Assignment #1 – Review Programming in C++
Due: Sunday, 04/10/16, 11:59pm

Grading: EVERY assignment in this course is graded by demoing your work for 10 minutes with a TA. You are required to meet with a TA within one week after the due date to demo. You are penalized for failure to see a TA within the week or missing a scheduled appointment. In either case, if you are within 1 day (24 hours) of the deadline, you lose 10 points. If you are within 7 days (1 week) of the deadline, then you lose 25 points, anything outside of a week from the deadline to demo is an automatic 50 point deduction. Your job is to convince the TA that your program works correctly, i.e. show your TA how to use/break your program☺

During your time in programming, you will undoubtedly run into a situation where you will need to go through information more than once or need to perform the same task over and over. The way we as programmers accomplish this is through the use of loops and functions.

(90 pts) Implementation: Problem Statement
You will write a program to compare state and county information. You must have the following structs in your program.

```
struct county {
   string name;  //name of county
   string *city;  //name of cities in county
   int cities;  //number of cities in county
   int population;  //total population of county
   float avg_income;  //avg household income
   float avg_house;  //avg household price
};
struct state {
  string name;  //name of state
  struct county *c;  //name of counties
  int counties;  //number of counties in state
  int population;  //total population of state
 };
```

You will receive the number of states and filename from the user as command-line arguments. The number supplied with the –s option is the number of states to be created and the text following the –f option is the filename with the state/county information: **a.out –s 2 –f states1.txt**

This would create a dynamic array of 2 states on the heap, and you would read the rest of the information about the state and counties from a file. Each line in the file will contain the information for each state and county in the following order:

State_name state_pop #_county
county_name county_pop county_income county_house #_cities city_name

Example:
Oregon 1000000 2
Benton 53000 100000 250000 1 Corvallis
Lane 80000 50000 150000 2 Eugene Springfield
South_Carolina 1000000 2
Anderson 80000 100000 80000 2 Anderson Pendleton
Pickens 50000 50000 20000 2 Clemson Pickens

Your program must define the following functions, with the exact prototypes:

```
bool is_valid_arguments(char *[], int);
state * create_states(int);
void get_state_data(state *, int, ifstream &);
county * create_counties(int);
void get_county_data(county *, int, ifstream &);
void delete_info(state *, int);
```

In addition to these functions above, you need to determine the other functions you will need to print information answering the following information:

- the state with the largest population,
- the county with the largest population,
- the counties with an income above a specific amount, (You must get input from the user for this!!!)
- the average household cost for all counties in each state,
- the states in sorted order by name,
- the states in sorted order by population,
- the counties within states sorted by population,
- the counties within states sorted by name.

You need to separate your files into interface and implementation and create a **Makefile** to handle the compilation. Create a **state_facts.h**, which has the struct type for states and counties, as well as the function declarations for your program. Now, separate your function definitions into a **state_facts.cpp** file and your main function into a **run_facts.cpp** file. Now, create a makefile that will create a state_facts executable game and clean your files.

Your program must be able to:

- Print a usage message to the user when too few arguments are supplied or when the options are not –s or -f. You do not need to recover from this, just handle by printing a message.
- Print an error message and recover, when the user doesn't supply positive, non-zero integer for the states value.

- Print an error message and recover, when the player doesn’t supply a valid filename to open.
- Provide the stats for the states.
- Continue to ask for new states and new filename. Make sure you do not have a memory leak!!!!!

(10 pts) **Program Style/Comments**
In your implementation, make sure that you include a program header in your program, in addition to proper indentation/spacing and other comments! Below is an example header to include. Make sure you review the style guidelines for this class, and begin trying to follow them, i.e. don’t align everything on the left or put everything on one line!
http://classes.engr.oregonstate.edu/eecs/spring2016/cs162-001/162_style_guideline.pdf

```
/******************************************************
** Program: run_stats.cpp
** Author: Your Name
** Date: 04/08/2016
** Description:
** Input:
** Output:
******************************************************/
```

Electronically submit your C++ program (**.h, .cpp, and Makefile files**, not your executable!!!) and your test files as a tarred archive by the assignment due date, using TEACH.

You must tar these files together using the following command:

```
tar –cvf assign1.tar facts.h facts.cpp run_facts.cpp Makefile test.txt
```

****NOTE:** The easiest way to upload your program from ENGR to TEACH is to map a network drive to your home directory on ENGR. Mac or Windows, See:
http://engineering.oregonstate.edu/computing/fileaccess/