CS2337 – PROJECT 3 – Cidercade Database

Pseudocode Due: before 10/28 at 11:59 PM

Core Implementation Due: before 11/3 at 11:59 PM

Final Submission Due: before 11/9 at 11:59 PM

KEY ITEMS: Key items are marked in red. Failure to include or complete key items will incur additional deductions as noted beside the item.

Submission and Grading:

- The pseudocode will be submitted in eLearning as a Word or PDF document and is not accepted late.
- All project source code will be submitted in Zybooks.
    - Projects submitted after the due date are subject to the late penalties described in the syllabus.
- Programs must compile using gcc 7.3.0 or higher with the following flags enabled
    - -Wall
    - -Wextra
    - -Wuninitialized
    - -pedantic-errors
    - -Wconversion
- **Type your name and netID in the comments at the top of all files submitted. (-5 points)**

Objective: Use object-oriented programming to implement and utilize a binary search tree

Problem Barcades have become very popular venues over the last few years. A local barcade, Bishop Cidercade, is working to open near downtown Dallas. Before they open, they need help establishing a system to track information on all of their games. They have hired you to build this system with a simple interface allowing them to manipulate and search the data.

Pseudocode: Your pseudocode should describe the following functions

- For each function, identify the following
    - Determine the parameters
    - Determine the return type
    - Detail the step-by-step logic that the function will perform
- Functions
    - Binary Tree
        - Adding a record
        - Deleting a record
        - Searching for a record
    - Main class
        - Main function
        - Editing a record
        - Printing out the database
        - Any additional functions that you plan to create

Core Implementation:

- Read database file and build tree
- Insert new nodes into tree
    - Can be iterative
- Search tree for complete and partial matches
    - Can be iterative
- Display output for each command processed
- Display tree in ascending alphabetical order
    - Must be recursive (because it is easier that way)
- Core implementation tests will utilize sample input file for testing and grading
- Templates are not required for core implementation
- Edit is not required for core implementation
- Delete is not required for core implementation
- Displaying in descending alphabetical order is not required for core implementation
- Displaying the final database is not required for core implementation

Zybooks Information:

- You will have multiple source files
    - `main.cpp`
    - `BinTree.h`
    - `Node.h`
    - `Game.h and Game.cpp`
- Core implementation has unlimited submissions
    - This will help you make sure the basic actions of your program are working properly in the environment
- Final submission is limited to 10 submissions
    - This will be manually checked by the grader
- White space will not be checked

Class Details:

- All classes must be of your own design and implementation.
    - **Do not use the pre-defined C++ classes (-20 points if pre-defined classes used)**
    - Exception: you can use the queue class from the STL for breadth-first traversal
- Binary Tree Class - `BinTree.h`
    - No reference to anything inside Game class
    - Attributes
        - Root – templated Node pointer
    - Overloaded constructor
        - Takes node argument and assigns to root pointer
    - Insert
    - Search
    - Delete

    - Any additional functions added must be specific to any binary search tree
        - Any functions related to a specific problem should be in main
    - All traversals of the tree will be done recursively (-10 points if not)
        - This includes functions that add, delete, search and display the tree
        - This does not include the breadth-first traversal to display the database at the end of the program
- Node Class - `Node.h`
    - Templated class (-10 points if not)
    - No reference to anything inside Game class
    - Attributes:
        - Left pointer
        - Right pointer
        - Templated payload variable to hold object
            - Will be used to hold game objects in the program
    - Overloaded constructor
        - Pass in object to hold in node
    - Overloaded insertion operator (<<)
    - Overloaded less than operator (<)
    - Overloaded greater than operator (>)
- Game Class - `Game.h and Game.cpp`
    - Attributes
        - Name
        - High score
        - Initials
        - Plays
    - Overloaded constructor
        - Pass in values for all attributes
    - Overloaded insertion operator (<<)
    - Overloaded less than operator (<)
    - Overloaded greater than operator (>)

User Interface:

- There will be 2 input files
    - Prompt the user for the name of the file containing the database first
    - Prompt the user for the name of the file containing batch commands
- Display all output to the console

Details:

- All records from the database files are stored in memory with a binary tree (-10 points if not)
- Batch commands
    1. Add a record to the database
    2. Search for a record and display it
    3. Edit a record

4. Delete a record
5. Sort records

- **Add a record:** Create a new node and add it to the tree
- **Search for a record:** The search term will be a word or phrase. Search through the tree and display the complete record for any game name that contains the search term.
- **Edit a record:** With a given game name, the program will update the record and confirm the change by displaying the new record on the screen. The following items can be edited:
    1. High score
    2. Initials
    3. Number of plays
        - If number of plays is edited, the revenue should be recalculated
            - $0.25 per play
- **Delete a record:** User will enter a game name. Delete the record from the tree
- **Sort records:** Display records in ascending or descending order by name.
- You can expect all input to be valid.

Database Format:

- Each record will be on a separate line in the file
- Format
    - `<name>, <high_score>, <initials>, <plays>, $<revenue><newline>`
    - Each field is separated with a comma and a space
- `<name>` - may be multiple words
- `<high_score>` - 1-9 digits
- `<initials>` - 3 characters – no white space
- `<plays>` - 1-4 digits
- `<revenue>` - <1-4 digits><decimal><2 digits>

Batch Command File:

- Each command will be on a separate line in the file
- Each line will have a newline character at the end of the line
    - The last line may or may not have a newline character
- Command format
    - There is a single space between fields.
    - Add a record
        - `1 ”name” high_score initials plays $revenue`
    - Search for a record
        - `2 <search term>`
        - Search term may contain spaces
    - Edit a record
        - `3 “name” <field number> <new value>`
        - `<field number>`
            - 1 = high score
            - 2 = initials

            - 3 = number of plays
        - The double quotes surround the name so that you know where the end of the name is
    - Delete a record
        - `4 <name>`
        - Name may contain spaces
        - Double quotes are not necessary here since there is no data after the name
    - Sort records
        - `5 <asc/des>`
        - A single word will follow the value: `asc` or `des`

Output:

- Each command in the input file will generate output to console.
- After each command output, write 2 blank lines to the file.
- The output for each command is as follows:
- Add a record

    -
      ```
      RECORD ADDED
      Name: <name>
      High Score: <high_score>
      Initials: <initials>
      Plays: <plays>
      Revenue: $<value> - formatted to 2 decimal places
      ```
- Search for a record
    - `<name> FOUND or <name> NOT FOUND`
    - If found

        -
          ```
          High Score: <high_score>
          Initials: <initials>
          Plays: <plays>
          Revenue: $<value> - formatted to 2 decimal places
          ```
- Edit a record
    - `<name> UPDATED`
    - `UPDATE TO <field> - VALUE <value>`
        - Fields:
            - `high score`
            - `initials`
            - `plays`

    -
      ```
      High Score: <high_score>
      Initials: <initials>
      Plays: <plays>
      Revenue: $<value> - formatted to 2 decimal places
      ```
- Delete a record

    -
      ```
      RECORD DELETED
      Name: <name>
      High Score: <high_score>
      Initials: <initials>
      ```

```
        Plays: <plays>
        Revenue: $<value> - formatted to 2 decimal places
```

- Sort records
  - `RECORDS SORTED <ASCENDING/DESCENDING>`
  - Display all records (one per line) in the proper order
  - `<name>, <high_score>, <initials>, <plays>, $<revenue><newline>`
- At the end of the program, write the database to a file
  - The database will be written to `cidercade.dat`
  - Write the tree to the file using a breadth-first traversal
  - Record format
    - `<name>, <high_score>, <initials>, <plays>, $<revenue><newline>`