Background

The most common data structure is probably an **array**, with which you have a lot of experience at this point. An array is a collection of elements stored in contiguous memory locations which can be easily accessed through array indices.

Possibly the second most common data structure is a **linked list**. A linked list is a linear sequence of data structure elements which are not necessarily contiguously stored in memory but which contain links to at least one other data structure element in the list. These **links** are in the form of pointers which point to the next (and possibly the previous) elements in the list. Thus, each data structure element in a linked list has its own data (characters, integers, floating point numbers, etc.) and its own pointers to other elements. Each data structure element is also called a **node**.

There are two main advantages to a linked list over an array. First, a linked list can have a dynamic size that can change as data is stored and used, while an array has a fixed size that must be known before data can be stored. Elements of a linked list can be added dynamically by using the `malloc()` function. Second, a linked list allows for easy insertion and deletion of data elements since only the pointers need to be changed, while smooth insertion and deletion for an array is much more difficult since elements must be shifted in memory.

Of course, there are two main disadvantages to a linked list over an array, too. First, more memory overall is required for a linked list since the pointers must be stored as well. Second, it is more difficult to randomly access a specific element in the middle of a linked list since the data is not contiguous and thus an index will not work.

In a **singly linked list**, each element only has one pointer link which points forward to the next element in the list. In a **doubly linked list**, each element has two pointer links: one which points forward to the next element in the list, and one which points backward to the previous element in the list. The main advantages to a doubly linked list are that it can be traversed both forward and backward, and that insertion and deletion of elements is typically more efficient. The main disadvantages to a doubly linked list are that it requires more memory for the second pointer, and that it is typically more complicated to implement. In this assignment, we will focus on implementing a doubly linked list.

Figure 1 below shows a representation of a doubly linked list. Note that the previous pointer for the first element and the next pointer for the last element are both NULL since there are no elements there. Also, note that the first element of a linked list is called the **head**, and the last element of a double linked list is called the **tail** (not labelled below). By keeping track of the head and tail, a doubly linked list can be traversed in either the forward or backward direction.

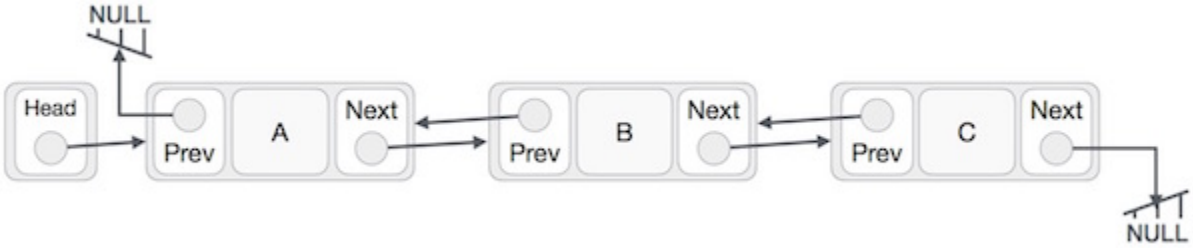

Figure 1: Representation of a doubly linked list from https://www.tutorialspoint.com/data_structures_algorithms/doubly_linked_list_algorithm.htm

In this assignment, you will implement four main functions for a doubly linked list: store already ordered data from an input file into a linked list, display the linked list in forward order, display the linked list in backward order, and insert a new node element. Within the last function, there are three options for inserting a new node: insert before the head, insert after the tail, and insert before another node in the middle of the linked list. Whenever a new node is inserted, you must set its previous and next pointer links appropriately, and you must change the previous and next pointer links of all surrounding nodes.

Problem Statement

You are developing an online movie information database which will allow you to add information about current and future movies. The website is still in early development, and right now, each movie just has a title (character array), a year released (integer), and a director (character array). You are provided with an input file containing some test movies. The first line of the file has the number of movies present in the file. Each line after that has the title of a movie, the year that movie was released, and the last name of the director for that movie. The movies in the file are already in forward alphabetical order. You will read in all of the data for these movies into a dynamically created doubly linked list.

Then, you will implement functions to display the movies in forward title alphabetical order, to display the movies in backward title alphabetical order, and to create a new movie and insert it into the proper alphabetical place in the linked list. You will also display the node addresses and pointer links for all movies when they are displayed so you can verify that the linked list structure works properly.

Instructions

- Complete the C33-1 assignment which introduces structs in C.
- Copy `APP_C33_1_EXT_SKELETON.cpp` and `APP_C33_1_EXT_movies.txt` from:

  `/share/EED/class/engr1281/students/c/Class_33/Application/`

- Begin by opening the skeleton code and adding your header information.
- Most of the code is provided for you, but you must fill in the provided blanks throughout and write the code for the `display_backward()` function based on the code for the `display_forward()` function.
- Test your program by displaying the results forwards and then backwards. Then add the following movies and display the results forwards and backwards again:

Title	Year	Director
Toy_Story	1995	Lasseter
Alien	1978	Scott
Dunkirk	2017	Nolan

- **Optional Challenge:** After you get the code working to insert new movies and display in alphabetical title order, we encourage you to try to implement new functions to display the movies in ascending and descending year or director alphabetical order. There are many ways to do this, so, have fun figuring it out!