Operations on Linked Lists

When Needed

- Dynamic allocation is specially appropriate for building lists, trees and graphs.
- We used an array for storing a collection of data items.
- Suppose the collection itself grows and shrinks then using a linked list is appropriate.
- It allows both insertion, deletion, search.
- But the random access capabilities of array is lost.
Operations on Linked Lists

Declaring a Node Type

- Linked list is a collection of data item, where each item is stored in a structure (node).
- The structure for node can be declared as follows:

```c
struct node {
    int info;
    struct node *next;  // ptr to struct of identical type
}
```
Operations on Linked Lists

Head of a List

- Linked list is accessed by accessing its first node.
- Subsequent nodes can be accessed by using the pointer `next`.
- So after declaring, a (empty) list is initialized to `NULL`.
- Creating a list is done by creating nodes one after another.
Operations on Linked Lists

Creating Nodes

```c
struct node *newNode;

// Allocate space for new node.
newNode = malloc(sizeof(struct node));

// Set information to be stored.
(*newNode).info = 0;

newNode->info = 0; // alternative way of referencing fields.
```
Inserting a Node at Beginning

insert into an empty list

inserting into a nonempty list
Operations on Linked Lists

Inserting a Node at Beginning

There are two limiting cases that should be handled properly:

- List could be an empty list.
- The element to be inserted may already be present.

```c
#include <stdio.h>
#include <stdlib.h>
struct node {
    int info;
    struct node *next;
};
```
Operations on Linked Lists

Inserting a Node at Beginning

```c
struct node *searchList(struct node *list, int n) {
    struct node *p = list;
    while (p != NULL) {
        if (p->info == n)
            break;  // Search successful
        else
            p = p->next;  // Check next node
    }
    return p;
}
```
Operations on Linked Lists

Inserting a Node at Beginning

```c
struct node *addToList(struct node *list, int n) {
    struct node *newNode;

    if (searchList(list, n) != NULL) {
        printf("%d exists, add not allowed\n", n);
        return list;
    }

    else {
        newNode = malloc(sizeof(struct node));
        if (newNode == NULL)
            printf("Creation of node failed\n");
        else {
            newNode->info = n;
            newNode->next = list;
        }
    }

    return newNode; // Becomes the first node of the list.
}
```
Operations on Linked Lists

Inserting a Node at Beginning

```c
void printList(struct node *list) {  // Print the list
    struct node *p = list;
    while (p != NULL) {
        printf("%5d", p->info);
        p = p->next;
    }
    printf("\n");
}

int main() {
    int i;
    struct node *first = NULL;
    for (i = 0; i < 5; i++)
        first = addToList(first, (i+1)*10);
    printList(first);
}
```
Append to a Linked List

- Navigate the linked list to reach the last node:
  - Create a `newNode`, set its `info` field to value provided.
  - Set `next` of `newNode` to `NULL`.
  - Set the `next` field of the last node in the linked list to point to `newNode`.

- Handle the case of empty linked list (return pointer to `newNode`)
Operations on Linked Lists

Append to a Linked List

Appending to an empty list:
- First node
- New node
- Appending to empty list

Appending to a nonempty list:
- First node
- New node
- Appending to nonempty list
- EndList(first)
Find the Tail of a Linked List

```c
struct node *endList(struct node *list) {
    struct node *p = list;

    if (p == NULL)
        return NULL; // Empty linked list
    while (p->next != NULL)
        p = p->next;
    return p; // Last node of the linked list
}
```
Operations on Linked Lists

Append to a Linked List

```c
struct node *appendToList(struct node *list, int n) {
    struct node *newNode, *p;
    if (searchList(list, n) != NULL)
        printf("%d exists in the list, append not allowed\n", n);
    else {
        newNode = malloc(sizeof(struct node));
        newNode->info = n;
        newNode->next = NULL;
        p = endList(list);
        if (p != NULL)
            p->next = newNode;
        else
            list = newNode;
    }
    return list;
}
```
Operations on Linked Lists

Delete from a Linked List

Deletion a middle node

Deletion of first node