ARIZONA STATE UNIVERSITY

CSE 310 — **Data Structures and Algorithms** — Fall 2019

Instructor: Dr. Violet R. Syrotiuk

Project #2

For SLN 84794 (MW): milestone due M 10/21/2019; complete project due M 11/04/2019
For SLN 83657 (TTh): milestone due T 10/22/2019; complete project due T 11/05/2019

Storm data, provided by the National Weather Service (NWS), contain a chronological listing, by state, of hurricanes, tornadoes, thunderstorms, hail, floods, drought conditions, lightning, high winds, snow, temperature extremes and other weather phenomena. The data also contain statistics on personal injuries and damage estimates. Data is available from 1950 to the present for the United States of America.

This project goal of this project is to implement a *storm event application* that manages storm event data and uses it to answer queries meeting given selection criteria. The storm event data will be indexed by a hash table, and a binary search tree and max-heap as appropriate to support the queries.

Note: This project is to be completed **individually**. Your implementation **must** use C/C++ and ultimately your code **must** run on the Linux machine `general.asu.edu`.

Your application is required to implement the search tree, hash table, and max-heap and the needed operations on them yourself. Any dynamic memory allocation **must** be done yourself, using either `malloc()` and `free()`, or `new()` and `delete()`. You may not use any external libraries to implement any part of this project, aside from the standard libraries for I/O, file I/O, and string functions. If you are in doubt about what you may use, ask me.

You must use a version control system as you develop your solution to this project, *e.g.*, GitHub or similar. Your code repository must be *private* to prevent anyone from plagiarizing your work.

The rest of this project description is organized as follows. §1 describes the storm and fatality event data format, and the queries that your storm event application must support. §2 summarizes the requirements for the storm event application for this project. §3 describes measures to help evaluate the effectiveness of the data structures used in the application. Finally, §4 describes the submission requirements for the milestone and the full project deadlines.

1 Storm Event Application: Data and Query Format

Your storm event application, named `storm`, reads command line argument `n`, indicating the number of years of storm data to manage. Following this are `n` years on the command line, each a distinct four digit year `YYYY`, 1950 ≤ `YYYY` ≤ 2019. For each year `YYYY`, there are two input data files. The first is named `details-YYYY.csv`, and is a *comma separated value* (CSV) file containing the storm event data. The second is named `fatalities-YYYY.csv`, also a CSV file, containing the storm fatality data. The format of these data files is described next.

1.1 The Storm and Fatality Event Data Format

For each year, each line of the file `details-YYYY.csv` except the first contains details of a storm event described by the following 14 fields, in order. The first line of the file contains the field names, and should be skipped.

1. `event_id`: An identifier assigned by the NWS for a specific storm event; it links the storm event with the information in the `fatalities-YYYY.csv` file. Example: 383097.
2. `state`: The state name, spelled in all capital letters, where the event occurred. Example: GEORGIA.
3. `year`: The four digit year for the event in this record. Example: 2000.
4. `month_name`: Name of the month for the event in this record spelled out. Example: January.

Table 1: Valid Event Types

Event Name	**Designator**	**Event Name**	**Designator**
Astronomical Low Tide	Z	Hurricane (Typhoon)	Z
Avalanche	Z	Ice Storm	Z
Blizzard	Z	Lake-Effect Snow	Z
Coastal Flood	Z	Lakeshore Flood	Z
Cold/Wind Chill	Z	Lightning	C
Debris Flow	C	Marine Hail	M
Dense Fog	Z	Marine High Wind	M
Dense Smoke	Z	Marine Strong Wind	M
Drought	Z	Marine Thunderstorm Wind	M
Dust Devil	C	Rip Current	Z
Dust Storm	Z	Seiche	Z
Excessive Heat	Z	Sleet	Z
Extreme Cold/Wind Chill	Z	Storm Surge/Tide	Z
Flash Flood	C	Strong Wind	Z
Flood	C	Thunderstorm Wind	C
Frost/Freeze	Z	Tornado	C
Funnel Cloud	C	Tropical Depression	Z
Freezing Fog	Z	Tropical Storm	Z
Hail	C	Tsunami	Z
Heat	Z	Volcanic Ash	Z
Heavy Rain	C	Waterspout	M
Heavy Snow	Z	Wildfire	Z
High Surf	Z	Winter Storm	Z
High Wind	Z	Winter Weather	Z

5. `event_type`: The event types permitted are listed in Table 1, spelled out.
6. `cz_type:` Indicates whether the event happened in a county/parish (C), zone (Z), or marine (M).
7. `cz_name`: County/Parish, Zone or Marine name assigned to the county or zone. Example: AIKEN.
8. `injuries_direct`: The number of injuries directly related to the weather event. Examples: 0, 56.
9. `injuries_indirect`: The number of injuries indirectly related to the weather event. Examples: 0, 87.
10. `deaths_direct`: The number of deaths directly related to the weather event. Examples: 0, 23.
11. `deaths_indirect`: The number of deaths indirectly related to the weather event. Examples: 0, 4, 6.
12. `damage_property`: The estimated amount of damage to property incurred by the weather event, *e.g.*, 10.00K = \$10,000; 10.00M = \$10,000,000. Examples: 10.00K, 0.00K, 10.00M.
13. `damage_crops`: The estimated amount of damage to crops incurred by the weather event *e.g.*, 10.00K = \$10,000; 10.00M = \$10,000,000. Examples: 0.00K, 500.00K, 15.00M.
14. `tor_f_scale`: Enhanced Fujita Scale describes the strength of the tornado based on the amount and type of damage caused by the tornado. The F-scale of damage varies in the destruction area; therefore, the highest value of the F-scale is recorded for each event. Examples: EF0, EF1, EF2, EF3, EF4, EF5.

The storm fatality data is also provided in a CSV file named `fatalities-YYYY.csv`. Each line of the file, except the first, contains information on fatalities that occurred within a storm event, described by the following 7 fields, in order. The first line of the file contains the field names, and should be skipped.

1. `fatality_id`: An identifier assigned by NWS to denote the individual fatality that occurred within a storm event. Example: 17582.
2. `event_id`: An identifier assigned by NWS to denote a specific storm event; it links the fatality with the storm event in the `details-YYYY.csv` file. Example: 383097.
3. `fatality_type`: D represents a direct fatality, whereas I represents an indirect fatality. Example: D.

Table 2: Direct Fatality Location Table

Abbreviation	**Location**
BF	Ball Field
BO	Boating
BU	Business
CA	Camping
CH	Church
EQ	Heavy Equip/Construction
GF	Golfing
IW	In Water
LS	Long Span Roof
MH	Mobile/Trailer Home
OT	Other/Unknown
OU	Outside/Open Areas
PH	Permanent Home
PS	Permanent Structure
SC	School
TE	Telephone
UT	Under Tree
VE	Vehicle and/or Towed Trailer

4. `fatality_date`: Date and time of fatality in MM/DD/YYYY HH:MM:SS format, 24 hour time. Example: 04/03/2012 12:00:00.
5. `fatality_age`: The age of the person suffering the fatality. Example: 38.
6. `fatality_sex`: The gender of the person suffering the fatality. Example: F.
7. `fatality_location`: The fatality locations permitted are listed in Table 2, spelled out. Example: Under Tree.

1.2 Storm Data Query Format

In the following, `<>` bracket parameters to the query, while the other strings are literals. There are 4 queries that your storm application must be able to support:

1. `find event <event_id>`, searches the hash table for the storm event with identifier `event_id`. If found, it follows the index for the storm event for the given year to print the information associated with the event (*i.e.*, the fields of the `struct storm_event` in order, with each field labelled on a separate line); otherwise it prints `"Storm event <event_id> not found."` If there are no fatalities associated with the event, print `"No fatalities."` Otherwise, it should print the fields of each fatality event associated with this `event_id`, with each field on a separate line.

   Example: `find event 383097` // find storm event with given id

2. `find max <number> <YYYY> <damage_type>`, where `number` is an integer ≤ 50, `YYYY` is either a specific four digit year or the literal `all`, and `damage_type` is either `damage_property` or `damage_crops`. This query builds a max-heap on the field `damage_type` for the given year `YYYY` or `all` years of storm data. It executes `number` Delete-Max operations on the heap, each time printing information for the most expensive storm in terms of the damage it caused to property or crops. Specifically, for each event print the `event_id`, `event_type`, and amount of damage of as a dollar amount, on a single line.

   Example: `find max 4 1950 damage_property` // find 4 most expensive storms to property in 1950
   `find max 10 all damage_crops` // find 10 most expensive storms to crops in all years

3. `find max fatality <number> <YYYY>`, where `number` is an integer ≤ 50, `YYYY` is either a specific four digit year or the literal `all`. This query builds a max-heap on the number of fatalities for the given year `YYYY` or `all` years of storm data. It executes `number` DELETE-MAX operations on the heap, each time printing information for the most fatal storm. Specifically, for each fatality print all fields of the `fatality_event`.

   Example: `find max fatality 4 1950` // find 4 most fatal storms in 1950
   `find max fatality 10 all` // find 10 most fatal storms in all years

4. `range <YYYY> <field_name> <low> <high>`, where `YYYY` is either a specific four digit year or the literal `all`, where `field_name` is either `state` or `month_name`, and `low` and `high` are strings. To answer this query, construct a binary search tree (BST) for the given year(s) on primary key `field_name` and secondary key `event_id`, *i.e.*, the tree is ordered first by `field_name`, and for equal `field_name` then ordered by `event_id`. Then, perform an in-order traversal of the search tree printing event information for events whose `field_name` is greater than or equal to `low` and less than or equal to `high`. If no applications are found whose `field_name` is in the given range print `"No storm events found for given range."` Otherwise, your application should print for each event within the range of the search parameters, first ordered by `field_name`, and then ordered by `event_id`, the following specific fields: For `field_name` of `state`, print the `state`, `event_id`, `year`, `event_type`, `cz_type`, and `cz_name`. For `field_name` of `month_name`, print the `month_name`, `event_id`, `year`, `event_type`, `cz_type`, and `cz_name`.

   Example: `range 1950 state A C` // all storms in states alphabetically from A to C in 1950
   `range all month_name January January` // all storms in January in all years

2 Program Requirements for Project #2

1. Write a C/C++ storm event application, named `storm`, that reads command line argument `n`, indicating the number of years of storm data to manage. Following this are `n` years on the command line, each a distinct four digit year `YYYY`, $1950 \leq$ `YYYY` ≤ 2019. That is, the application is invoked as: `storm n` $\texttt{YYYY}_1 \ldots \texttt{YYYY}_n$. For example,

   ```
   storm 1 1950
   storm 3 1950 1951 1952
   ```

   For each year `YYYY`, there are two input data files. The first is named `details-YYYY.csv`, and is a *comma separated value* (CSV) file containing the storm event data. The second is also a CSV fle, named `fatalities-YYYY.csv`; it contains the storm fatality data.

2. Provided for you is a file named `defns.h` in which structures have been defined for storing the storm and fatality event data with fields matching this format. For each year i, $1 \leq i \leq$ `n` of storm data, initialize an array of `struct annual_storms`. This involves:

   (a) First initializing an array of `struct storm_event` of size s_i where s_i is the number of events in the file $\texttt{details-YYYY}_i\texttt{.csv}$.

   (b) As you initialize each entry in the `storm_event` array, you must also insert a subset of the fields of the event into a hash table. The hash table is an array of pointers to `struct hash_table_entry` all initialized to NIL. Each hash table entry has three fields: The `event_id` and `year` of the storm event, and the index position `event_index` at which the event is stored in the `storm_event` array.

   The hash function is computed as the `event_id` modulo the hash table size. The hash table size is the first prime number larger than $2 \times (s_1 + \cdots + s_n)$, where s_i is the number of storm events in $\texttt{details-YYYY}_i\texttt{.csv}$, $1 \leq i \leq n$. (You may find the function in `prime.cc` useful for this purpose.)

(c) Now, process the `fatalities-YYYY`$_i$`.csv` file. Hash on the `event_id` of each fatality event to find the `event_index` in the `storm_event` array. Insert the fatality into a linked list of fatality events for the corresponding storm event.

3. Once the array(s) and hash table have been initialized, your storm event application is ready to start processing queries. It reads `q`, the number of queries, from `stdin`, followed by `q` queries, one per line. The format of queries follows the format described in §1.2. The output must be written to `stdout`.

   Example: 3 // q=3 queries
   `range 1951 month_name February March` // all storms $\geq$ February and $\leq$ March in 1951
   `find event 383097` // find storm event with given event id
   `find max fatality 10 1950` // find 10 most fatal storms in 1950

4. After processing a query, collect and output the characteristics of the data structures you've built as described in §3.

5. After a query is processed, any max-heap or binary search tree data structures created dynamically must be deallocated on answering the query.

Sample input files that adhere to the format described in §1.1 ansd §1.2 will be provided on Canvas; use them to test the correctness of your program.

3 Characteristics of the Data Structures

For a BST constructed to answer a query of the form `range <YYYY> <field_name> <low> <high>`:

1. Print a count of the total number of nodes in the BST.
2. Print the height of the root of the BST, and the height of its left and its right subtree.

For a max-heap constructed to answer a query of the form `find max <number> <YYYY> <damage_type>`, or of the form `find max fatality <number> <YYYY>`:

1. Print a count of the total number of nodes in the max-heap.
2. Print the total height of the max-heap, and the height of its left and right subtrees.

After processing all queries, print a summary of the hash table that includes:

1. Print a table that lists for each chain length ℓ, $0 \leq \ell \leq \ell_{max}$, the number of chains of length ℓ, up to the maximum chain length ℓ_{max} that your hash table contains.
2. Compute and print the load factor for the hash table.

4 Submission Instructions

Submissions are always due before 11:59pm on the deadline date.

1. For SLN 84794 (MW) the milestone is due on Monday, 10/21/2019. For SLN 83657 (TTh) the milestone is due on Tuesday, 10/22/2019. See §4.1 for requirements.
2. For SLN 84794 (MW) the complete project is due on Monday, 11/04/2019. For SLN 83657 (TTh) the complete project is due on Tuesday, 11/05/2019. See §4.2 for requirements.

It is your responsibility to submit your project well before the time deadline!!! Late projects are not accepted. Do not expect the clock on your machine to be synchronized with the one on Canvas!

An unlimited number of submissions are allowed. The last submission will be graded.

4.1 Requirements for Milestone Deadline

For the milestone deadline you must complete steps 1 and 2 of §2, the Program Requirements for Project #2. You also must be able to answer `find event` queries. As well, you must print characteristics of the hash table as described in §3.

Using the submission link on Canvas for the Project #2 milestone, submit a zip[1] file named:

`yourFirstName-yourLastName.zip`

that unzips into the following:

Project State (10%): In a folder (directory) named `State` provide a brief report (.pdf preferred) that addresses the following:

1. Describe any problems encountered in your implementation for this project milestone.
2. Describe any known bugs and/or incomplete implementation in the project milestone.
3. While this project is to be completed individually, describe any significant interactions with anyone (peers or otherwise) that may have occurred.
4. Cite any external books, and/or websites used or referenced.

Implementation (40%): In a folder (directory) named `Code` provide:

1. In one or more files, your well documented C/C++ source code implementing the requirements for this project milestone.
2. A `makefile` that compiles your program and produces an executable named `storm` on `general.asu.edu`. Our TA will write a script to compile and run all student submissions on `general.asu.edu`; therefore executing the command `make storm` in the `Code` directory must produce the executable `storm` also located in the `Code` directory.

Correctness (50%): The correctness of your program will be evaluated by running `find event` queries on storm event and fatality files, some of which will be provided to you on Canvas prior to the deadline for testing purposes.

The milestone is worth 30% of the total project grade.

4.2 Requirements for Complete Project Deadline

For the full project deadline, you must implement all the requirements for Project #2 as described in §2. Using the submission link on Canvas for the complete Project #2, submit a zip[2] file named

`yourFirstName-yourLastName.zip`

that unzips into the following:

Project State (10%): Follow the same instructions for Project State as in §4.1 except applied to all project requirements.

Implementation (40%): In a folder (directory) named `Code` provide:

1. In one or more files, your well documented C/C++ source code implementing *all* requirements for this project.
2. A `makefile` that compiles your program and produces an executable named `storm` on `general.asu.edu`. Our TA will write a script to compile and run all student submissions on `general.asu.edu`; therefore executing the command `make storm` in the `Code` directory must produce the executable `storm` also located in the `Code` directory.

Correctness (50%): The correctness of your program will be evaluated by testing all queries in §1.2 on storm event and fatality files, some of which will be provided to you on Canvas prior to the deadline for testing purposes.

[1]**Do not** use any other archiving program except `zip`.
[2]**Do not** use any other archiving program except `zip`.