CS2351 Data Structures

Final Project
Due: 11:59pm, June 26, 2011

In this project, you are asked to practise the usage of data structures by implementing a contact book. You may design any new data structures that you find appropriate, or you may simply use the ones that you have learnt in the class for this assignment. All the data structures in your program (except arrays and pointers) must be implemented by yourself, which means that you cannot use the libraries of data structures such as STL in C++. You should submit the source code of your program to the iLMS system (http://lms.nthu.edu.tw/) before the deadline. To grade your assignment, we will arrange a 25-minute session with you for a demonstration of your program, at General Building II Room 734 on either June 27 or June 28. Please make sure that the source code can be compiled without any error. Late submission will get at most 60% for the grade.

Contact Book

A contact book contains the data you need when contacting your friends. The data may consist of various attributes such as first name, last name, age, sex, phone number, email, and birthday. See Table 1 for an example. The data should be stored in some data structures (instead of storing as a plain text) to allow efficient operations. The major challenge of this assignment is for you to think about what kind of data structures should be used in order to support the desired functions (as described in the next section). For example, using a sorted array for phone number is very good for searching, but if phone numbers may be inserted or deleted from time to time, a simple array will not be suitable. To simplify the task, you may assume each element in some of the attributes is unique. For instance, in our example in Table 1, we have assumed that each phone number is unique.

First name	**Last name**	**Phone number**
Harry	Potter	0995435355
Hermione	Granger	0958416991
Ron	Weasley	0901772105
Ginny	Weasley	0926925019
Draco	Malfoy	0990224661
Albus	Dumbledore	0911276562
Tom	Riddle	0975867725

Table 1: An example of a contact book

Functions of a Contact Book

We define the following three categories of functions in this project: Basic, optional and others. Your program should implement all the basic functions on at least three attributes. To get a higher grade, you may implement some optional functions, or add other functions that you find useful. The following are the descriptions of the basic functions:

- **User interface:** A readable user interface. The interface needs not be window-based, a command-line interaction will already be quite good.

```
7
Harry,Potter,0995435355
Hermione,Granger,0958416991
Ron,Weasley,0901772105
Ginny,Weasley,0926925019
Draco,Malfoy,0990224661
Albus,Dumbledore,0911276562
Tom,Riddle,0975867725
```

Figure 1: Example file content of Table 1

- **Search:** Given an attribute of a friend, find another (or all) attribute of that friend.
- **Insertion:** Insert a new friend into your contact book.
- **Deletion:** Delete an existing friend from your contact book by given a specific attribute of that friend.
- **Edit:** Update an attribute of a friend by given a specific attribute of that friend.
- **Load:** Read data from a file which contains data of multiple friends. See Figure 1 as a file example. This function may be used at the time the program starts, and the function can be thought as a sequence of multiple insertions. However, we encourage you to think about if there is some way to load data faster. Note that you can define your own file format, which means that your program do not need to be stored according to the format of Figure 1.
- **Save:** Write data of your contact book to a file. Since the data of a running program would be lost if the system crashes, saving data is a good way to protect the data against crash. Note that the file format should be consistent with your load function.

The following are the descriptions of the optional functions:

- **Max (or Min):** Find the maximum (or minimum) of a specific attribute.
- **Average:** Find the average of a specific attribute. Note that the given attribute can only be numeric attribute.
- **Undo:** Undo an insertion or a deletion. Your program can also support undoing edit.
- **Redo:** Redo the function which is undone before.
- **And:** Given the values of two or more attributes, find the friends with all attribute values matching.
- **Or:** Given the values of two or more attributes of some friends, find the friends with at least one attribute value matching.
- **Similarity search:** Given an attribute value $X*$ of some friends, find the friends whose attribute starts with X, where X is some string. For instance, if we search the phone number 099$*$ in Table 1, the results are the data of Harry and Draco. Note that this function may only be done in string attribute.

- **Sorting:** List the friends sorted by a specific attribute. This function should work on *all* attributes, which means that we can list the friends sorted by any attribute we want.
- **Connectivity:** Suppose the relationships of all your friends are stored in the contact book, and all friends of your friends are also in the contact book. Given two of your friends A and B, decide whether A can get the data of B without you. For example, if you (abbreviated as Y) have four friends A, B, C and D in the contact book, and the following relationships are stored
$$\begin{array}{c|ccccc} & Y & A & B & C & D \\ \hline Y & 1 & 1 & 1 & 1 & 1 \\ A & 1 & 1 & 1 & 0 & 1 \\ B & 1 & 1 & 1 & 0 & 0 \\ C & 1 & 0 & 0 & 1 & 0 \\ D & 1 & 1 & 0 & 0 & 1 \end{array}$$
where 1 represents the corresponding row and column are friend which can contact each other directly, 0 otherwise. Then we can see that A can contact B directly without Y, but C cannot. Since C can only contact Y firstly and then get the contact information of B. Thus C can only contact B through Y. In the other case, although D cannot contact B directly, D can contact A firstly and then get contact information of B. Thus D can contact B without Y. To explain further, we assume phone number is stored in the contact book. Considering a situation that your friend A likes your friend B and wants to get the phone number of B. Since A and you are friend, A can call you firstly and ask the phone number of B. Or A can call his friend C_1 to get the phone number. If C_1 does not know the phone number, he may call his friend C_2 for help, and so on. Finally, there are two results, one is you help A directly or indirectly[1], the other is you do not help A at all. This function decide whether A can get the data of B only if you help A. Note that the friend relationships must be stored in your data structures additionally.
- **Date:** Date search is supported by the contact book. The date attribute, such as birthday, should be stored in the data structures and contain the data of year, month and day.

Documentation

This project requires you to provide a simple document which contains the following contents:

- What kinds of functions do your program support?
- What are the data structures used for each function?
- Why do you use these data structures?

Grading

Your grade is consisted of two parts, function grade and documentation grade. For the function grade, it is the sum of the grades, as shown in Table 2, for the functions supported by your program. For the documentation grade, the evaluation is according to the reasons why you choose the particular data structures to be implemented in the your program. Note that the function grade is at most 100%, and the documentation grade is between 0.7 and 1.0. The final grade of the project is

$$\text{Final\_grade} = \text{Function\_grade} \times \text{Document\_grade}$$

[1]If you are the friend of C_i for some i, and you help to find out the phone number of B, then you help A indirectly.

	Function	**Function grade**
Basic	User interface	10%
	Search Insertion Deletion Edit Load Save	60%
Optional	Max Min Average	3% 3% 3%
	Undo Redo And Or	5% 5% 5% 5%
	Similarity search Sorting	10% 10%
	Connectivity	20%
	Date	5%
Others		1∼20%

Table 2: Functions and the corresponding grades