COMP 2401B -- Assignment #3

Due: Thursday, November 15, 2018 at 12:00 pm (noon)

Goal

Timmy Tortoise and Harold the Hare have been kidnapped by an evil wizard who wants them to organize his vast collection of magic spell books. Timmy's plan is to write a program that can rearrange the books very quickly at his command. He hopes that this might confuse the wizard long enough for our heroes to escape. Using the VM provided for the course, you will assist Timmy by implementing a C program that manages a collection of books as a linked list.

Learning Outcomes

You will practice problem solving and designing modular functions to implement a solution. You will work more extensively with pointers and collection structures by implementing a linked list in C, as well as working with double pointers and dynamically allocated memory.

Instructions

1. Data structures

Your program will define the following data types as structures:

- the `BookType` structure will contain a book's title and author (both strings), and the year of publication (an integer)
- the `ListType` and `NodeType` structures will implement a singly linked list, as we did in class, with both a head and a tail; do **not** use dummy nodes in this list

2. Book initialization function

You will write a book initialization function with the prototype:

```
int initBook(BookType **book);
```

This function will prompt the user for a book title, author, and year. If the user enters "end" for a book title, the function will return a failure code. If a different string is entered for a title, the user will be prompted for the author and year, then a book structure will be dynamically allocated and initialized with the data entered. The new book will be returned using the function's output parameter `book`, and success will be returned as the return value.

3. List manipulation functions

You will write several list management and manipulation functions with the following prototypes:

```
void initList(ListType *list);
void addByTitle(ListType *list, BookType *newBook);
void addByYear(ListType *list, BookType *newBook);
void copyList(ListType *src, ListType *dest);
void copyByYear(ListType *src, ListType *dest);
void delAuthor(ListType *list, char *name);
void printList(ListType *list);
void cleanupList(ListType *list);
void cleanupData(ListType* list);
```

The functions will be implemented as follows:

- the `initList()` function will initialize the fields of the given `list`
- the `addByTitle()` function will add `newBook` to the correct position in `list`, to keep the list in ascending alphabetical order by title
- the `addByYear()` function will add `newBook` to the correct position in `list`, to keep the list in descending order by year
- the `copyList()` function will add each book currently in the `src` list to the `dest` list, in ascending order by title; no changes should be made to the `src` list
- the `copyByYear()` function will add each book currently in the `src` list to the `dest` list, in descending order by year; no changes should be made to the `src` list
- the `delAuthor()` function will remove from `list` all the books by the author specified in `name`
- the `printList()` function will print all the data for the books contained in `list`; after all the books are printed out, the function will indicate which book corresponds to the head node, and which one corresponds to the tail node
- the `cleanupList()` function will clean up the memory associated with the nodes in `list`
- the `cleanupData()` function will clean up the memory associated with the books contained in `list`

Note: Any changes to the function prototypes provided will result in zero marks for that function

4. Program behaviour

Using the functions described above, your program will do the following:

- declare and initialize three `ListType` structures: `booksByTitle`, `booksByYear`, and `tmpList`
- prompt the user to enter book data, dynamically allocate the corresponding `BookType` structure, and add it by order of title to the `booksByTitle` linked list; data entry will end when the user enters “end” for a book title
- print out the `booksByTitle` list
- make a title-ordered copy of the `booksByTitle` list into `tmpList`
- prompt the user to enter the name of an author, and delete all of that author’s books from `tmpList`
- print both `booksByTitle` and `tmpList`, making sure that the books were deleted only from `tmpList`
- make a year-ordered copy of `booksByTitle` into `booksByYear`
- print both `booksByTitle` and `booksByYear`, checking that one is ordered by title and the other by year
- cleanup the book data and all three lists

A sample print out of a book list is shown below.

```
*** BOOK LIST BY TITLE ***
BOOK LIST:
--                                  Endymion by     Dan Simmons, Yr: 1996
--              Hitchhiker's Guide to the Galaxy by   Douglas Adams, Yr: 1979
--                                  Hyperion by     Dan Simmons, Yr: 1989
--             Life, the Universe and Everything by   Douglas Adams, Yr: 1982
--                             Mostly Harmless by   Douglas Adams, Yr: 1992
--          So Long, and Thanks for all the Fish by   Douglas Adams, Yr: 1984
--                           The Fall of Hyperion by     Dan Simmons, Yr: 1990
--     The Restaurant at the End of the Universe by   Douglas Adams, Yr: 1980
--                           The Rise of Endymion by     Dan Simmons, Yr: 1997

--> HEAD is: --                                    Endymion by     Dan Simmons, Yr: 1996
--> TAIL is: --                             The Rise of Endymion by     Dan Simmons, Yr: 1997
```

Constraints

- your program must be correctly designed and separated into modular, reusable functions
- your program must reuse functions everywhere possible
- your program must perform all basic error checking
- your program must be thoroughly documented, including each function and parameter
- compound data types must always be passed by reference
- all dynamically allocated memory must be explicitly deallocated
- the function prototypes provided must be used **exactly**, without any changes
- do not use any global variables

Submission

You will submit in *cuLearn*, before the due date and time, one `tar` or `zip` file that includes the following:

- all source code, including the code provided, if applicable
- a readme file that includes:
  - a preamble (program author, purpose, list of source/header/data files)
  - the exact compilation command
  - launching and operating instructions

Grading (out of 100)

Marking components:

- 16 marks: correct implementation of `initBook()` function
- 4 marks: correct implementation of `initList()` function
- 15 marks: correct implementation of `addByTitle()` function
- 15 marks: correct implementation of `addByYear()` function
- 8 marks: correct implementation of `copyList()` function
- 8 marks: correct implementation of `copyByYear()` function
- 16 marks: correct implementation of `delAuthor()` function
- 8 marks: correct implementation of `printList()` function
- 5 marks: correct implementation of `cleanupList()` function
- 5 marks: correct implementation of `cleanupData()` function

Deductions:

- Packaging errors:
  - 100 marks for an incorrect archive type that is not supported by the VM
  - 50 marks for an incorrect archive type that is supported by the VM
  - 10 marks for missing readme
- Major programming and design errors:
  - 50% of a marking component that uses global variables
  - 50% of a marking component that is incorrectly designed
  - 50% of a marking component that doesn't pass compound data types by reference
  - 100% of a marking component where the function prototype has been modified
- Minor programming and design errors:
  - 10 marks for consistently missing comments or other bad style
  - 10 marks for consistently failing to perform basic error checking
  - 10 marks for memory leaks
- Execution errors:
  - 100% of a marking component that cannot be tested because it doesn't compile or execute in VM
  - 100% of a marking component that cannot be tested because it's not used in the code
  - 100% of a marking component that cannot be proven to run successfully due to missing output