the problem by making a 2-dimensional array, with one string along the top and one along the side. This array has as many rows as the length of the first string plus one more row, and as many columns as the length of the second string plus one more row. We initialize it by making the first row and column all 0:

		B	O	N	O	B	O
	0	0	0	0	0	0	0
B	0						
O	0						
B	0						
O	0						

Now we fill in the table according to this rule: If there's a cell in the array at row i and column j, and the letters of string 1 at i-1 and string 2 at j-1 match, we take the maximum of three numbers: (the number in row i - 1, column j; the number in row i, column j - 1; and 1 + the number in row i - 1, column j - 1). If there's no match, we take the maximum of (the number in row i - 1, column j and the number in row i, column j - 1.

Filling in our example we get the following table, with the top score in the bottom right corner:

		B	O	N	O	B	O
	0	0	0	0	0	0	0
B	0	1	1	1	1	1	1
O	0	1	2	2	2	2	2
B	0	1	2	2	2	3	3
O	0	1	2	2	3	3	4

The 2 matches are:

```
BO - - BO          and          B - - OBO
BONOBO                          BONOBO
```

To recover these instead of just getting the number of matching letters right, you'd need to do more work, and that's beyond the scope of this lab. (But it can be done. It's just more bookkeeping.)

For this lab, download Aligner.h, Aligner.cxx, TestAligner.cxx, and the Makefile from moodle to a new directory in the VM. Open TestAligner.cxx; instead of compiling this code with the buttons, you're going to use your first Makefile to put it together. Go to the Build dropdown menu in Geany and click Make to run the Makefile and compile the code.

Your task is to finish the match() method in Aligner.cxx that computes the longest common subsequence. This works according to the logic I showed you in the slides on Friday. You should test your match method against a few string matches that you have worked out by hand.

Your code prompts the user for 2 strings (you will type these in) and then matches them by computing the alignment in the function:

```
unsigned int Aligner::match()
```

In this code, you must calculate each match between letter q-1 of sequence 1 and letter r-1 of sequence 2. Be sure you're clear on which string depends on q and which one depends on r.

1) If the letters at sequence1[q-1] and sequence2[r-1] match, then set the matching at row q, column r to the maximum of (1+ the matching at row q-1, column r-1 OR the matching at row q-1, column r OR the matching at row q, column r-1).

2) If the letters do not match, then set the matching at row q, column r to the maximum of (the matching at row q-1, column r OR the matching at row q, column r-1).

This involves a series of if statements inside the 2 for loops in the **`match()`** function, but if you can correctly predict the matching of a few short strings from your TA before lab is over, you can call it good and leave. Upload your code to the Moodle for safety.