Project

1. Basic Requirements

There will be two input sets in this project and the tasks of this project are:

(1). Print the Cartesian Product of these two sets.

(2). Print all the subsets of the first set.

(3). **Print the intersection of these two sets.**

The elements for the given sets are read from two input files. Please reference the Input File Specification section.

Please **pay attention**: if there are duplicated elements from input files, you should **remove duplicates**, and keep every element unique in your sets. And thus, we can get the size of each set easily for verification.

Meanwhile, the public functions of the Set class **should not include function parameters based on the order of the elements**, e.g., there should be no function to retrieve the i-th element in the internal array. A **hint**: look at the APIs of set in other programming languages, where set elements are unordered.

In this project, you can assume the elements in our sets are all **string** type instances. Please just use the utilities in the given code, and **don't use C++ Standard Template Library** (STL).

2. Program Structure

The program structure is provided, including the implementation of a basic Set class, and framework of the main function, consisting of 4 files: p02.cpp, Set.h, Set.cpp, and p02make. The file names of this project should be **exactly as given**. Pre-defining the file names is like an agreement when collaborating with others in a real project, so that others can easily call your program.

2.1 Program Files

Please reference the following file list for this project (**the files are given in this project, and you just need to "fill-in the blanks"**):

File	Description
p02.cpp	File **p02.cpp** contains the main function, and other functions that process command line arguments and help with calculation of Cartesian Product and subsets.
Set.h	File **Set.h** contains the interface of the basic *Set* class.
Set.cpp	File **Set.cpp** contains the implementation of class *Set*.
p02make	File **p02make** contains instructions to create executable program **p02**. File **p02make** is interpreted by the Linux command ***make***: **$ make -f p02make** (**If you are using Windows, please change this accordingly, based on your past experience in Programming I**)

You can choose to use the provided source code, or implement everything by yourself (but keep the file names unchanged). You can also make any changes to the provided source code as long as the input and output are consistent. There are some unimplemented parts in the given source code, and you need to finish those parts to make the program complete. They are marked as "**TODO**" in the comments.

2.2 Command Line

To compile all your source code, we can make use of our make file:

```
make -f p02make
```

To execute your program after a successful compilation, you can use:

```
./p02 input1.dat input2.dat output.dat
```

This is the way how we use command line argument to get input parameters from the users. You can find further information regarding command line argument from:

http://www.cplusplus.com/articles/DEN36Up4/

The input and output files should not be hard coded in your program.

2.3 Input Specification

The input files consist of a sequence of strings. Strings are separated by white space(s). White space could be from Space, Enter, or Tab on the keyboard, and as a result, the character of white space(s) recorded in the file would be **a blank character, a newline character, and a tab character,** respectively.

An example for input1.dat could be (pay attention to the large white spaces in the last line):

```
123

345
123       345                               543
```

An example for input2.dat could be (pay attention to the empty line, i.e., the third line):

```
abcd   john          Smith
codecodecode

john
```

2.4 Output Specification

The output format has been specified in the given code. You just need to calculate the contents of the corresponding sets.

The first part of the output file consists of the Cartesian products of the two sets read from the two input files. Elements of the set are enclosed in curly braces, { }, and separated by commas. Each element of the set is an ordered pair enclosed in parentheses where an element from the first set is separated from and element from the second set by a comma. If there is an **empty set**, just put nothing in the braces. Here **no duplicates** should be involved in the sets. And at the end print the **total number of elements in the Cartesian Product (the value should not be hard-coded)**.

The second part of the output file consists of all the subsets created based on the first input file. Elements of the resultant set should be enclosed in curly brackets, i.e., { }, and separated by commas. If there is an **empty set**, just put nothing in the braces. Here **no duplicates** should be involved in the sets. And at the end print the **total number of subsets (the value should not be hard-coded)**.

The third part of the output file consists of the intersection of the two sets read from the two input files. If there is an **empty set**, just put nothing in the braces. Here **no duplicates** should be involved in the sets.

An example for output.dat could be:

```
A={123,345,543}

B={abcd,john,Smith,codecodecode}

A X B={

(123,abcd),(123,john),(123,Smith),(123,codecodecode),
(345,abcd),(345,john),(345,Smith),(345,codecodecode),
(543,abcd),(543,john),(543,Smith),(543,codecodecode)}

There are 12 elements in the Cartesian Product.

-----------------------------------------------------------------------------

A={123, 345, 543}.
Subsets of A:
{},
{123},
{345},
{543},
{123,345},
{123,543},
{345,543},
{123,345,543}.
There are 8 subsets of A.

-----------------------------------------------------------------------------

A = {123,345,123,345,543}

B = {abcd,john,Smith,codecodecode,john}

A ∩ B = {}
```