Prathyoosha Chaya

CS 202 PROGRAMMING SYSTEMS

Prof. Karla Fant

**DESIGN WRITE-UP 3:**

This program will attempt to have the following self-regulating classes in order to act in an object-oriented fashion:

The Used apps class is responsible for managing the linked list of the collections of apps used – sorted according to the frequency of use. It will include a collection of node classes, which contain the BST (the balanced tree used is an AVL tree), a pointer to the next node, and the function definitions of the public member functions dealing with list management (insert, remove). The following functions will be implemented for the Used apps class, with the list removal/insertion functions not included, as they will not be available to the user and will only be invoked in the case of loading the file data initially into the data structures.

Display all function (list): This public function will display all of the information in the Used app list by going to the head pointer and calling the display all function for the tree in the node. It then recursively calls display (list) again for the next node in the list. It takes a pointer to a node as a parameter.

Write function for list (to accompany load function): The purpose of this function is to write the contents of the list back into the file. It takes a node pointer as a parameter. It calls the write function for the tree at the current node, then recursively calls the function with the next node until all list members are written. It returns a failure or success case.

Display function (list): This public function will display either the frequently used (1st node), moderately used (2nd node), or the rarely used (3rd node) apps in the list by going to the head and displaying the tree under the respective node. It returns the success/failure case (if the table is empty) by Boolean value, and it takes a pointer to a node as a parameter.

The App Collection class is responsible for managing the collection of apps, which is contained in each node of the linked list. This class “has” an app class, and its primary responsibility is to manage the balanced tree management. The data structure used in this class and its derived classes is an AVL tree.

Insert app function: The insert app function is public and takes the app pointer as a parameter, along with a pointer to a node in the tree. It inserts the item based on the App usage hours of the app to be inserted. It returns an integer value depending on the case of the insertion success/failure by returning the count of nodes traversed until insertion. It will recursively call the insert function with the left and right pointers.

Remove term function: The remove app function is public and takes a name of an app as a parameter. It first displays all of the apps. The user enters the app that they would like to remove (which runs through the convert to uppercase function), which is then used as a comparison term when traversing the tree. If the item is found at a leaf, then it is deleted and set to null. If it is an internal node with 1 left or right child, then a temp pointer is created to point to its child and the root is deleted and set to the temp pointer. If it is an internal node with 2 children where the right node has no left children, then the right child’s data is copied to the root’s data. If the right node has a left child, then the tree is traversed until the left pointer is null, dragging a prev. pointer, then the parent adopts the temp’s right child and the in-order successor’s data becomes the root’s data. The in-order successor is then deleted. If a match to the search term is not found or the tree is empty, nothing is removed. It returns a success or failure flag of if the item was removed/not found/not removed.

Retrieve function: The purpose of the public retrieve function is to take the search term to be found from the client program and return the data of that term (if it is found) as a parameter. It uses the same algorithm used in the insert function, and returns the found item’s data through the function parameters. If a match to the search term is not found or the tree is empty, no data is returned as a parameter. It returns a success or failure flag of if the item was removed/not found/not removed.

Display all function (tree): This public function will display all of the information in the tree by going to the root node in the tree and displaying, then returning the function calls for the left and right subtrees. It returns the success/failure case (if the table is empty) by Boolean value, and it takes a pointer to a node as a parameter.

Write function for tree (to accompany load function): The purpose of this function is to write the contents of the tree back into the file. It takes a node pointer as a parameter. The app data will be outputted to the file (no append), separated with “:” delimiters. At the end of the app information, a “\n” delimiter is placed. This is recursively called until all tree members are written. It returns a failure or success case.

Load function for tree: The purpose of the load function is to take the App information in the text file and store it into the tree. It first creates temporary arrays to store the name, and opens the text file to attempt to write the term to set EOF. While the file has not reached the end, it compares each value to the current subtree’s App usage hours (done recursively) – if it is greater than or equal to the root, it enters the right, and if less than root, it enters left. It uses the existing delimiters mentioned in the “write” function to read into each entry’s fields.

The App class is the abstract base class to the derived types of apps: Website, Utility, and Game, along with function definitions of the public member functions dealing with tree management. It contains edit data, retrieve app, display, write, and load functions (pure virtual) to manage the app data and perform functionality required by the classes they are contained in. This app data is managed by the derived classes, and includes general information such as “name”, “amount of hours used”, “date used”, that will be managed by the base class. Specialized information (such as URL, game desc, utility features) will fall upon the derived classes to manage.

Constructors: The purpose of the constructors are to initialize the data members. All classes will have default constructors, along with copy constructors in the case of dynamic memory. Initialization lists will be used where possible.

Destructors: The destructors will deallocate all previously allocated dynamic memory in their respective classes.

Convert to uppercase function: The purpose of this private function is to convert the inputted array of characters to uppercase, using the toupper functionality in the cctype library. It takes a source and destination array of characters as input parameters, and changes the contents of destination to be the uppercase version of the source array of characters, going character by character. This will be a puclic method in the base class (location class).

IMPLEMENTATION FILES:

The main implementation file will contain the function definitions below. Others will contain the function definitions for the class methods specified above.

In the main function, the menu interface will be implemented for the app user to interact with the app. An object of type game, and an object of type map are created for the purpose of calling game and map class functions and passing the user’s information to be added into these functions. Then, a “menu option” variable (type int, to keep track of the menu item entered). The setup function for the game class is called, and players are entered, followed by the build map function.

Then, the menu is outputted by calling the menu function. The user can choose which option they would like to execute from the following menu format:

1: Use an app (or record a used app)

2: Check frequency of an app, and app information

3: Display all apps used

4: Retrieve app

5: Remove used app

6: Save and quit app

7: Quit app

The user will be prompted to enter a number, which is stored into the “menu option” variable once the function exits. If a 1 is entered, then the player can change their current checkpoint to any surrounding checkpoint. Once the function is exited, the menu function is entered again. The functions continue to be called until the user decides to quit (or save and quit). The program can be quit without writing to the file, or quit after writing the contents of the Used App list to the file.

Menu function: This function takes no parameter and returns an int value. It outputs the menu for the user to select, and returns the user’s selection. The menu displayed is shown in the main function.

