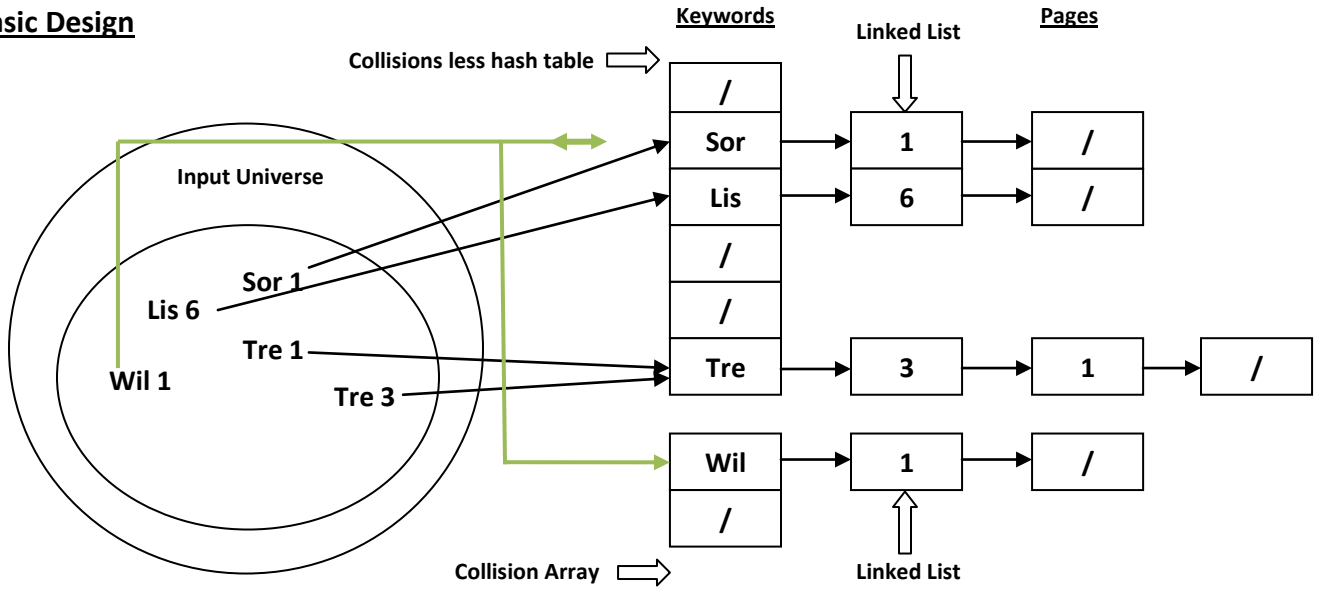
PROJECT REPORT

Programming Assignment 2011
Course: CS2022 Data Structures & Algorithms
Name: C.U. Kumarasinghe

Index No: 100282N

A. DESIGN

Basic Design



Object Oriented Design

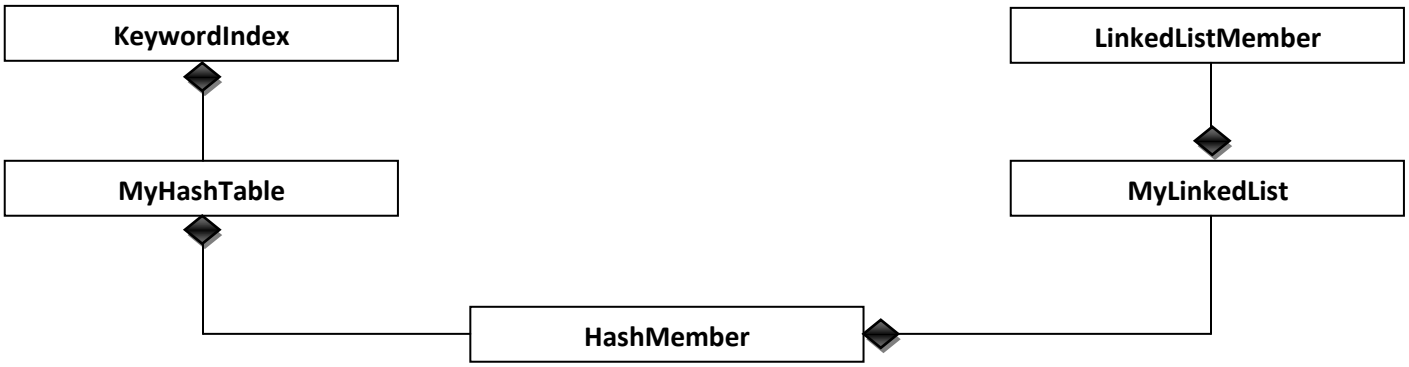
KeywordIndex	
- line: String	- inputFile: BufferedReader
- keywordListStart: boolean	- outputFile: BufferedWriter
- queriesStart: boolean	
- HT: MyHashTable	
+ try()	

MyHashTable
+ SIZE_OF_HASHTABLE: static final int
+ MAX_NUM_KEYWORDS: static final int
- members: HashMember[]
- collisionMembers: HashMember[]
- collisionMembersCounter: int
- theKeywordsOnThePage: String[]
+ MyHashTable(): void
+ addMemberToTable(String keywordToBeAdded, int page): boolean
+ getMemberPages(String keywordToBeRetrieved, BufferedWriter BW): boolean
+ getFirstPage(String keywordFirstPageToBeRetrieved, BufferedWriter BW): boolean
+ getKeywords(int thePageEnterd, BufferedWriter BW) : boolean
+ showKeywords(BufferedWriter BW): void
+ getHashCode(String keyword): int

HashMember
- keyword: String - pages: MyLinkedList
+ HashMember(): void + getKeyword(): String + pages(): MyLinkedList

MyLinkedList
+ SIZE_OF_LINKEDLIST: static final int - first: LinkedListMember - sortedPages: int[] - numberOfPages: int - listIsAlreadySorted: boolean
+ MyLinkedList(): void + insert(int thePage): void + sortPages(BufferedWriter BW): void + storeToSortedPagesList(int[] pages) : void + showFirstpage(BufferedWriter BW) : void + printFirstPage(BufferedWriter BW) : void + showPagesList(BufferedWriter BW) : void + printPagesList(BufferedWriter BW) : void + getSortedPagesList(): int[] + isListAlreadySorted(): boolean +getNumberOfPages(): int

LinkedListMember
- page: int - nextLinkedListMember: LinkedListMember
+LinkedListMember(int thePage, LinkedListMember theNextLinkedListMember): void +getPage(): int +getNextLinkedListMember(): LinkedListMember +setPage(int tempPage): void +setNextLinkedListMember(LinkedListMember theLLM): void



B. THE DATA STRUCTURES AND ALGORITHMS USED

DATA STRUCTURES

1. Hash Table with chaining
2. Linked List

When I saw the project instantly one word came to my notice that made my decision easy. That word was "keyword". As I read through the project I notice that implementing a tree or graph was not useful as a hash table for this instance. The first two operations require the keyword to be searched. To do this in minimum time hash table is the best option as it takes only O(1) time for that. I may be proved wrong but that was my decision.

To implement a hash table which uses keywords as keys and it requires a linked list or an double linked list so that it can store page values. As I found it difficult to implement the double linked list I ended up implement a linked list only. Double linked list would have been the best option it the pages required to be sorted. But I ended up implementing the sorting with more memory been wasted.

ALGORITHMS

1. Bubble Sort
2. Bucket Sort

As there is not a large amount of pages when considering number of keywords for faster operation got me to look at Bucket sort. It is one of the fasters integer sorting algorithms around. But it cost a lot of memory. As I wanted to get the faster sorting than the memory limitation I used that. Otherwise merge sort would have been a good choice.

The only choice when comparing strings was bubble sort. As in the instructions said that the third operation is less frequent, number of keywords may be really high and as I used a lot of space for the bucked sorted pages less memory consumption was the decision maker there. So selection, merge or quick sort was disregarded.

C. ASSUMPTIONS

- Page values are between 0 to 200, as in the instructions it only said the maximum number of pages are 200 there can be page values more than 200.
- Keywords doesn't contain numeric values in them.

D. PROBLEMS FACED

- I tried to impelment a hash table with self replecating ability, that is to hash the new keyword in a new hash table when a collision happens, it would have been able to insert an unlimited amount of members untill memory limitation, but it didn't do all the required operations. Then I went through to implement a hash table with a array of collision members with chaining.
- The input and output file location will needed to be changed in order to get the program running.

E. DISCUSSION

- If the first two operations are to be more memory efficient implementing a merge sort would be ideal.
- Rather than using linear collision handling(chaining) if open addressing can be achived memory efficiency will be more.