COM 201 – Data Structures and Algorithms

Linked Lists - 2

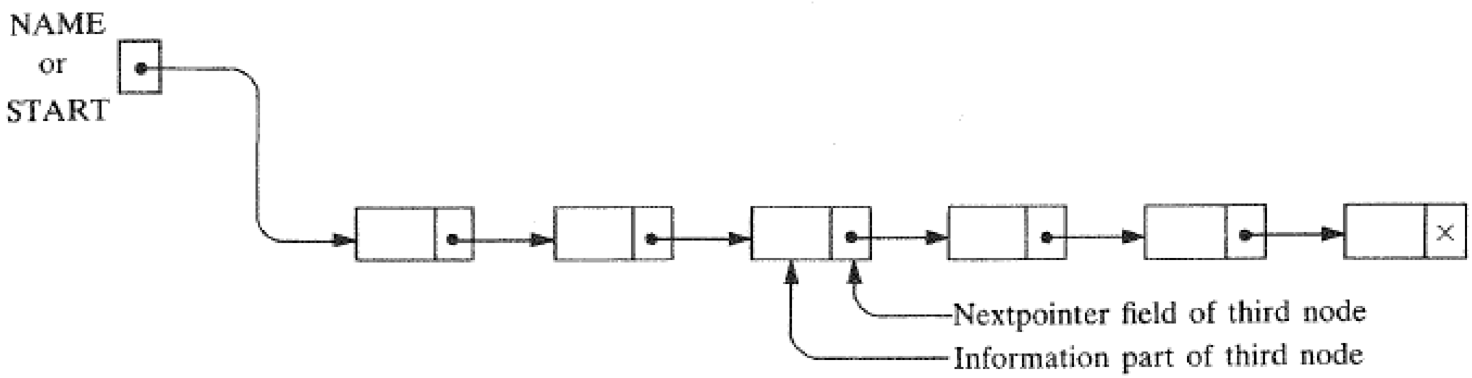
Assist. Prof. Özge ÖZTİMUR KARADAĞ

Department of Computer Engineering – ALKÜ

Alanya

Previously

- Linked Lists
  - Element field
  - Link field (next pointer)
- Operations
  - Traverse
  - Search
- Memory Allocation


NAME
or
START
×
Nextpointer field of third node
Information part of third node

Today

- Representation of Linked List in C programming Language

- Operations
    - Insertion

    - Deletion

Linked Lists in C

- Defining a linked list in C
  - We can represent a node using structures.
  - A linked list node with integer data:

```
struct node {
        int val;
        struct node * next;
} node_t;
```

Linked Lists in C

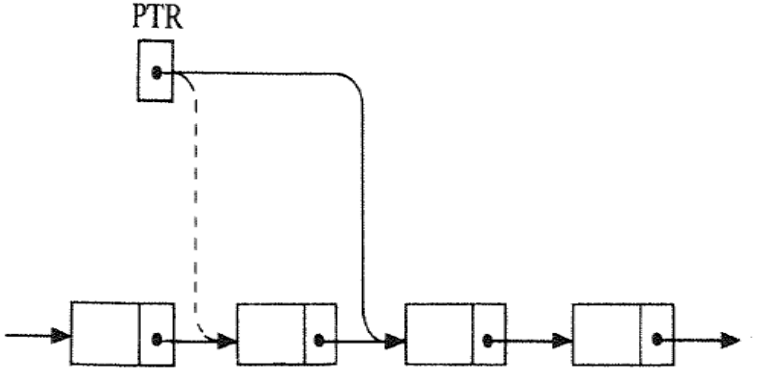
- Construction of a linked list with two nodes

```
#include <stdio.h>
#include <stdlib.h>

struct Node {
        int data;
        struct Node* next;
};

int main()
{
        struct Node* head = NULL;
        struct Node* second = NULL;
        head = (struct
Node*)malloc(sizeof(struct Node));
        second = (struct
Node*)malloc(sizeof(struct Node));
        head->data = 1; // assign data in first
node
        head->next = second; // Link first node
with
        second->data = 2;
        second->next = NULL;
        return 0;
}
```

Linked Lists in C

- Traversing a Linked List

```
void printList(struct Node *n){
        while(n!=NULL){
                printf(" %d ",n->data);
                n=n->next;
        }
}
```



Linked List in C

- Searchig a Linked List
  - Unsorted list:

```
struct Node* searchUnsorted(struct Node* head, int x){
        struct Node* current = head;
        while(current!=NULL)
        {
                if (current->data==x)
                        return current;
                current = current->next;
        }
        return NULL;
}
```

Linked List in C

- Searchig a Linked List
    - Sorted list:

```
struct Node* searchSorted(struct Node* head, int x){
    struct Node* current = head;
    while(current!=NULL)
    {
        if (x> current->data)
            current = current->next;
        else if (x==current->data)
            return current;
        else
            return NULL;
    }
    return NULL;
};
```

Linked List in C

- Insertion:
  - Insert at the beginning
  - Insert to a specific location
  - Insert into a sorted list

Linked List in C

- Insert at the beginning

```
void insertAtBeginning(int data)
{
   struct Node* newNode;
   newNode = (struct Node*) malloc(sizeof(struct Node));
   newNode->data = data;
   newNode->next = head;
   head = newNode;
   return;
}
```

Linked List in C

- Insert to a specific location

```
void insertLOC(int data,struct Node* loc)
{
    if (loc==NULL)
        insertAtBeginning(data);
    else
    {
        struct Node* newNode;
        newNode = (struct Node*)malloc(sizeof(struct Node));
        newNode->data = data;
        newNode->next = loc->next;
        loc->next = newNode;
    }
}
```

Linked List in C

- Insert into a Sorted List

```
void insertSortedList(int data){

     struct Node *newNode, *current, *loc,*save;
    if(data<head->data){
                 insertAtBeginning(data);
                 return;
   }

    newNode = (struct Node*)malloc(sizeof(struct Node));
    newNode->data=data;
    save=head;
    current=head->next;
    while(current !=NULL){
        if(data<current->data){
            loc=save;
            return;
        }
        save = current;
        current = current->next;
    }
    loc=save;
    insertLOC(data,loc);
}
```

Linked List in C

- Deletion
  - Delete a node following a given node
  - Delete a node with a given ITEM of information

Linked List in C

- Delete a node following a given node

```
void deleteLoc(struct Node* loc)
{
   struct Node *tmp;
   tmp=loc->next;
   if(loc->next->next==NULL)
      loc->next =NULL;
   else
      loc->next = loc->next->next;
   free(tmp);
}
```

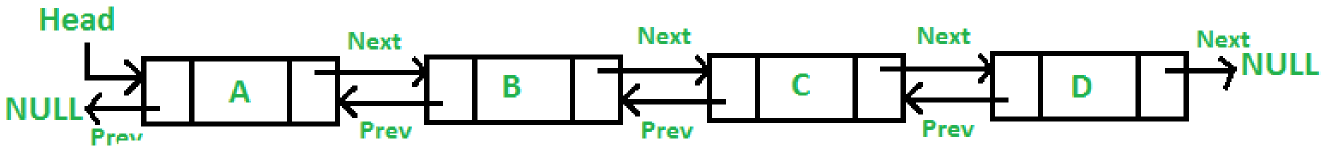
Linked List in C

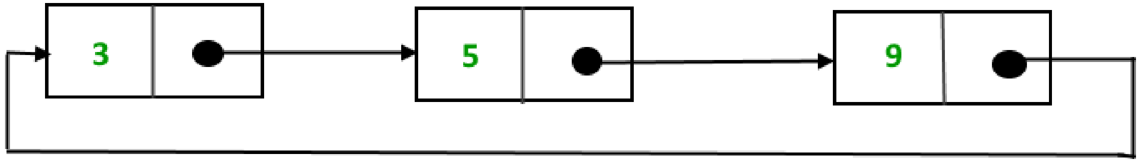
- Delete a node with a given ITEM of information

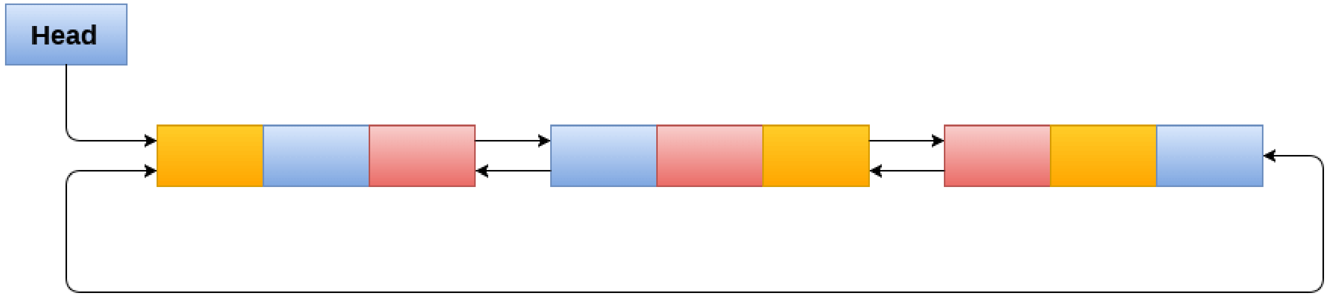
```
void deleteNode(int x)
{
    struct Node *prev,*current;
    current=head;
    while(current!=NULL){
        if(x==current->data)
            if (current==head){              //delete the first node
                head = head->next;
                free(current);
                current=head;
            }
            else{
                prev->next = current->next;
                free(current);
                current=prev->next;
            }
        else{
            prev = current;
            current = current->next;
        }
    }
}
```

Types of Linked Lists

- Singly linked lists
- Doubly linked lists
- Circular linked lists
- Circular doubly linked lists







Doubly Linked List

- Advantages
  - A DLL can be traversed in both forward and backward directions.
  - One can quickly insert a node before a given node.
  - In a singly linked list, to delete a node, need a pointer to the previous node. To get this node, sometimes the list is traversed. DLL can get the previous node using the previous pointer.

- Disadvantages
  - Every node of DLL require extra space for a previous pointer.
  - All operations require an extra pointer to be mantained.

Doubly Linked List

- Add a node after a given node

```
void insertAfter(struct Node* prev_node, int new_data)
{
        /*1. check if the given prev_node is NULL */
        if (prev_node == NULL) {
                printf("the given previous node cannot be NULL");
                return;
        }

        /* 2. allocate new node */
        struct Node* new_node= (struct Node*)malloc(sizeof(struct Node));

        /* 3. put in the data */
        new_node->data = new_data;

        /* 4. Make next of new node as next of prev_node */
        new_node->next = prev_node->next;

        /* 5. Make the next of prev_node as new_node */
        prev_node->next = new_node;

        /* 6. Make prev_node as previous of new_node */
        new_node->prev = prev_node;

        /* 7. Change previous of new_node's next node */
        if (new_node->next != NULL)
                new_node->next->prev = new_node;
}
```