

# Assignment 2: Linked Lists and Sets

## Jake Gendreau

## Program Design

## February 8, 2024

---

I estimate that implementing this program would take me 1.5 - 2.5 hours. I understand the concepts quite well, but I'm pretty rusty on file reading and linked lists.

## Data structure: Node

---

1. A pointer to the next node, called next.
2. A string called data.

## Main function

---

1. Create two null node pointers corresponding to data sets 1 and 2.
2. Use `readFile()` to get the words from data sets 1 and 2 into linked lists.
3. Use `findIntersect()` to make a new linked list containing the intersect between the two data sets.  
 $D1 \cap D2$
4. Use `concatList()` to make a new linked list containing the combination of the two linked lists.  
 $D1 + D2$
5. Use `subtractIntersect()` to subtract the intersect linked list from the combination linked list and get the union.  
 $D1 \cup D2 = D1 + D2 - (D1 \cap D2)$
6. Print out the intersect linked list and union linked list using `printList()`

## Required Functions

---

- `readFile()` - A function to read the files.
- `appendToList()` - A function to add an item to a linked list.
- `deleteNode()` - A function that can delete an item from a linked list.
- `printList()` - A function to print a linked list.
- `searchList()` - A function to search a list for a term.
- `findIntersect()` - A function to find the intersect between two lists.
- `subtractIntersect()` - A function that can subtract the intersect from two lists.
- `concatLists()` - A function that can combine two lists.
- `isAlpha()` - Determine whether an input character is part of the alphabet or not
- `cleanWord()` - Remove punctuation from an input word

## `readFile(file, head)`

1. Each infile will have its own linked list.

2. Read off each word of the file.
3. Set the word equal to cleanWord(word)
4. Call searchList() to see if the word already exists in the linked list.
  - If it does already exist in the list, do not append the current word to the list.
  - Otherwise, use appendToList() the current word to the list.

## appendToList(data)

1. Make a new node that points to head with data passed into the function.
2. Add new node to the front of the list.

## deleteNode(query)

1. Make a temporary node = head.
2. Temp node iterates through the linked list
  - If the data of the temp node = query, then delete the temp node, and connect the next of the previous node to the node following the temp node.

## printList(head)

1. Make a temp node = head.
2. Temp node traverses through the list, printing it's data as it goes.

## searchList(query, head)

1. Make a temp node = head.
2. Temp iterates through the list.
  - If the data in temp = query, return true.
  - Otherwise, return false.

## findIntersect(head1, head2)

1. Make a new node called intersect.
2. Make a new temp node = head1.
3. Temp node traverses through the list
  - if searchList(query, head2) is true, then the element is in both lists, so use appendToList(query, intersect).
4. Return intersect

## subtractIntersect(head1, head2)

*head 1 is the intersect, and head2 is the combination of the two file lists*

1. Make a temp node p = head1
2. Make a new string called query.
3. p Iterates through the list
  - If the data in p = query, then use deleteNode(query, head2) to get rid of the duplicate element.

## concatList(head1, head2)

1. Make a new node called union
2. Make a temp node called p = head1.
3. Have p traverse through head1, appending to union using appendToList() as it goes.
4. Have p = head2.
5. Have p traverse through head2, appending to union using appendToList() as it goes.
6. Return union

## isAlpha(char)

1. Test if character  $\geq$  'a' and  $\leq$  'z' or  $\geq$  'A' and  $\leq$  'Z'
2. Return true if it is, false if it isn't

## cleanWord(string)

1. Take in a string
2. Make a buffer string
3. Iterate through the characters of the string
4. Append to buffer if isAlpha(character) is true
5. return buffer