This is a 90-minute programming exam. It is semi-open book:

- You are allowed to read the following offline materials: past programs that you have written (including assignments, labs, and homework), the course textbook, and any of your notes.
- You are allowed to read the following online materials: the course website (including Websheets exercises), the course Piazza forum, and library reference pages from cplusplus.com and cppreference.com.
- You are not allowed to read any other materials.
- You are not allowed to communicate with anybody other than course staff.

Complete the programs on your laptop. You will upload them to the course website for submission (the link is inside this exam). Like a programming assignment: you can run some basic tests through the website, though they might not exhaustively check for all bugs; you can upload your work as many times as you like, but only the last submission will be graded.

We will primarily grade your code on correctness. Code that compiles will receive a bonus compared to code that does not. *Please write your name at the top of every file you submit.* Other than this, we will not be picky about style, but we recommend that you use good style because it will help you debug your code more quickly, and help us read your code.

Print your name, print your email address, and select your lecture section now. **You must hand this exam booklet back to the course staff at the end of the exam.**

Your Name: 

Your USC e-mail: 

Your Lecture Section:

<table>
<thead>
<tr>
<th>Section</th>
<th>Time</th>
<th>Instructor</th>
</tr>
</thead>
<tbody>
<tr>
<td>29991</td>
<td>2PM MW</td>
<td>David Pritchard</td>
</tr>
<tr>
<td>29902</td>
<td>11AM TTh</td>
<td>David Pritchard</td>
</tr>
<tr>
<td>29906</td>
<td>2PM TTh</td>
<td>David Pritchard</td>
</tr>
</tbody>
</table>
As the end of the semester comes near, you are planning your summer vacation and some trips abroad. To aid you on these trips, in this exam, you will develop a C++ class to help with translation between different languages.

**Part 1: Bon Voyage (13 points)**

You will create a class `Dict` that remembers how to translate from one language to another. The Part 1 API is:

```cpp
class Dict {
    public:
        Dict(); // construct empty translation dictionary
        void add(string word, string trans); // add translation of word to dictionary
        string lookup(string word); // get translation of word ("??" if none)
    private:
        // whatever you decide
};
```

Here is a series of sample calls to the API (`p1test.cpp`):

```cpp
// p1test.cpp
Dict enfr;
enfr.add("hello", "bonjour");
enfr.add("hi", "salut");
enfr.add("cat", "chat");
cout << "cat lookup is " << enfr.lookup("cat") << endl; // "chat"
cout << "dog lookup is " << enfr.lookup("dog") << endl; // "??"
cout << "hello lookup is " << enfr.lookup("hello") << endl; // "bonjour"
```

As you can see, the `add` function is used to store a word in the original language, and its translation in the other language. (For this example, the translation is from English to French.) When we query the `Dict` with `lookup`, it should look through the stored information and return the correct translation. If a translation was never added (like for "dog"), the returned string should be "??".

To implement this functionality, we recommend you either (a) store 2 parallel vectors, or (b) define a vector of `structs` each containing a word and its translation. However, any correct approach using the tools learned in the course is ok. Note that it may be correct for your constructor to be empty.

You can assume that we never try to add two translations for the same word. For instance, we will never run `enfr.add("book", "livre"); enfr.add("book", "cahier");`

We provide skeleton code, other useful files, and the submission link here:

[http://bits.usc.edu/exam/dict](http://bits.usc.edu/exam/dict)

Write your class in `dict.h` and `dict.cpp`. We recommend you upload these files and check your Part 1 code before continuing to Parts 2 and 3.
Part 2: C’est What? (12 points)

Good news! You are on your trip and travelling around the world. However, you’ve found it cumbersome to have to write a new C++ program every time you want to say a new sentence. Therefore, you decide to automate your translation tool.

Part 2A: First, you will add one more member function to the Dict API:

```cpp
// open a text file, each line of which has a word and its translation
// add _each pair_ to the Dict
void add_all(const char* filename);
```

Be sure to uncomment this line in `dict.h`.

Here is an example of calling this new function. Suppose `deen.txt` contains this:

```
contents of deen.txt:
ballons balloons
schokolade chocolate
neunundneunzig ninety-nine
```

Then, the following example code (`p2test.cpp`) will print out "chocolate":

```
// p2test.cpp
Dict d;
d.add_all("deen.txt");
cout << d.lookup("schokolade") << endl;
```

You should assume that words and their translations do not contain spaces.

Part 2B: Additionally, create a program `translate.cpp` that takes a dictionary filename as a command-line argument, and reads a message from standard input, then prints out the translated version of that message, using the Dict class. The input will be a single line separated by spaces, and the output should be too. For instance if `msg.txt` contains the following:

```
contents of msg.txt:
neunundneunzig luft ballons
```

Then, the output of `./translate deen.txt < msg.txt` should be

```
ninety-nine ?? balloons
```

We recommend you upload and check your Part 2 code before continuing to Part 3. Upload your modified `dict.cpp`, `dict.h`, and the new program `translate.cpp` to the link given earlier.
Part 3: Broken Telephone (5 points)

Due to your expertise with global travel, you’ve just scored a great internship at an extremely exciting dictionary company. However, the company is very cheap! They want to combine their existing dictionaries to create new ones. To help accomplish this, you will add two new functions to the Dict API:

// print out contents of this dictionary, one word and its translation per line
void print_all();

// create a new dictionary. translating with it should have the same effect
// as using the first dictionary, followed by the other dictionary
Dict compose(Dict other);

Be sure to uncomment these lines in dict.h.

We give you the test program compose.cpp which calls these functions:

// compose.cpp
int main(int argc, char* argv[]) {
    Dict first, second;
    first.add_all(argv[1]);
    second.add_all(argv[2]);
    first.compose(second).print_all();
}

Assume that enfa.txt contains the following:

<table>
<thead>
<tr>
<th>contents of enfa.txt:</th>
</tr>
</thead>
<tbody>
<tr>
<td>lion shir</td>
</tr>
<tr>
<td>chocolate shokolat</td>
</tr>
<tr>
<td>milk shir</td>
</tr>
<tr>
<td>balloons bahdkonakha</td>
</tr>
</tbody>
</table>

Then ./compose deen.txt enfa.txt should output

ballons bahdkonakha
schokolade shokolat

Note, only words that could be translated all the way from the first language into the second and onto the third should be listed. E.g., it would be wrong to include neunundneunzig ?? as another line of output in the example just given. However, the order of the lines printed by print_all() does not matter.

Upload your modified dict.cpp and dict.h to the link given earlier.
Extra Tests

The files `fren.txt`, `msg2.txt`, `msg3.txt` contain the following:

<table>
<thead>
<tr>
<th>contents of fren.txt:</th>
</tr>
</thead>
<tbody>
<tr>
<td>comme like</td>
</tr>
<tr>
<td>ci this</td>
</tr>
<tr>
<td>lait milk</td>
</tr>
<tr>
<td>livre book</td>
</tr>
<tr>
<td>chocolat chocolate</td>
</tr>
<tr>
<td>cahier book</td>
</tr>
<tr>
<td>lion lion</td>
</tr>
<tr>
<td>ca that</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>contents of msg2.txt:</th>
</tr>
</thead>
<tbody>
<tr>
<td>comme ci comme ca</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>contents of msg3.txt:</th>
</tr>
</thead>
<tbody>
<tr>
<td>livre est comme cahier</td>
</tr>
</tbody>
</table>

Example: the output of `./translate fren.txt < msg2.txt` should be

```
like this like that
```

Example: the output of `./translate fren.txt < msg3.txt` should be

```
book ?? like book
```

Example: the output of `./compose fren.txt enfa.txt` should be

```
lion shir
chocolat shokolat
lait shir
```

(though the order of the lines is not important).