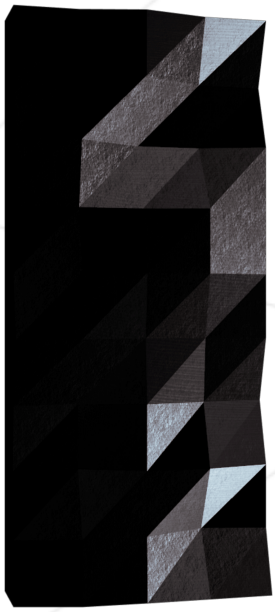

C++ - Module 00

Namespaces, classes, member functions, stdio streams, initialization lists, static, const, and some other basic stuff

Summary:
This document contains the exercises of Module 00 from C++ modules.

Version: 8

Chapter III

Exercise 00: Megaphone

	Exercise : 00
Megaphone	
Turn-in directory : *ex*00/	
Files to turn in : `Makefile, megaphone.cpp`	
Forbidden functions : `None`	

Just to make sure that everybody is awake, write a program that behaves as follows:

```
$>./megaphone "shhhhh... I think the students are asleep..."
SHHHHH... I THINK THE STUDENTS ARE ASLEEP...
$>./megaphone Damnit " ! " "Sorry students, I thought this thing was off."
DAMNIT ! SORRY STUDENTS, I THOUGHT THIS THING WAS OFF.
$>./megaphone
* LOUD AND UNBEARABLE FEEDBACK NOISE *
$>
```

Solve the exercises in a C++ manner.

Chapter IV

Exercise 01: My Awesome PhoneBook

	Exercise : 01
My Awesome PhoneBook	
Turn-in directory : $ex01/$	
Files to turn in : `Makefile, *.cpp, *.{h, hpp}`	
Forbidden functions : `None`	

Welcome to the 80s and their unbelievable technology! Write a program that behaves like a crappy awesome phonebook software.

You have to implement two classes:

- **PhoneBook**
  - It has an array of contacts.
  - It can store a maximum of **8 contacts**. If the user tries to add a 9th contact, replace the oldest one by the new one.
  - Please note that dynamic allocation is forbidden.
- **Contact**
  - Stands for a phonebook contact.

In your code, the phonebook must be instantiated as an instance of the **PhoneBook** class. Same thing for the contacts. Each one of them must be instantiated as an instance of the **Contact** class. You're free to design the classes as you like but keep in mind that anything that will always be used inside a class is private, and that anything that can be used outside a class is public.

`Don't forget to watch the intranet videos.`

On program start-up, the phonebook is empty and the user is prompted to enter one of three commands. The program only accepts ADD, SEARCH and EXIT.

- **ADD**: save a new contact
  - If the user enters this command, they are prompted to input the information of the new contact one field at a time. Once all the fields have been completed, add the contact to the phonebook.
  - The contact fields are: first name, last name, nickname, phone number, and darkest secret. A saved contact can't have empty fields.
- **SEARCH**: display a specific contact
  - Display the saved contacts as a list of **4 columns**: index, first name, last name and nickname.
  - Each column must be **10 characters** wide. A pipe character ('|') separates them. The text must be right-aligned. If the text is longer than the column, it must be truncated and the last displayable character must be replaced by a dot ('.').
  - Then, prompt the user again for the index of the entry to display. If the index is out of range or wrong, define a relevant behavior. Otherwise, display the contact information, one field per line.
- **EXIT**
  - The program quits and the contacts are lost forever!
- **Any other input is discarded.**

Once a command has been correctly executed, the program waits for another one. It stops when the user inputs EXIT.

Give a relevant name to your executable.

http://www.cplusplus.com/reference/string/string/ and of course http://www.cplusplus.com/reference/iomanip/

Chapter V

Exercise 02: The Job Of Your Dreams

	Exercise : 02
The Job Of Your Dreams	
Turn-in directory : *ex*02/	
Files to turn in : `Makefile, Account.cpp, Account.hpp, tests.cpp`	
Forbidden functions : `None`	

`Account.hpp, tests.cpp and the log file are available for download on the intranet page of the module.`

Today is your first day at *GlobalBanksters United*. After successfully passing the recruitment tests (thanks to a few *Microsoft Office* tricks a friend showed you), you joined the dev team. You also know the recruiter was amazed by how quickly you installed *Adobe Reader*. That little extra made all the difference and helped you defeat all your opponents (aka the other applicants): you made it!

Anyway, your manager just gave you some work to do. Your first task is to recreate a lost file. Something went wrong and a source file was deleted by mistake. Unfortunately, your colleagues don't know what `Git` is and use USB keys to share code. At this point, it would make sense to leave this place right now. However, you decide to stay. Challenge accepted!

Your fellow developers give you a bunch of files. Compiling `tests.cpp` reveals that the missing file is `Account.cpp`. Lucky you, the header file `Account.hpp` was saved. There is also a log file. Maybe you could use it in order to understand how the **Account** class was implemented.

You start to recreate the `Account.cpp` file. In only a few minutes, you code a few lines of pure awesome C++. After a couple of failed compilations, your program passes the tests. Its output matches perfectly the one saved in the log file (**except for the timestamps** which will obviously differ since the tests saved in the log file were run before you were hired).

Damn, you're impressive!

The order in which the destructors are called may differ depending on your compiler/operating system. So your destructors may be called in a reverse order.

You can pass this module without doing exercise 02.