Developer documentation of current state

Facebook – like Application

Description: The Facebook – like Application is a C-based program which handles the data of users in a Facebook-like style. This program allows saving and organizing the data of the users, and the members who invited them. Main functionalities of the application are searching, saving and deleting the members. The program uses structures, functions and linked lists to store and organize the database, and BST to search through the members.

Table of Contents

Structures

- Date: Structure that stores user's birthdate.
  - int day: Day of the year
  - int month: Month of the year
  - int year: The year

- nameData: Structure that stores name info of the user.
  - char *name: Name of the user.
  - char *surname:Surname of the user.
  - char *username:The username(nickname) of user.
  - char *invited_by:Name of the person who invited the user.

- Member: Structure that saves the info of the members.
  - person name_info: The name info of the members.
  - date date_info:Birthdate info.
  - char *bplace: Pointer that points to the birthplace of the user.
  - int age:The age of the user.

- linkedLIST: Linked list which saves the members data, and a pointer to the next node.
  - member data:.
  - struct list *next:.

- BST:Binary Search Tree, saves data, a pointer to the right and left node.
  - member data:.
  - struct bst *right:
  - struct bst *left:

Functions

Date functions

- **date DATE(int d, int m, int y)**: Function that creates date by users input. Takes integer numbers as day, month and year. It generates and returns x as a date structure variable.

NameData Functions

- **nameData allocateName (char *n, char *sn, char *un, char *i_by)**:Function that creates person by users input. Takes pointers to the name, surname, username, and name of the member who invited them, generates the name and returns x as a person structure variable.

- **void freeName(nameData name):** Function which is meant to free the memory we allocated for all the memory we allocates in allocateName function.

Member Functions

- **member allocateMember (char *name, char *surname, char *username, char *invitedby, int d, int mo, int y, char *bp)**: Function that creates the members by user's input. Works in the same style as allocateName, where it returns x as a member structure variable which encapsulates all the data of the person.

- **void freeUserData(member m):** Frees the memory which was allocated by the function allocateMember.

List Functions

- **linkedLIST *insertAtFront(linkedLIST *head, member userData)**: Function that inserts a node in the front of the list.
- **linkedLIST *insertAtEnd(linkedLIST *head, member userData):** Function that inserts a node at the end of the list.
- **void freeLinkedList(linkedLIST *head):** Function that takes the head of a linked list as input, and frees the memory allocated for each node.
- **void printList(linkedLIST *head):** Function that takes the head of linked list and prints the data of each member.
- **linkedLIST *BSTtoLinkedList(BST *root, linkedLIST *head):** Function that takes the head of linked list, and root of the binary tree, and recursively changes the bst to the linked list. Returns the head of the newly allocated linked list
- **linkedLIST* scanMember(linkedLIST *head):** Function that scans the members, by user input. Adds the members to the existing linked list, and returns the new linked list with the new members.
- **linkedLIST *createHead(member member_data):** Function that allocates memory for a new node in linked list. Takes the user data as input and returns the new allocated node.

BST Functions

- **BST *insertNodesinBST(BST *root, member memberData):** Inserts a member in the BST. First checks if the tree exists, if not it allocates memory for the root.

If the tree exists it compares the name, to see if it belongs before or after the current node, and places it in the right side.

- **BST *searchNameInBST(BST *root, char *what)**Searches for a member by name in the BST.

Checks if the name we are looking for is the root of the BST first and or if the tree exists and returns the pointer to the root. Otherwise, the function works recursively by comparing the name to the data on the nodes to find the right position in the BST

- **BST *createRoot(member member_data):** Function that allocates memory for a new node in the bst. Takes user data and returns the newly allocated node for bst.

- **void destroyBST(BST *root):** Function that destroys the bst. It loops through all the nodes and recursively frees the memory at each node.

- **void printUser(BST *root):** Function that prints the data of the member if found by BST.

- **BST* linkedListToBST(linkedLIST* head):** Function that makes a BST from the linked list. Takes the linked list head as input and returns the root to the newly created bst. It loops though the linked list and allocates memory for each node, of the existing node in the linked list. Uses the function insertNodesinBST, to insert nodes recursively at the designated place.
-
- **BST *deleteNodeInBST(BST *root, char *what):** Function that deletes a node in the BST. Takes the root to the bst, and the name of the member that we want to delete. It has many cases. If the root has no children, if the root has only 1 child, and if root has two children. Compares the name at the root, to the name of the member and recusively searches and deletes once the member is found.

FILE Handling Functions

- **void encodeData(FILE *file, member data):** Function that encodes, or serializes data, so that it can be in a CSV format. Comma Seperated Values. Prints the data to the file with comma separating each item. Takes the file pointer as an input and, the member data.
-
- **void loadDatainFile(char *filename, linkedLIST *head):** Function which loads or saves the data in the file. Takes the filename, and the pointer to the linked list head as input, and while looping through the file, it saves each member using the encodeData function.
-
- **member decodeUser(char *bufferLine):** Function that parses the data from the file, input is a bufferLine string which contains the data from file separated by commas, and creates a member structure based on that.
-
- **linkedLIST *readLinkedList(char *filename):** Function that reads from the file and saves the data in a linked list. Takes the filename as input, and returns a pointer to the head of new linked list. It opens the file in read mode and checks if its null or not. If the file cannot open, it will print a message displaying that the file couldn't be opened. On the other hand, it will go through the file and parse each data, create the member structure from the data, and insert it into the newly allocated linked list.