Project 3 CSc 402 Kutztown

Points: 25
Due: TBD. No late submissions accepted
Purpose: Encoding data in text form for fast access.
Assignment: In this assignment, you will write two short programs, one to create a text file of variable length records and another to access and print records and fields.

On the web and in the public area on **acad** for Project 3, is text file *States90.txt* . This file contains data on each state according to the 1990 census. Each entry consists of the state name on the first line and, in order, the population, population rank, population density, area, area rank, month, day, and year of admission to the US, order of admission to the US, and state capital, on the second line. For example, the entry for Ohio is in the box.

```
OHIO
10847115 7   264.9  41330 35  3  1 1803 17 COLUMBUS
```

.

Encoder

You will write a program (file name **encode.cpp**) to read *States90.txt* and encode each state into a series of 11 fields. All data is written to this file in textual form. Also, the file will contain no other characters; there are no newlines.

A class **State** has been provided in the appropriate areas of acad and on the course website. You are to use/update this class, or one of your own making, to input and store the data in a file named **states.enc** . The provided class requires support of a class **Date**, also provided. Note that the present overload of << is for outputting to the screen.

To create the encoded file, write a member function to write the state's data to an ostream in encoded form. While creating the encoded file, create a separate index file, where each index contains a state name preceded by its absolute starting position in the file. This file is to be named **states.idx** .

Accessor

Your second program will input the index into an appropriate data structure for fastest access and interact with the user as shown in the box. For example, when a state name is entered (P & L options), the index will be consulted, the state either located or reported as not found, and if found, the state's info in the index file accessed.

```
F)ull State Output
P)rint State Names
L)ookup State Data
T)otal Population
A)verages
```

Notes:

- The states aren't in alphabetical order in the states file.
- Each state must be accessible via one lookup in the index file, and any of its fields by one additional lookup.
- An ostrstream is recommended for converting numeric data. Clear it before each subsequent use.
- The various datum have varying lengths.
- All printouts must be well labeled and neatly formatted.
- Menu Notes:
  - The F option prompts for state name and outputs the entire data for that state, neatly and well formatted.
  - The L option prompts for state name, and if found, uses a submenu to solicit individual fields to be displayed. This menu persists until the user chooses to exit back to the main menu.
  - The A option prints the average population and area for each state.
- All input, including state names, should be handled case insensitively.
- Accessing the encoded data file is by opening it once and then moving the file pointer to the desired location.
- The application creates the index file. Obtaining and storing the information is up to you. Detail your data structure selections and design decisions in your readme. During or following input of state data, create the index file and write each state, in encoded form, to **states.enc**.
- You **must** error check opening the files in the accessor application. Failure must be reported
- You may **not** use an array of **State** in the accessor application. **State**s are read on demand into a single State object.
- This program will be graded using scripts. It is of critical importance that your program follow specs.
- You may use a programming language other than C++.
- Senior/graduate level documentation is expected in all files, including those provided.
- Create a Doxygen (or other appropriate) site whose mainpage describes all design decisions.

Turnin:

Turn in your project using the turnin script. Submit the **State** class (2 files), the **Date** class (2 files), **encode.cpp**, and **access.cpp**. You must submit a **makefile** whose default action will create both executables, named **encode** and **access**, which can also be built separately**.** Also, submit States90.txt.

Your readme, which details design decisions for indexing the states data, the data structures selected and justification for those selections, also provides info on running the program and the Doxygen link, which must take the user to the mainpage, which is to contain a synopsis. It is to be submitted to the **Project 3 Readme** dropbox on D2L. There will be a penalty for any deviation from these specs.