CS 162 - Lect 1 - Rec 16
January 8th, 2018
Nick Olson

Assignment 1, Economic Data Viewer

Understanding the Problem:

This problem is broken up into a few distinct parts. First, signing up for Piazza, which is fairly self explanatory. Second, the required structure of the project when it comes to files and the interactions between these files. Third, the programs requirements in terms of file reading, data manipulation, and data presentation, all while properly managing memory. The structure of the project files is explicit in the assignment statement. I will need to seperate my source code into at least three distinct .cpp or .hpp files that are (1) the implementation of the methods that allow the functionality of the program, (2) the header file with all struct definitions and function prototypes, and (3) the interface which is the main method that uses my modular methods to accomplish the specific task of exploring the data to the assignments requirements. Although the implementation would likely be the only file with many methods requiring fine modularization as per instruction, all three files should adhere to the style guidelines of the course, especially when it comes to complete and consistent headers for the files and each function, as well as adequate use of whitespace to increase readability. The assignment is also clear that there should be no files in the github repository other than these source code files, the README.md file, and the makefile for compilation. For the specific requirements of exploring data, I will need to implement a methods to search for and print specific state or county structs that have either the highest or the lowest of a given data-point, as well as sort the list of states or counties in various ways. Finally I need to be able to allow the user to “select a state” and look at its counties. Because the lists of states and counties may be variable length, I’m assuming I should implement some way to search for these states by name, rather than the user scrolling through lists and inputting indices.

Devise a Plan:

explore_econdata.cpp		
	main method	
		Takes argc and argv[] to take in command-line argument.
		Passes this stream to open_file() to open the passed file dir to the stream.
		Pass the stream to get_num_states() to record how many states were read.
		Create list of states by passing the number of states to allocate_states()
		Populate the list of states by passing all data to read_state_data()
		Launch the menu_system() to allow for exploration of the data.
		(When menu is done) pass data to free_state_data() to delete all new objects.
		Close the file stream
		return

econdata.cpp		
	open file	
		Takes the argument data from main method and the filestream
		Opens the file if the parameters are valid. If it fails or they aren't valid, returns an error code.
	get number of states	
		Takes the file stream
		Reads the first line of the file and returns the number of states printed there
	allocate states	
		Takes the number of states

		Returns a new list of state structs of the passed length.
	read state data	
		Takes the state list, number of states, and the file stream
		Reads each line, word by word, putting data into a new state struct's fields. When it comes to the counties, execute allocate_counties() with the number of counties read from the file, and read_county_data() with the relative data. Filling this list.
	allocate counties	
		Takes the number of counties in a state
		Returns a new list of county structs of the passed length.
	read county data	
		Takes the county list, the number of counties, and the file stream
		Fills the counties fields by reading the components of each line of the filestream for the number of counties listed.
	free state data	
		Takes the list of states and the number of states
		For each state, delete its county list, then delete the entire state list.
	menu system	
		Takes the list of states and the number of states
		Displays a menu system, using user input and switch statements to execute various small exploration methods described in the headers below.

I like to list out my file and function headers / comments before I implement, so here they are for the econdata.cpp file:

```cpp
/*
** Program:      econdata.cpp - Economic data viewer function implementation file.
** Author:       Nick Olson
** Date:         01/09/2019
** Description:          This program implements functions that read data from a user-provided
**               data file that is passed from a command-line argument, and gives the
**               user tools to interpret the data.
*/



/*
** Function:     open_file
** Description:          Opens the file the user specified in their command-
**                      argument. Returns error codes otherwise.
** Parameters:           int argc:
**                      Number of arguments passed.
**               char * argv[]:
**                      Array of arguments passed, starting with program
**                      filename at argv[0].
**               std::ifstream data_file:
**                      The ifstream to open the file into.
** Return Value:    Returns 0 for successful execution, 1 for argument
**                      error, 2 for a file read error, etc.
** Pre-Conditions:    File is unopened.
** Post-Conditions:    File has been opened.
*/
int open_file(
  int argc,
  char * argv[],
  std::ifstream & data_file);

/*
** Function:     get_num_states
** Description:          Gets the number of states contained in the data
**                      file on the first line, returns it.
** Parameters:           std::ifstream data_file:
**                      The filestream the data is opened into.
** Return Value:    Returns the number of states in the data file,
**                      or -1 in the event of an error.
** Pre-Conditions:    The data_file filestream is initialized and
**                      the file is already open; file correct;
**                      the stream has not been accessed yet.
** Post-Conditions:    The filestream is onto the first state's line.
*/
int get_num_states(
  std::ifstream & data_file);


/*
** Function:     allocate_states
** Description:          Allocates an array of the passed number of states.
** Parameters:           int num_states:
**                      The number of states in the data file.
** Return Value:    Returns an array of type struct state with length
**                      of num_states.
** Pre-Conditions:    num_states >= 0
*/
struct state * allocate_states(
  int num_states);


/*
```

```
** Function:     read_state_data
** Description:          Reads the data from the passed data_file fstream
**                       into the passed passed state array. Leaving
**                       the fstream at the end of the file.
** Parameters:           struct state * state_list:
**                       The list to populate with the states in the
**                       filestream.
**               int num_states:
**                       The number of states in the filestream.
**               std::ifstream & data_file:
**                       The filestream containing the contents of the
**                       data file passed to the program.
** Pre-Conditions:    num_states >= 0, data_file is at the line about
**                       the first state.
** Post-Conditions:    state_list is populated with all the states in
**                       the data_file (and the counties for each state).
*/
void read_state_data(
  struct state * state_list,
  int num_states,
  std::ifstream & data_file);


/*
** Function:     allocate_counties
** Description:          Allocates an array of the passed number of counties.
** Parameters:           int num_counties:
**                       The number of counties in the data file.
** Return Value:    Returns an array of type struct county with length
**                       of num_counties.
** Pre-Conditions:    num_counties >= 0
*/
struct county * allocate_counties(
  int num_counties);


/*
** Function:     read_county_data
** Description:          Reads the data from the passed data_file fstream
**                       into the passed passed county array. Leaving
**                       the fstream at the next county's section or
**                       the end of the file.
** Parameters:           struct county * county_list:
**                       The list to populate with the counties in the
**                       filestream.
**               int num_counties:
**                       The number of counties in the filestream.
**               std::ifstream & data_file:
**                       The filestream containing the contents of the
**                       data file passed to the program.
** Pre-Conditions:    num_counties >= 0, data_file is at the line about
**                       the first county.
** Post-Conditions:    county_list is populated with all the counties in
**                       the given state.
*/
void read_county_data(
  struct county * county_list,
  int num_counties,
  std::ifstream & data_file);


/*
** Function:     free_state_data
** Description:          Releases all data associated with the states and
**                       subsequent counties contained within the given
**                         state_list array.
** Parameters:              struct state * state_list:
**                         The array of states to be processed.
**                 int num_states:
**                         The number of states in the array.
** Pre-Conditions:     state_list is not empty (if num_states > 0), while
**                         num_states >= 0.
** Post-Conditions:     All data related to the states in the passed array
**                         or the counties contained within is released,
**                         avoiding memory leaks.
*/
void free_state_data(
  struct state * state_list,
  int num_states);


/*
** Function:       menu_system
** Description:            System of menus that gets what the user would like
**                         to do with the data to better view it, allows
**                         the user to either exit or repeat operations.
** Parameters:              struct state * state_list:
**                         The list of states to be sorted through.
**                 int num_states:
**                         The number of states represented in the list.
** Pre-Conditions:     state_list is not empty, all setup has been run.
*/
void menu_system(
  struct state * state_list,
  int num_states)


/*
** Function:       find_state
** Description:            Returns the struct state with the highest/lowest of
**                         a number of possible fields, all determined by
**                         the parameters of the function.
** Parameters:              struct state * state_list:
**                         The list of states to be searched through.
**                 int num_states:
**                         The number of states in the state_list.
**                 std::string search_type:
**                         The type of search to be done (searching for the
**                         "min" or the "max" of a given field).
**                 std::string field_name:
**                         The field to be considered in the search, can be
**                         "med_income" or "unemployed_2015".
** Return Value:     Returns the state found with the highest/lowest of
**                         the requested type. Leaves all params unchanged.
** Pre-Conditions:     state_list in non-empty.
*/
struct state find_state(
  struct state * state_list,
  int num_states,
  std::string search_type,
  std::string field_name);


/*
** Function:       find_county
** Description:            Returns the struct county with the highest/lowest of
**                         a number of possible fields, all determined by
**                         the parameters of the function.
** Parameters:              struct county * county_list:
**                         The list of counties to be searched through.
**               int num_counties:
**                      The number of counties in the county_list.
**               std::string search_type:
**                      The type of search to be done (searching for the
**                      "min" or the "max" of a given field).
**               std::string field_name:
**                      The field to be considered in the search, can be
**                      "med_income" or "unemployed_2015".
** Return Value:    Returns the county found with the highest/lowest of
**                      the requested type. Leaves all params unchanged.
** Pre-Conditions:    county_list in non-empty, unless num_counties is 0.
*/
struct county find_county(
  struct state state_to_expand,
  std::string search_type,
  std::string field_name);


/*
** Function:     compare_state_employment
** Description:          Returns whether the first state passed has a smaller
**                      or more negative change in its unemployment from
**                      2007 to 2015.
** Parameters:           struct state first_state:
**                      The first state to be compared.
**                      struct state second_state:
**                      The second state to be compared.
** Return Value:    Returns whether the change in unemployment is less
**                      in the first param than the second as a bool.
** Pre-Conditions:    The states are both initialized fully.
*/
bool compare_state_employment(
  struct state first_state,
  struct state second_state);


/*
** Function:     compare_state_income
** Description:          Returns whether the first state passed has a lower
**                      median income value than the second.
** Parameters:           struct state first_state:
**                      The first state to be compared.
**                      struct state second_state:
**                      The second state to be compared.
** Return Value:    Returns whether the median income is less in the
**                      first param than the second as a bool.
** Pre-Conditions:    The states are both initialized fully.
*/
bool compare_state_income(
  struct state first_state,
  struct state second_state);


/*
** Function:     compare_county_employment
** Description:          Returns whether the first county passed has a smaller
**                      or more negative change in its unemployment from
**                      2007 to 2015.
** Parameters:           struct county first_county:
**                      The first county to be compared.
**                      struct county second_county:
**                      The second county to be compared.
** Return Value:    Returns whether the change in unemployment is less
**                      in the first param than the second as a bool.
** Pre-Conditions:    The counties are both initialized fully.
```

```
*/
bool compare_county_employment(
  struct county first_county,
  struct county second_county);


/*
** Function:     compare_county_income
** Description:          Returns whether the first county passed has a lower
**                      median income value than the second.
** Parameters:           struct county first_county:
**                      The first county to be compared.
**                      struct county second_county:
**                      The second county to be compared.
** Return Value:    Returns whether the median income is less in the
**                      first param than the second as a bool.
** Pre-Conditions:    The counties are both initialized fully.
*/
bool compare_county_income(
  struct county first_county,
  struct county second_county);


/*
** Function:     display_state_data
** Description:          Prints all the stored data about a state to cout.
** Parameters:           stuct state state_to_display:
**                      The struct state to display.
**               int indentation_level:
**                      How indented the data should be (single indent
**                      is two spaces in this context).
** Pre-Conditions:    The state is initialized fully.
*/
void display_state_data(
  struct state state_to_display,
  int indentation_level);


/*
** Function:     display_county_data
** Description:          Prints all the stored data about a county to cout.
** Parameters:           stuct county county_to_display:
**                      The struct county to display.
**               int indentation_level:
**                      How indented the data should be (single indent
**                      is two spaces in this context).
** Pre-Conditions:    The county is initialized fully.
*/
void display_county_data(
  struct county county_to_display,
  int indentation_level);


/*
** Function:     search_for_state
** Description:          Finds and returns the first state with the given
**                      name found in the passed array of states.
** Parameters:           struct state * state_list:
**                      The list of states to search through.
**               int num_states:
**                      The number of states referenced in state_list.
**               std::string search_key:
**                      The keyword to search for in the state_list.
** Return Value:    Returns the first struct state found in the given
**                      state_list that contains the passed keyword.
** Pre-Conditions:    state_list is not empty, the states are all fully
**                       initialized with names.
*/
struct state search_for_state(
  struct state * state_list,
  int num_states,
  std::string search_key);


/*
** Function:      search_for_county
** Description:           Finds and returns the first county with the given
**                       name found in the counties in the passed state.
** Parameters:            struct state state_to_expand:
**                       The state with the counties to search through.
**               std::string search_key:
**                       The keyword to search for in the state_list.
** Return Value:    Returns the first struct county found in the given
**                       state's list that contains the passed keyword.
** Pre-Conditions:    county_list is not empty, the counties are all fully
**                       initialized with names, the state is initialized.
*/
struct county search_for_county(
  struct state state_to_expand,
  std::string search_key);


/*
** Function:      display_states
** Description:           Prints all states in a given state_list to cout.
** Parameters:            struct state * state_list:
**                       The list of states to be displayed.
**               int num_states:
**                       The number of states contain in state_list.
** Pre-Conditions:    The state_list is not empty, unless num_states
**                       is equal to 0, and all states contained are
**                       fully initialized (at least names).
*/
void display_states(
  struct state * state_list,
  int num_states,
  int indentation_level);


/*
** Function:      display_counties
** Description:           Prints all counties in a given county_list to cout.
** Parameters:            struct state state_to_expand:
**                       The state containing the counties to display.
** Pre-Conditions:    The county_list is not empty, unless num_counties
**                       is equal to 0, and all counties contained are
**                       fully initialized (at least names), state is
**                       also initialized.
*/
void display_counties(
  struct state state_to_expand,
  int indentation_level);
```

Testing Plan:

Test Cases				
Variable	Type	Value	Expected Result	Actual Result
command-line argument	edge	<empty>	Print invalid argument, exit.	TBD
	bad	ahhhh.txt	Print no file found, exit.	TBD
		1	Print invalid argument, exit.	TBD
	good	test_data.txt	Continue program.	TBD
		./test_data.txt	Continue program.	TBD
menu input	edge	<empty>	Ask for integer re-input.	TBD
		"ahhasd"	Ask for integer re-input.	TBD
	bad	-1 or 9	Ask for re-input in range.	TBD
	good	0 through 8	Continue program.	TBD
state / county search	edge	<empty>	Print no states found, cont.	TBD
		1	Print no states found, cont.	TBD
	bad	"new york"	Print no states found, cont.	TBD
		"ahhhhh"	Print no states found, cont.	TBD
	good	"Oregon"	Find Oregon, continue.	TBD
		"oregon"	Find Oregon, continue.	TBD
		"or"	Find Oregon, continue.	TBD
		"pen"	Find Pennsylvania, cont.	TBD