Assignment 3

Group BJ

Ghofran Abourayana & Simon Lasak

https://csgitlab.ucd.ie/ghofran/asssignment-3-bj.git

How do you load a board?

```
// loads a list from a given file name. (used for option 1 and 2 from menu)
void load(ListNodePtr *sPtr, char fileName[])
{
    FILE *fptr;
    char name[20]; // will be read from file (may be item or list)

    if ((fptr = fopen(fileName, "r")) == NULL)
    {
        printf("error opening file\n");
        fflush(stdout);
        return;
    }

    while (fgets(name, 20, fptr) != NULL) // loops until end of file
    {
        // removes new line character
        int size = strlen(name);
        if (name[size - 1] == '\n')
        {
            name[size - 1] = '\0';
        }

        // checks if name is an item or list, if it ends in : then its a list
        size = strlen(name);        //(get size again after removing new line character
        if (name[size - 1] == ':') // if : present then its a list
        {
            name[size - 1] = '\0';
            insertList(sPtr, name); // since its a list, use insert list function.
        }
        else
        {
            ListNodePtr curr = *sPtr;
            while (curr->nextPtr != NULL) // find the last entered list in the linked list to
add its items.
            {
                curr = curr->nextPtr;
            }
            insertItem(&((curr)->items), name); // insertItem called with the last entered
list.
        }
    }
    fclose(fptr);
}
```

To load a board, we created a function which takes in a pointer to the start of the linked list, and the name of a file to be opened.

The file is read line by line using fgets and each line is stored in the array ‘name’. Then the last character is checked to see if it is ‘:’

If ‘:’ is present at the end, then it is identified as a list, the function insertList is called, it is given a pointer to the start of the linked list as an argument and the name of the list to be added.

insertList function: this function traverses through the linked list until it reaches the end, it then adds the given name as a new list.

```
void insertList(ListNodePtr *sPtr, char name[])
{
    ListNodePtr newPtr = malloc(sizeof(ListNode)); // allocates memory for the new list to be
added
    if (newPtr != NULL)                            // if space is available
    {
        strcpy(newPtr->name, name); // copies name we want to insert
        newPtr->items = NULL;       // initially sets items to null, another function is used
to insert items
        newPtr->nextPtr = NULL;

        ListNodePtr prev = NULL;
        ListNodePtr curr = *sPtr;
        while (curr != NULL) // iterates until it reaches end of linked list
        {
            prev = curr;
            curr = curr->nextPtr;
        }

        if (prev == NULL) // occurs when linked list is NULL
        {
            *sPtr = newPtr;
        }
        else
        { // adds newPtr to end of linked list.
            prev->nextPtr = newPtr;
        }
    }
    else
    {
        printf("error, no space available\n");
        fflush(stdout);
    }
}
```

If the last character is not ':' , then 'name' is an item, the function insertItem is given the current list as an argument, as well as the name of the new item to be added.

insertItem function: This function traverses through the items of a given list until it reaches the end, it then adds the name of the item under the list.

```
void insertItem(ItemNodePtr *sPtr, char name[])
{
    ItemNodePtr newPtr = malloc(sizeof(ItemNode));
    if (newPtr != NULL) // space available
    {
        strcpy(newPtr->name, name); // copies name we want to insert
        newPtr->nextPtr = NULL;
        ItemNodePtr curr = *sPtr;
        ItemNodePtr prev = NULL;
        while (curr != NULL)
        {
            prev = curr;
            curr = curr->nextPtr;
        }
        if (prev == NULL)
        {
            *sPtr = newPtr;
        }
        else
        {
            prev->nextPtr = newPtr;
            newPtr->nextPtr = NULL;
        }
    }
    else
    {
        printf("Error: not inserted\n");
        fflush(stdout);
    }
}
```

Once end of file is reached, the file is closed.

How did you edit lists and items?

Edit items

First thing that happens is the program checks whether the pointer to the first list is NULL or not. When the pointer to the first List is NULL that means there is no List on the current board. No list means nothing to edit -> exit the function and advise the user.

The user can choose to edit a list, where they can edit the items of the list, or to edit a board which handles the lists. When the user chooses to edit a certain list the function `ListNodePtr findList(ListNodePtr startingList);` is called, which asks the user for the name of the list they want to edit. This function is also called when the user wants to edit the board, but not straight away.

The function `findList` accounts for the user input to have a semicolon or not. Also, the comparison is made with `strcasecmp(currentList->name, editName)` which means that is case insensitive instead of the regular strcmp() function. All this input handling results in „Mark:“ being the same as „mark“ or „MaRk“ and so on. This makes it easier to find the desired list.

Then there is a submenu which gives the user the option to choose between (1) Editing the name of the item, (2) Adding a new item, (3) Deleting an item or (4) Going back to the main menu.

The `void editItem(ListNodePtr listPointer);` prompts the user to enter the name of the item they want to edit. It iterates through the linked list of items, searching for an item with a name that matches the one entered by the user. If it finds a matching item, it prompts the user to enter a new name for that item and updates the name accordingly. If it doesn't find a matching item, it informs the user that the item was not found.

The `void addItem(ListNodePtr listPointer);` prompts the user to enter the name of the new item. It reads the user input and removes the trailing whitespace (newline character) from the input. It calls the insertItem function to insert the new item into the linked list, passing a pointer to the pointer to the list's items. This allows the insertItem function to modify the list's items directly.

The `void deleteItem(ListNodePtr listPointer);` prompts the user to enter the name of the item they want to delete. It reads the user input and removes the trailing whitespace (newline character) from the input. It initializes two pointers, currentItem and nextItem, pointing to the first and second items in the list, respectively. If the item to delete matches the name of the first item in the list, it frees the memory allocated for that item and updates the listPointer to point to the next item in the list. If the item to delete is found later in the list, it adjusts the pointers to skip over the item to be deleted and frees its memory. If the item to delete is not found in the list, it informs the user that the item was not found.

Finally, the function prints out the edited board.

Edit lists

First, there is a submenu which gives the user the option to choose between (1) Editing the name of a list, (2) Adding a new list, (3) Deleting a list or (4) Going back to the main menu.

The function `void renameList(ListNodePtr pointerToFirst);` checks if the pointer to the first node of the list (pointerToFirst) is NULL, indicating that there is no list to rename. If so, it notifies the user and returns. It calls the findList function to locate the list to be renamed. If the list to be renamed is not found (i.e., findList returns NULL), the function returns. It prompts the user to enter a new name for the list. It reads the user input and removes any trailing newline character. It checks if the user entered a colon (:) at the end of the new name and removes it if present. Because of the nature of strings, it copies the new name into the name field of the list node, effectively renaming the list.

The function `void addList(ListNodePtr *sPtr);` prompts the user to enter the name of the new list. It reads the user input and removes the trailing whitespace (newline character) from the input. It checks if the last character of the input is a semicolon (:) and removes it if present. It calls the insertList function, passing the pointer to the pointer to the first node of the linked list of lists (sPtr) and the name of the new list (newList).

The function `void deleteList(ListNodePtr *sPtr);` checks if the pointer to the first node of the list of lists (*sPtr) is NULL, indicating that there is no list to delete. If so, it notifies the user and returns. It prompts the user to enter the name of the list they want to delete. It reads the user input and removes the trailing whitespace (newline character) from the input. It checks if the last character of the input is a semicolon (:) and removes it if present. It initializes two pointers, currentList and nextList, pointing to the first and second lists in the list of lists, respectively. If the list to delete matches the name of the first list in the list of lists, it frees the memory allocated for that list and updates *sPtr to point to the next list in the list of lists. If the list to delete is found later in the list of lists, it adjusts the pointers to skip over the list to be deleted and frees its memory. If the list to delete is not found in the list of lists, it informs the user that the list was not found.

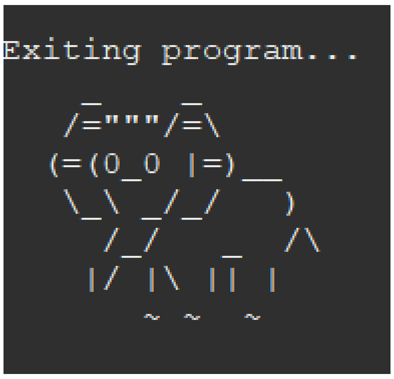
Finally, it prints out the edited board.

How do you save a board?

To save a board to a file the function `void saveBoard(ListNodePtr pointerToFirst);` is called and works as follows: it initializes a pointer currentList to point to the first list in the board. It checks if there is at least one list on the board by verifying if pointerToFirst is NULL. If not, it notifies the user that there is no board to save and returns. It prompts the user to enter a filename for the text file to save the board's contents. It reads the user input for the filename and removes the trailing whitespace (newline character) from the input. It attempts to open the specified file for writing. If the file cannot be opened or created, it notifies the user and returns. If the file is opened successfully, it iterates through each list in the board. For each list, it writes the name of the list followed by a colon (:) to the file (This helps with identifying what is a list name and what is an item later for reading from the file). For each item in the list, it writes the name of the item to the file. After writing all lists and their items to the file, it notifies the user that the board has been saved successfully. It closes the file.

For better handling and ensuring the program works correctly the user is prompted to only input .txt filenames.

What's your Ascii art about?



```
Exiting program...

    _    _
   /="""/=\
  (=(0_0 |=)___
   \_\ _/_/   )
     /_/    _  /\
    |/ |\ || |
       ~ ~  ~
```

Our ascii art is of an elephant!

List Git Repo , use your "group name" + " Assignment 3"?

Group BJ

Ghofran Abourayana & Simon Lasak

https://csgitlab.ucd.ie/ghofran/asssignment-3-bj.git