**CPE 327 – Software Engineering**

**Lab Exercise 7: Refactoring Exercise**

In this exercise, you will study the code for a software system with multiple code modules written by someone else. You will modify this code so that it has better code quality, without changing the functionality or behavior. As discussed in Lecture 7, this process is called **refactoring**. The code you will be working with was written by Aj. Sally several years ago for the data structures class. This software implements a simple social network, using an adjacency-list version of a graph data structure. The application is discussed in more detail below.

The code we are giving you has been modified to deliberately introduce some “code smells”, that is, poor structure or design that could be improved by refactoring. In fact, even without these changes, the original code could probably be improved by thoughtful refactoring. There is no such thing as perfect code!

**What you have to do:**

1. Compile and link the original code, then experiment with running it until you understand how it behaves. We have provided a Makefile which you can use to build the application. If you don’t have a make tool, you can look at the last command in the Makefile, which links the executable, to see what code modules the system needs.
2. Examine the code, looking for opportunities to refactor it. These might include:

* Repeated code;
* Inconsistent code;
* Functions that are too long to be easily understood;
* Inappropriate intimacy, that is, code that “knows” too much about the code in another module;
* Freeloader modules, that is, modules that provide very little functionality and could be included in other modules;
* “God” modules, that is, modules that have low cohesion and try to do everything.

1. Each time you find a possible refactoring opportunity, modify the code to improve it. Then compile and link the modified application and test it to make sure that it still works (the same way as before).
2. **Very Important!** Every time you make a change to the code we have provided, you must put in a comment with the following form:

/\* ***6207050nnnn – Modified 2021-09-22 – Reason for the change \*/***

Replace the “nnnn” with your ID. Change the date to the date when you actually made the change. Explain briefly why you made the change (for instance, based on the list of problems above). For instance:

**VERTEX\_T \* pCurrent = NULL;**

**VERTEX\_T \* pAdjacent = NULL;**

**queueClear();**

**colorAll(WHITE);**

**/\* 62070501234 – Modified 2021-09-16 – Moved initialization loop to**

**separate colorAll function to shorten this one \*/**

We will search your code for comments of this form in order to give you a grade on the lab, so don’t forget to include them, and be sure they have the correct format.

1. Repeat the step above until you either don’t find any refactoring opportunities or you run out of time.

This exercise is due on FRIDAY 25 September at noon.

**What to submit:**

You should submit a ZIP file called ***6207050nnnn\_Lab7.zip***. This file should contains *all the code modules (.c and .h) in the final refactored system, whether you changed them or not*. Do not include object or executable files.

Your grade will be based on the following:

* Whether you followed instructions;
* Whether your modified code compiles and runs correctly;
* What changes you made along with your reasons for the changes.

There is no one right answer for this lab. We want to see whether you understand the concept of refactoring and whether you can identify really serious problems that can be fixed by refactoring.

This is quite a difficult lab, because it requires you to understand code written by someone else. However, this is a realistic situation that you will meet when you start to work in a software development team. ***If you have questions about the code, post them on the Facebook group for your sections.***

**About the Application**

The ***socialNetworkRF*** program implements a simple social network, similar to Facebook. It has the following capabilities:

1. Read a file that defines users and friend relationships into memory-based data structures. A sample file is shown later and will be included with the code.
2. Add a new user to the social network. User information includes a unique username, first name, last name and date of birth.
3. Print all the friends of a particular user. Note that in this social network, the “friend” relationship goes in both directions. That is, if A is a friend of B, B is also a friend of A.
4. Add friends for a particular user. This social network does not have the concept of a “friend request”. If you add B as a friend to A, this happens immediately without any need to “accept”.
5. Suggest friends for a particular user. Suggestions are based on people who are friends of that user’s existing friends.

The name of the input file is a command line argument. That is, you must run the program as follow:

**./socialNetworkRF inputnetwork.txt**

The input file is a text file with two different kinds of lines. A line that begins with the word FRIEND defines a friend relationship between two users. Any other type of line defines a user, with four fields: username, first name, last name and birthday.

The username must be unique. The program checks this.

A sample input file is shown below. Note that there is no limit to the number of lines or users that could be in an input file.

***Sample Input File***

The text below shows the contents of the file ***networkinput.txt*** which is provided as a sample.

marisa Marisa Johnson 12-01-1976

lorelei Lori Rogers 01-01-2000

walrus Reuben Saunders 09-08-2001

jerry Jerry Marshall 31-02-2000

looo Louis Johnson 03-12-1998

maryMe Mary Anderson 11-08-2001

babydoll Linda Smith 14-09-1997

roberta1 Roberta Ho 08-08-1988

sasparilla Suzy Sommers 17-07-2003

chaosrules Mark Marshall 16-04-1978

cathy Catherine Holmes 05-11-2005

ricky Richard Masterson 30-07-2000

ricky2 Rick Rogers 18-10-2000

martham Martha Murray 31-12-2002

arnie Arnold Schwartznegger 01-02-1964

joe Joseph Howard 13-05-1999

lottie Charlotte Smith 14-09-2007

babalu Sidney Carter 01-12-2001

huggybear Jenny Jones 03-09-1977

abby123 Abigail Jackson 17-07-2000

rogerRabbit Roger Rabbit 09-07-1955

roberta2 Roberta Henderson 11-11-2000

whoareyou Carl Doherty 10-06-1995

morrisTheCat Morris Frank 01-01-1989

babalu Sidney Carter 01-12-2001

betty Betty Boop 22-01-1920

brendaB Brenda Baker 24-07-1989

alice Alice Smith 19-10-2000

alice2 Alice Ransom 09-05-2002

pooh Patty Parker 07-12-2004

billTheCat William Opper 10-04-1963

brandon Brandon Potter 14-11-2001

henry123 Henry Knight 18-01-1993

jilljones Jill Jones 02-12-1997

markMMason Mark Mason 09-11-1984

prudence Prudence Dorrington 17-03-2002

kathy Katherine Loon 19-02-1983

kendall Kendall Craik 15-05-2003

mmorgan Marilyn Morgan 18-09-1991

FRIEND maryMe babydoll

FRIEND maryMe roberta1

FRIEND sasparilla chaosrules

FRIEND chaosrules huggybear

FRIEND huggybear cathy

FRIEND cathy maryMe

FRIEND arnie lottie

FRIEND arnie babalu

FRIEND babydoll billTheCat

FRIEND billTheCat rogerRabbit

FRIEND whoareyou brandon

FRIEND markMMason cathy

FRIEND roberta2 betty

FRIEND roberta2 alice

FRIEND roberta2 alice2

FRIEND brandon henry123

FRIEND jilljones chaosrules

FRIEND jilljones alice2

FRIEND mmorgan prudence

FRIEND prudence arnie

FRIEND lottie jilljones

FRIEND pooh betty

FRIEND huggybear abby123

FRIEND abby123 alice2

FRIEND jerry looo

FRIEND sasparilla babalu

FRIEND ricky roberta2

FRIEND ricky2 martham

Before you start refactoring the code, you should experiment with the program as provided, so that you understand how it works. Be sure to try all the options.