ECS 60 **Programming Assignment #4, 50 points (2 hours coding)** **Winter 2015**

Due: Wednesday, February 25th at 11:59pm to p4 directory of cs60
Primary file names: familytree.cpp, familytree.h Executable name: family.out Makefile name: Makefile
Format of authors.txt: author1_email author1_last_name, author1_first_name
author2_email author2_last_name, author2_first_name

This program will read information about a people and their children from a file, and then provide answers to queries about the resulting family trees. The program takes a file name as command line arguments.

You are to handin authors.txt, and all files upon which your program is dependent, except CPUTimer.h, familyRunner.cpp, familyRunner.h. The grading script will copy those files into your directory. You will find CreateFamily.out, data files, familyRunner.*, barebones familytree.*, and my family.out in ~ssdavis/60/p4.

Here are the specifications:

1. People
   - 1.1. Each person is uniquely specified with year of birth <short>, last name <char[12]>, first name <char[12]>, and gender <char>
   - 1.2. The sources for names are LastNames.csv with 60 entries, MaleNames.csv with 31 entries, FemaleNames.csv with 45 entries.
   - 1.3. Each person has at most one spouse, and may have 0 to 7 children.
2. Family files
   - 2.1. These files are read by the FamilyRunner code, so you do not have to worry about their format. Nonetheless, if you are curious, here is the format.
   - 2.2. The first line of the files had the number of generations, number of starting pairs, and number of queries.
   - 2.3. The next series of lines are the queries. Each line of these lines contains the specifications of two people followed by the youngest common ancestor of the two people. If they have no common ancestors, then the two people are followed by -1.
   - 2.4. The last series of lines contains a person description followed by a list of people in his/her immediate family.
     - 2.4.1. If there is a second person listed, then it will be the spouse of the person, else family.spouse.year = -1
     - 2.4.2. Any people listed after the spouse will be children of the couple listed in order of year of birth.
   - 2.5. The lines in are ordered by year of birth in descending order.
3. answers array.
   - 3.1. Your Family::queries() method fills in the answers array with the specification of the youngest common ancestor of the two people in the corresponding query. If they do not have a common ancestor, then set the DOB to -1.
4. Grading
   - 4.1. (20 points) Creates a correct answers array for a family file of at least 10 generations. If a program does not correctly answer every query, then it will zero for the entire assignment.
   - 4.2. (15 points) CPU time for entire program in response to a very large family file with only one query. In essence this tests for the time of insertions. Score = min (18, 15 * Sean's CPU Time / Your CPU Time);
   - 4.3. (15 points) CPU time for entire program in response to a large family file with at least 10,000 queries. Score = min (18, 15 * Sean's CPU Time / Your CPU Time);
   - 4.4. Programs must be compiled without any optimization options. You may not use any precompiled code, including the STL and assembly.

```
class Person
{
public:
  short year;
  char lastName[12];
  char firstName[12];
  char gender;
}; // class Person

class Family
{
public:
  Person person;
  Person spouse;  // spouse.year = -1 for none
  Person children[7];
  short childCount;
};

class Query
{
public:
  Person person1;
  Person person2;
};
```

```
int main(int argc, char* argv[])
{
  ifstream inf(argv[1]);
  int generations, pairs, queryCount, familyCount;
  char dummy;
  inf >> generations >> dummy >> pairs >> dummy >> queryCount;
  inf.ignore(10, '\n');
  Family *families = new Family[30000];
  Query *queries = new Query[queryCount];
  Person *answers = new Person[queryCount];
  Person *answerKeys = new Person[queryCount];
  readQueries(inf, queries, answerKeys, queryCount);
  familyCount = readFamilies(inf, families);
  CPUTimer ct;
  ct.reset();
  FamilyTree *familyTree = new FamilyTree(families, familyCount);
  delete [] families;
  familyTree->runQueries(queries, answers, queryCount);
  cout << "CPU Time: " << ct.cur_CPUTime() << endl;

  for(int i = 0; i < queryCount; i++)
    if(answerKeys[i].year == -1)
      if(answers[i].year != -1)
        cout << "You found an ancestor when there was none on query #"  << i << endl;
      else
        if(answers[i].year != answerKeys[i].year
          || strcmp(answers[i].lastName, answerKeys[i].lastName) != 0
          || strcmp(answers[i].firstName, answerKeys[i].firstName) != 0
          || answers[i].gender != answerKeys[i].gender)
          cout << "Disagreement on query #" << i << endl;

  return 0;
}  // main()
```

```
[ssdavis@lect1 Create]$ CreateFile.out
Number of generations (1 - 20): 3
Number of starting pairs (1 - 60): 2
Number of queries (1 - 100000): 5
Seed: 2
[ssdavis@lect1 Create]$ cat Family3-2-5-2.csv
3,2,5
1887,Jarvinen,Clyde,M,1920,Le,Monica,F, -1
1887,Jarvinen,Clyde,M,1889,Le,Bernardo,M, -1
1887,Jarvinen,Clyde,M,1887,Jarvinen,Clyde,M,1887,Jarvinen,Clyde,M
1913,Le,Clyde,M,1887,Jarvinen,Clyde,M, -1
1889,Le,Bernardo,M,1887,Jarvinen,Clyde,M, -1
1928,Le,Jula,F
1923,Le,Nicholas,M
1920,Le,Monica,F
1917,Le,William,M
1913,Le,Clyde,M
1889,Le,Bernardo,M,1888,Hsu,Clairissa,F,1913,Le,Clyde,M,1917,Le,William,M,1920,Le,Monica,F,1923,Le,Nicholas,M,1928,Le,Jula,F
1888,Hsu,Clairissa,F,1889,Le,Bernardo,M,1913,Le,Clyde,M,1917,Le,William,M,1920,Le,Monica,F,1923,Le,Nicholas,M,1928,Le,Jula,F
1887,Jarvinen,Clyde,M,1886,Howell,Amanda,F
1886,Howell,Amanda,F,1887,Jarvinen,Clyde,M
[ssdavis@lect1 Create]$ family.out Family3-2-5-2.csv
CPU Time: 0.000255
[ssdavis@lect1 Create]$

[ssdavis@lect1 p4]$ family.out Family10-60-10000-5.csv
CPU Time: 0.209052
[ssdavis@lect1 p4]

[ssdavis@lect1 private]$ family.out Family3-3-10-1.csv
CPU Time: 0.0003
Disagreement on query #3
Disagreement on query #7
Disagreement on query #8
[ssdavis@lect1 private]$
```