Lab 1 baby

Udaiveer Singh

February 3, 2012

CASE STUDY Designing a Telephone Directory Program

1 Problem

You have a client who wants to store a simple telephone directory in her computer that she can use for storage and retrieval of names and numbers. She has a data file that contains the names and numbers of her friends. She wants to be able to insert new names and numbers, change the number for an entry, and retrieve selected telephone numbers. She also wants to save any changes in her data file.

2 Analysis

The following four subproblems were identified for the telephone directory program:

- Read the initial directory from an existing file
- Insert a new entry
- Edit an existing entry
- Retrieve and display an entry

The author describes 3 *use cases* to help analyze these tasks. There are only 3 since he decided to combine "insert" and "edit" into one because to the user, these tasks would be very similar. Here is what he envisioned:

Use Case for Inserting a New Entry or Editing an Existing Entry

Step	User's Action	System's Response
1.	User issues the command to insert or change an entry.	
2.		System prompts for the name.
3.	User enters name.	If user cancels entry of name, process terminates.
4.		System prompts for the number.
5.	User enters number.	If user cancels entry of number, process terminates.
6.		The directory is updated to contain the new name and number. If the name was not already in the directory, the user is notified that a new name was entered. If the name already exists, the user is notified that the number was changed and is shown both the old and new numbers.

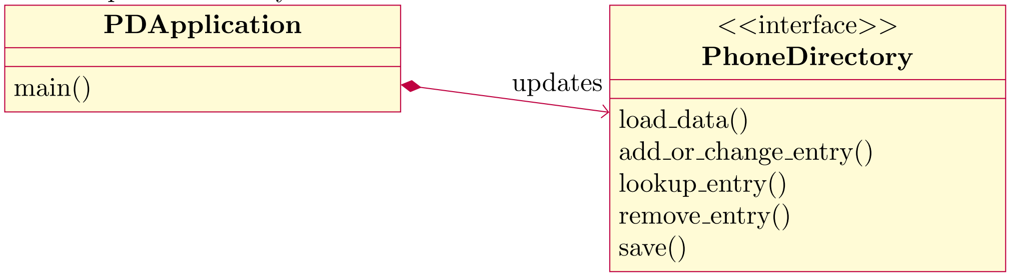
During the analyis he decided to modify the four subproblems to:

- Read the initial directory from an existing file
- Insert a new entry or edit an existing entry
- Retrieve and display an entry
- Save the modified directory back to the file.

He also provides a specification of a Phone_Directory ADT

Function	Behavior
void load_data(const string& source_name)	Load the data file containing the directory, or establish a connection with the data source.
string lookup_entry(const string& name)	Look up an entry in the directory. If there is no such entry, return an empty string.
string add_or_change_entry(const string& name, const string& number)	Changes the number associated with the given name to the new value. Returns the previous value or an empty string if this is a new entry.
string remove_entry(const string& name)	Removes the entry with the specified name from the directory. Returns the name or an empty string if the name was not in the directory.
void save()	Writes the directory to the data file.

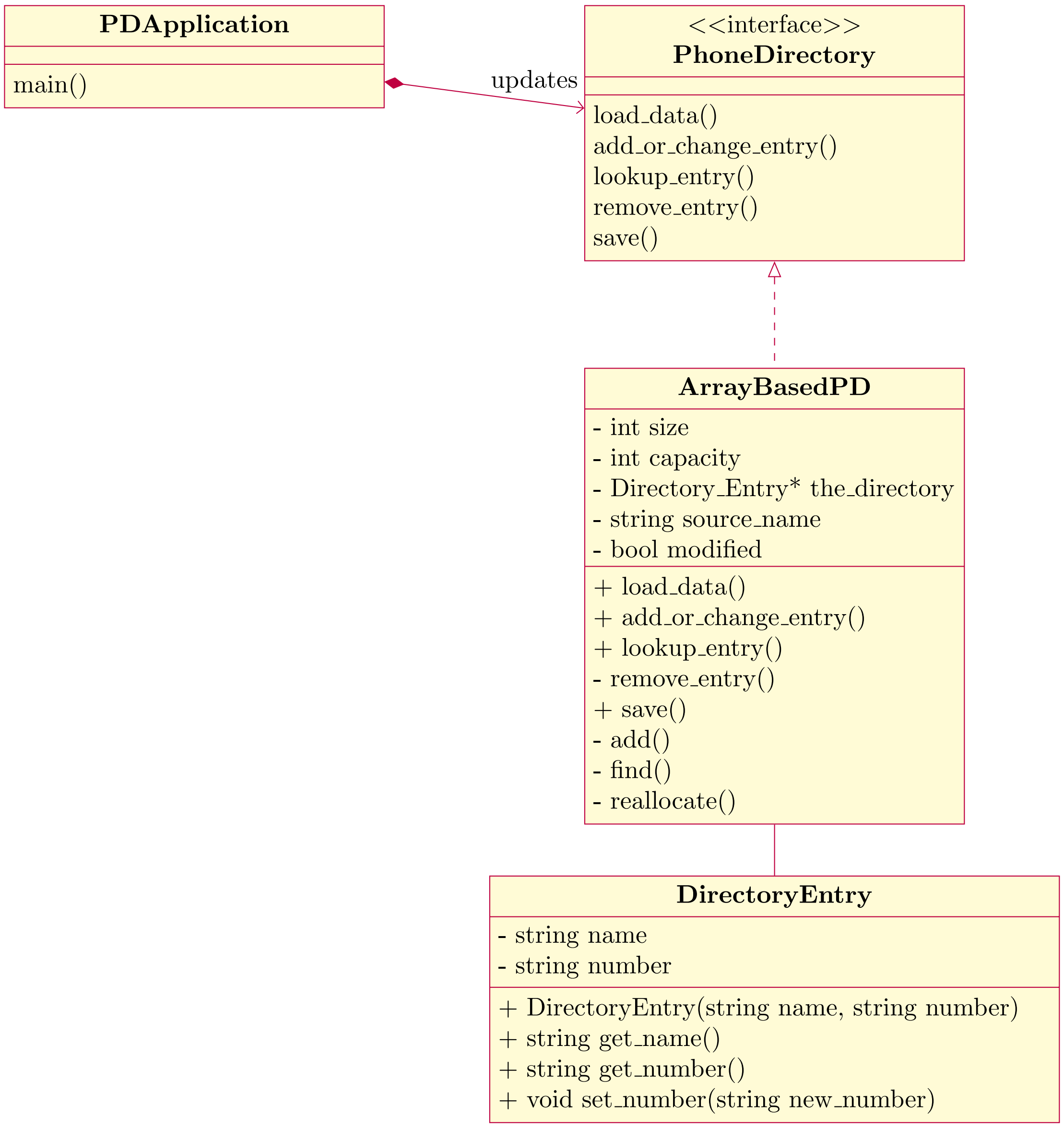
This is an Abstract Data Type (ADT) since it specifies only what the data and operations should be, but not how the data will be represented in memory. UML class diagrams are used to give us a visual picture of the public interface and static relationships between classes. In this diagram, ”interface” indicates that the phone directory is an ADT.



Information about operations such as parameters and return types can be indicated in these diagrams if desired. Although we show PDApplication as a ”class”, actually, in C++, the main function is a global function and not part of any class, so it is not really a class in the true sense of the word.

3 Design

Our design of the interface (ADT) for Phone_Directory doesn't show how the data is represented in memory. We need to ”implement the interface” still. One possible implementation would be to use an array to store the directory information. The author has modified the class diagram (see page 113) to do that. The dotted line with the arrow is UML notation for implementing an interface. The line between Array_Based_PD and Directory_Entry indicates that although the directory is used by the array-based implementation, it has a life of its own and may be used by other classes as well.



An important part of design is to create algorithms for each of the member functions. Here is an example for add_or_change_entry (from page 112).

Algorithm for Function add_or_change_entry

```
1. Call function find to see whether the name is in the directory.
2. if the name is in the directory
3.         Change the number using the set_number function of the Directory_ Entry.
4.         Return the previous value of the number.
   else
5.         Add a new entry using function add.
6.         Return an empty string.
```

Note that we have identified another new function, `find`, for class Array_Based_PD.

4 Implementation

This function calls the internal function `find` to locate the name in the array. Function `find` will either return the index of the entry, or return -1 (minus 1) to indicate that the entry is not in the array. If the entry is in the array, that entrys `set_number` function is called to change the number; otherwise a new entry is added by calling the `add` function:

```
              ▷▷ The add_or_change_entry Function ◁◁
/** Add an entry or change an existing entry.
    @param name The name of the person being added or changed
    @param number The new number to be assigned
    @return The old number or, if a new entry, an empty string
*/
string Phone_Directory::add_or_change_entry(const string& name, const string& number)
{
  string old_number = "";
  int index = find(name);
  if (index != - 1)
  {
    old_number = the_directory[index].get_number();
    the_directory[index].set_number(number);
  }
  else
  {
    add(name, number);
  }
  modified = true;
  return old_number;
}
```

◇◇◇

▷▷ The remove_entry(int index) Function ◁◁

```
/** This finctions besicly shifts all the array parts
to the right. I should point out that the deleated element
is not deleated because an array size can't change.
*/
void Phone_Directory::remove_entry(int index)
  {
  while(index < size)
  {
     the_directory[index] = the_directory[index+1];
     index++;
  }

  }
```

◇◇◇

▷▷ The :remove_entry Function ◁◁

```
/**
This functions takes in a name as a string
 returns you th index so you can use
remove with the index you found to delete the
element you want.
*/
string Phone_Directory::remove_entry(const string& name)
{
  int index = find(name);
  if (index != -1 ) // means it was found
  {
    remove_entry(index); // calls the viod version because
                    // the paramater is a integer
    return name;
  }
  else
  {
    return "";  //
  }
}
```

◇◇◇

▷▷ The do_remove_entry Function ◁◁

```
/**
this function uses a combination of the previous
```

```
two functionss, This functions to remove the entry.
This function takes in the name and number and feeds
it to remove_entry and remove_entry give the
do_remove_entry the index and the do_ remove_entry
does all  the dirty work of shifting the array and deleting
the element.
*/
void do_remove_entry(Phone_Directory& the_directory)
{
  cout << "Enter name: ";
  string name;
  getline(cin, name);
  string name_return = the_directory.remove_entry(name);
  if (name_return != "") {
    cout << name << " has been run over by a truck his number was:"
 << name_return << "and has been deleted";
  } else {
    cout << name << "not in the directory";
  }
}
```

◇ ◇ ◇

5 Test

Unlike Java, C++ does not require that the files have the same name as the class that is defined within. The author suggests that even though our file is Array_Based_PD.h, that we still name the class Phone_Directory based on the name of our original ADT. That has advantages since the user doesn't need to know how we implemented the ADT (e.g. using array) and because we could later change to a different implementation without requiring any change for the user.

He suggests many tests once we have the code completed:

- run it with data files that are empty or
- that contain a single name-and-number pair.
- run it with data files that contain an odd number of lines (ending with a name but no number).
- see what happens when the array is completely filled and you try to add a new entry.
- does function `reallocate` properly double the arrays size?

- when you do a retrieval or an edit operation, make sure you try to retrieve names that are not in the directory as well as names that are in the directory.
- If an entry has been changed, verify that the new number is retrieved.
- check that all new and edited entries are written correctly to the output file.

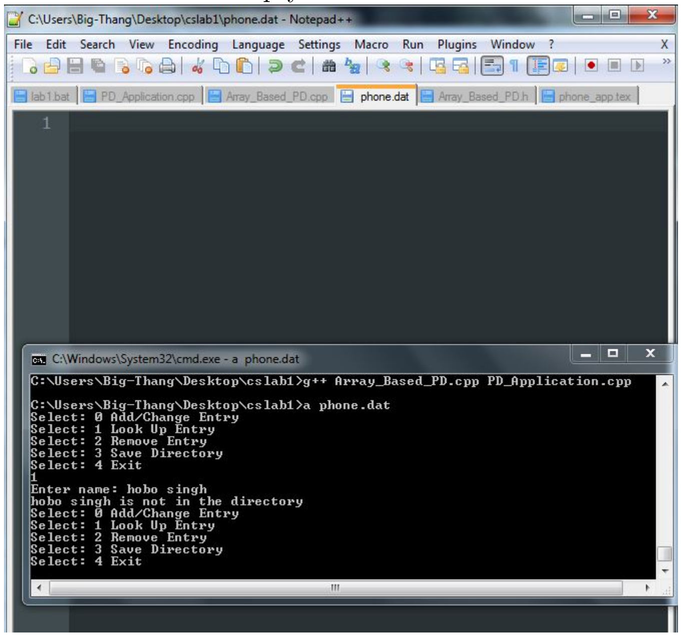
An important part of design is to create algorithms for each of the member functions. Here is an example for add_or_change_entry (from page 112).

Algorithm for Function add_or_change_entry

```
1. Call function find to see whether the name is in the directory.
2. if the name is in the directory
3.         Change the number using the set_number function of the Directory_ Entry.
4.         Return the previous value of the number.
   else
5.         Add a new entry using function add.
6.         Return an empty string.
```

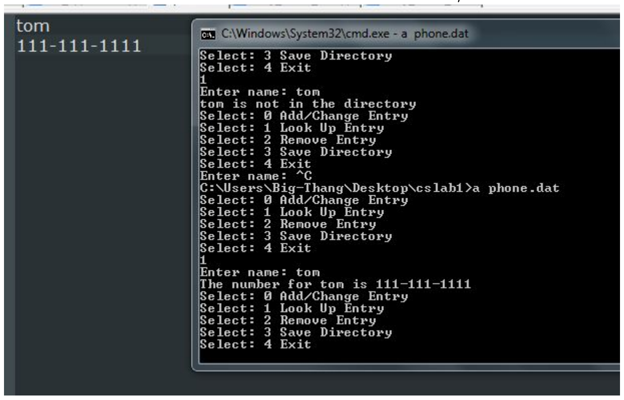
Note that we have identified another new function, `find`, for class Array_Based_PD.

This is the test empty


C:\Users\Big-Thang\Desktop\cslab1\phone.dat - Notepad++
File Edit Search View Encoding Language Settings Macro Run Plugins Window ?
lab1.bat
PD_Application.cpp
Array_Based_PD.cpp
phone.dat
Array_Based_PD.h
phone_app.tex
C:\Windows\System32\cmd.exe - a phone.dat

```
C:\Users\Big-Thang\Desktop\cslab1>g++ Array_Based_PD.cpp PD_Application.cpp

C:\Users\Big-Thang\Desktop\cslab1>a phone.dat
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
1
Enter name: hobo singh
hobo singh is not in the directory
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
```

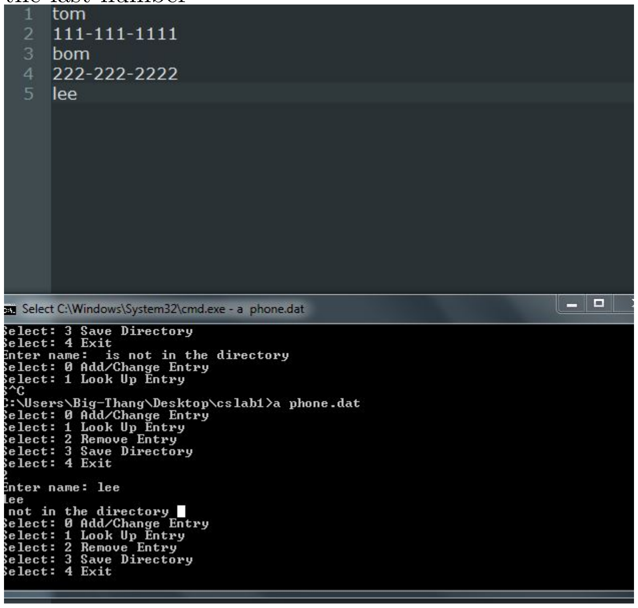
This is the test case with one name/ number



```
tom
111-111-1111
```

```
Select: 3 Save Directory
Select: 4 Exit
1
Enter name: tom
tom is not in the directory
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
Enter name: ^C
C:\Users\Big-Thang\Desktop\cslab1>a phone.dat
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
1
Enter name: tom
The number for tom is 111-111-1111
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
```

test case with odd names and numbers with no number and i checked if the lee is still a person, but if no number and only name present the person acts on the last number

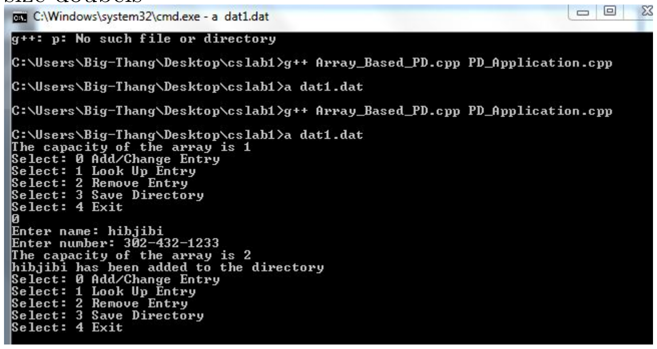


```
tom
111-111-1111
bom
222-222-2222
lee
```

Select C:\Windows\System32\cmd.exe - a phone.dat

```
elect: 3 Save Directory
elect: 4 Exit
nter name:  is not in the directory
elect: 0 Add/Change Entry
elect: 1 Look Up Entry
^C
:\Users\Big-Thang\Desktop\cslab1>a phone.dat
elect: 0 Add/Change Entry
elect: 1 Look Up Entry
elect: 2 Remove Entry
elect: 3 Save Directory
elect: 4 Exit

nter name: lee
ee
 not in the directory
elect: 0 Add/Change Entry
elect: 1 Look Up Entry
elect: 2 Remove Entry
elect: 3 Save Directory
elect: 4 Exit
```

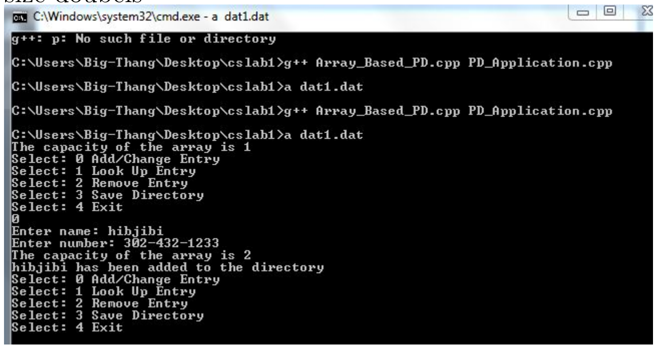
This test case basicly shows that even when you have a full array and you add a new entry you will be still able to add another contact because the array size doubels



C:\Windows\system32\cmd.exe - a dat1.dat

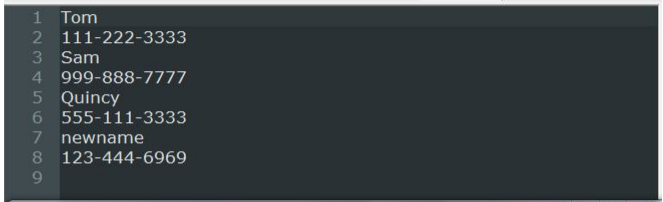
```
g++: p: No such file or directory

C:\Users\Big-Thang\Desktop\cslab1>g++ Array_Based_PD.cpp PD_Application.cpp

C:\Users\Big-Thang\Desktop\cslab1>a dat1.dat

C:\Users\Big-Thang\Desktop\cslab1>g++ Array_Based_PD.cpp PD_Application.cpp

C:\Users\Big-Thang\Desktop\cslab1>a dat1.dat
The capacity of the array is 1
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
0
Enter name: hibjibi
Enter number: 302-432-1233
The capacity of the array is 2
hibjibi has been added to the directory
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
```

This test case basicly shows that even when you have a full array and you add a new entry you will be still able to add another contact because the array size doubels

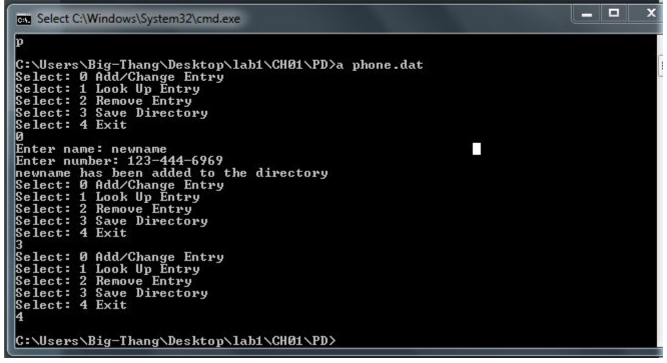


```
C:\Windows\system32\cmd.exe - a  dat1.dat
g++: p: No such file or directory

C:\Users\Big-Thang\Desktop\cslab1>g++ Array_Based_PD.cpp PD_Application.cpp

C:\Users\Big-Thang\Desktop\cslab1>a dat1.dat

C:\Users\Big-Thang\Desktop\cslab1>g++ Array_Based_PD.cpp PD_Application.cpp

C:\Users\Big-Thang\Desktop\cslab1>a dat1.dat
The capacity of the array is 1
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
0
Enter name: hibjibi
Enter number: 302-432-1233
The capacity of the array is 2
hibjibi has been added to the directory
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
```

I entered a name and changed the number and changed the number to something different and saved the changes. I checked the file and the file was updated



```
Tom
111-222-3333
Sam
999-888-7777
Quincy
555-111-3333
newname
123-444-6969

```



```
Select C:\Windows\System32\cmd.exe
p

C:\Users\Big-Thang\Desktop\lab1\CH01\PD>a phone.dat
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
0
Enter name: newname
Enter number: 123-444-6969
newname has been added to the directory
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
3
Select: 0 Add/Change Entry
Select: 1 Look Up Entry
Select: 2 Remove Entry
Select: 3 Save Directory
Select: 4 Exit
4

C:\Users\Big-Thang\Desktop\lab1\CH01\PD>
```

6 Implementation

Below is the code for the remove entry function This document has only a sample of the design and implementation of the case study in order to illustrate how we can use Latex to write about our programs as well as to write the code for them. There are ways to format effectively and to add diagrams to clarify ideas. Let's finish the code for this program first, test it, then later we will modify it to be an Employee directory (Project 1 on page 127)