Assignment 1

Understanding the problem:
For this assignment the program should in a broad sense accept a txt file with data about states and counties, store the data in structs, and the be able to access the data based on a certain set of parameters.
The data files are assumed to be correct, this program will have no error handling in terms of the file but will for command line inputs. The data will come in the following format; the number of states will be given by the command line argument. This can be used to create an array of state structs. The first line of the file will be a state with its name, information, and the number of counties accompanying it in the file. The following number of lines will be the counties associated with the state from above, there should be the same number of counties as listed in the state. The state's name and data should be saved and an array of county structs created based on the number of counties. Iterating through the counties array, the program will store the name and information of each county. For each county an array will be created based on the number of cities, which will be used to store a dynamic number of cities.
This program relies heavily on the use of dynamic memory allocation during runtime, memory management is critical.
It is assumed that the command line arguments could be false and will be error checked.
It is assumed that the data file is correct and will not be error checked.
It is assumed that if the command line argument is correct syntactically, it is correct in the number of states found in the file.

int main(int argc, char **argv)
Purpose:
Main is the driver function for the whole program. It will call the command line arguments using external functions, call for the storage of the state data into the structs, and finally present options for the user to sort the data they supplied.

Pseudo code:
```
int main(int argc, char **argv)
        pass is_valid_argument arc and argv
        pass and store create_states the number of states from command line
        pass get_state_data the stored struct array from create_states as well as
                        number of states and the file
        pass info_sort the struct array and file
        call delete_info
        return 0
```

bool is_valid_arguments(char*[] argv, int argc)
Purpose:
This function is used to verify that the command line arguments are in fact correct. If the command line arguments are missing the correct number of flags and correct flag names then the function will inform the user and exit the program. If the flags are correct but their values are incorrect it will re-prompt without exiting. The -s flag is expected to be a positive non-zero integer, if it is not there will be a re-prompt. If the input is a positive non-zero it is assumed to be correct in term of the number of states in the file. The -f flag expects a valid file name. If the file cannot be found there will be a re-prompt.

Psseudo code:
```
bool is_valid_arguments(char*[] argv, int argc)
```

if there are not the correct flags
    inform the user that they need to re-run the program correctly
    exit the program

if the -s flag is not positive and non-zero
    continue to re-prompt until it is valid

if the -f flag file cannot be found
    continue to re-prompt until it is valid

state * create_states(int num_states)
Purpose:
This function should create an array of state structs based on the number of states inputted in the command line argument. The function will accept the number as an input and return the array.

Pseudo code:
state * create_states(int num_states)
    create an array of state structs on the heap of num_states long
    return the address of this array

void get_state_data(state * states , int curr_state, ifstream &);
Purpose:
This function should accept the array of state structs as well as the index for the state data that will be stored and the file contents. The function should read the relevant line from the file, store the corresponding data. It will then call a function to create the array of counties based on the number given in the file. It should then call the get_county_data function with the array to populate it with corresponding county data.
Pseudo code:
void get_state_data(state * states , int curr_state, ifstream &)
    read in the relevant line of data regarding the current state
    at index curr_state in states array store the name and other data
    struct county *c = create_counties(number_of_counties)
    call get_county_data(&c, int number_of counties,ifstream &)

county *create_counties(int num_counties)
Purpose:
This function will create an array of county structs based on the supplied number of counties. This function will return the pointer to the array.

Pseudo code:
county *create_counties(int num_counties)
    create an array of state structs on the heap of num_counties long
    return the address of this array

void get_county_data(county * c, int num_counties ,ifstream &)
Purpose:
This function should accept an already created array of county struct as well as the number of counties (length of array) and the file data. The function will iterate through the file, storing each line as a new

county. Based on the data the function should also create an array of strings for the cities, and then store the city names within it.

Pseudo code:
void get_county_data(county * c, int num_counties ,ifstream &)
    store the name of the county
    store the county population
    store county income
    store county house price
    store the number of cities
    create an array of strings based on the number of cities
    each city name to the array of cities

void delete_info(state ** states , int num_states)
Purpose:
This function should be run to delete all data on the heap to prevent a memory leak. To avoid leaving some data inaccessible the function can't just delete the states array, it must iterate through and delete each sub array first. It should iterate over the states, within which it should iterate over each county, within which each city string array will be deleted. Then each county array should be deleted. Finally, the states array can be deleted.
Sudo code:
void delete_info(state ** states , int num_states)
    for each state in states
        for each county in c
            delete cities array
        delete county array
    delete states array

void largest_state(state * states)
Purpose:
The purpose of this function is to find and print the state with the largest population. The function simply loops through the structs, recording the highest value it find, and at the end printing the state associated with the highest number.
Pseudo code:
void largest_state(state * states)
    int largest_pop = 0
    int largest_index = 0
    for each state in states
        if state population largest than largest_pop
            save the index to largest_index

    print out the name and population of the largest index

void largest_county(state * states)
Purpose:
This function should loop through each state and each county associated with the state, recording the county with the largest population. Two for loops will be required and the function will have to store the state and county's indexes.

Pseudo code

```
void largest_county(state * states)
        largest_county =0
        county_index = 0
        state_index = 0
        for each state in states
                for each county in c
                        if county is larger that largest_county
                                set largest county to new population
                                set county index to current index
                                set state index to current stat index

        print the county, state, and county population based on index
```

void county_above_income(state * states)

Purpose:

This function should print every county with income above a user inputted integer value. It will do so by using a double nested for loop similar to the largest county function. If the input is not an integer it will re-prompt.

Pseudo code:

```
void county_above_income(state * states)
        prompt the user for input
        save integer input
        for each state in states
                for each county in c
                        if the county income is above saved input
                                print county information
```

void avg_count_income(state * states)

Purpose:

This function will calculate the average county income for each state using double nested for loops. While looping through each county the function will keep a tally of added up county incomes. At the end of the counties for that state the function will divide it by the number of counties and print the result before moving on to the next state with new tally variables.

Pseudo code:

```
void avg_count_income(state * states)
        for each state in states
                int total_tally = 0
                for each county in c
                        add county population to total_tally
                divide total_tally by the number of counties
                print the result
```

void states_alph(state * states)

Purpose:

This function will sort and print the states in alphabetical order. This will use a bubble sort to re-order the entire array and then print it.

Pseudo code:

```
void states_alph(state * states)
    while unsorted
        for each state in states
            if state.name > state.name+1
                swap = state.name
                state.name = state.name+1
                state.name+1 = swap
    loop through states and print each state information
```

void counties_alph(state * states)

Purpose:

This function pass through each state, sort their counties alphabetically, and then print each state. It uses a bubble sort to order the county arrays for each state and then prints them

Pseudo code:

```
void counties_alphstate * states)
    for each state in states
        for each county in c
            swap = county.name
            county.name = county.name+1
            county.name+1 = swap

        print state name
        for each county in c
            print county info
```

void states_by_pop(state * states)

Purpose:

This function will sort the states by their total population. It uses a bubble sort to re-order the entire state array, then printing each state and their info

Pseudo code:

```
void states_by_pop(state * states)
    while unsorted
        for each state in states
            if state.avg_income > state.avg_income+1
                swap = state.avg_income
                state.avg_income = state.avg_income+1
                state.avg_income+1 = swap

    loop through states and print each state information
```

void counties_by_pop(state * states)

Purpose:

This function will use a bubble sort to re-order the county array for each state according to descending average household income. After sorting each county array it will print the counties and the perspective state.

Pseudo code:

```
void counties_by_pop(state * states)
        for each state in states
                while unsorted
                        for each county in c
                                if county.avg_income > county.avg_income+1
                                        swap = county.avg_income
                                        county.avg_income = county.avg_income+1
                                        county.avg_income+1 = swap

                print state name
                for each county in c
                        print county info
```

Testing table:

Input	Expected output	Actual Output
a.out -s 2 -f states.txt (valid file)	Program should accept and store data	
a.out -s 0 -f states.txt (valid file)	Program should re-prompt for states	
a.out -s 4 -f statesas.txt (invalid file)	Program should re-prompt for file name	
a.out -f states.txt (valid file)	Inform user missing flag, exit program	
a.out -s 2	Inform user missing flag, exit program	
a.out -s -1	Inform user missing flag, exit program	
a.out -s 3 -f	Program should re-prompt for file name	
a.oout -s 100	Program won't run	
0 (above certain income)	Prints all counties	
-1	Prints all counties	
100000	Prints all counties with income above 100000	
Ab	Re-prompt for integer	