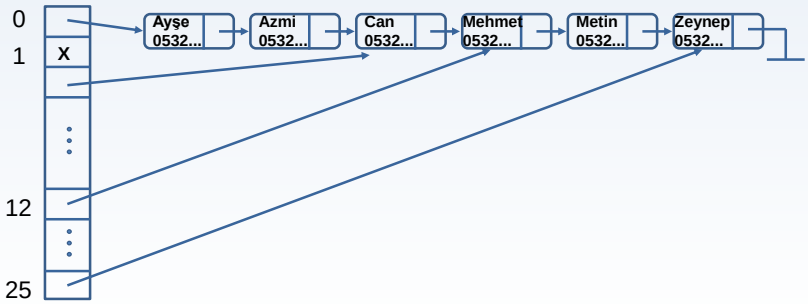
Data Structures

List Applications

List Application– Lecture Example

- Searching in a linked list can become a very time consuming operation when there are many list elements.
- For example, in an ordered phone book containing 3000 phone number records, searching for the record "Zeynep Ata" may be a very costly operation.
- The search function given in the lecture example operates by traversing the list records from the very beginning and hence is an expensive method.

- To speed up the search operation and access to data, we could design a new data structure
- In this data structure, letter indices (corresponding to their order in the alphabet) are held in an array.
- Thus, faster access to records starting with the same letter could be provided.
- For example, the records starting with letter 'M' could be accessed using the 13. element of the array (array[12]).



This index array is defined as follows:

```
#ifndef LIST_H
#define LIST_H
#include "node.h"

struct List{
  Phone_node *head;
  Phone_node *index[26];
  int nodecount;
  char *filename;
  FILE *phonebook;
  void create();
  void close();
  void makeEmpty();
  void insert(Phone_node *);
  void remove(int ordernum);
  int search(char *);
  void update(int recordnum, Phone_node *);
  void read_fromfile();
  void write_tofile();
};
#endif
```

Adding

- The function for adding to a linked list should be written in such a way that it also updates the index array mentioned above.
- Note: There will be no uppercase/lowercase distinction ("Ahmet" and "ahmet" will be located next to each other in the list).
- We assume that names start with only the letters of the alphabet (a-z, A-Z).
- Hint 1: We can ignore the case of names beginning with special Turkish characters and use their ASCII values.
- Hint 2: The function "int tolower(int)" returns the lowercase version of a given character.


```
void List::insert(Phone_node *toadd){
   Phone_node *traverse, *behind, *newnode;
   traverse = head;
   newnode = new Phone_node;
   *newnode = *toadd;
   newnode->next = NULL;
   int ch = tolower(newnode->name[0]);
   if (index[ch - 'a'] == NULL)
          index[ch - 'a'] = newnode;
   // first node being added
   if (head == NULL) {
          head = newnode;
          nodecount++;
          return;
   }
                                        // add to head of list
                                        if (stricmp(newnode->name, head->name) < 0){
                                               newnode->next = head;
                                               head = newnode;
                                               nodecount++;
                                               index[ch - 'a'] = newnode;
                                               return;
                                        }

                                        while (traverse &&
                                               (stricmp(newnode->name, traverse->name) > 0)){
                                               behind = traverse;
                                               traverse = traverse->next;
                                        }

                                        newnode->next = traverse;
                                        behind->next = newnode;
                                        if (tolower(behind->name[0])!=tolower(newnode->name[0]))
                                               index[ch -'a'] = newnode;
                                        nodecount++;
                                        }
                                      }
```

Searching

- The function for searching in a linked list is written in a way to conduct fast search using the data structure above.
- The parameters that will be passed to this function can only start with the letters of the alphabet (a-z, A-Z).
- The parameter could be a single letter or several letters.
- Similar to the search function created in class, when a single letter is entered, all records starting with that letter will be displayed.
- The function will be written to make the fewest possible number of comparisons.


```
int List::search(char *target)
{
   Phone_node *traverse;
   int counter = 0;
   int found = 0;
   traverse = index[target[0] - 'a'];

   while ( traverse && tolower(traverse->name[0]) == tolower(target[0]) ){
     counter++;

     if ( strnicmp(traverse->name, target, strlen(target)) == 0) {
        found++;
        cout << counter <<"." <<traverse->name<<" "<<traverse->phonenum<< endl;
      }
     traverse = traverse->next;
   }
   return found;
}
```