



Faculty of Engineering and Technology
Computer Science Department

Linked Lists

Struct : Review

```
struct node;  
typedef struct node *ptr;  
struct node{  
    int ID;  
    float price;  
    char *name;  
    ptr next;  
};
```

```
typedef ptr list;  
typedef ptr pos;
```

What is the problem of Arrays?

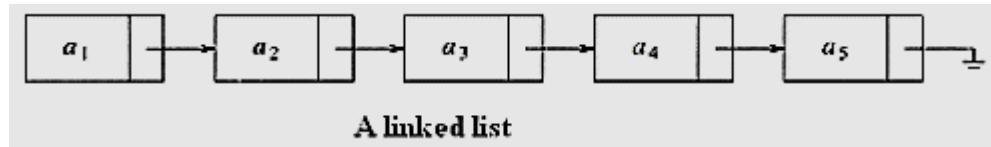
- Arrays has fixed size.
- If programmers allocate large enough size, majority of space is wasted.
- Insertion of elements occurs at the last index. If not, it will consume more time for shifting.
- Deletion?

Lists

- List: is a collection of elements that are organized sequentially.
- Linked lists and arrays are similar since they both store collections of data.
- The *array's* features all follow from its strategy of allocating the memory for all its elements in one block of memory.
- *Linked lists* use an entirely different strategy: linked lists allocate memory for each element separately and only when necessary.

Linked Lists

- The linked list consists of a series of structures called nodes.
- Each node contains two fields:
 - "data" field
 - "next" field which is a pointer used to link one node to the next node.



```
struct node{  
    int ID;  
    float price;  
    char *name;  
    ptr next;  
};
```

Types of Linked Lists

- Single (normal) Linked List
- Doubly Linked List
- Single **circular** linked list
- Doubly **circular** linked list

Operations on Linked List

- Creation
- isEmpty (check if the list has no nodes)
- Insertion
- Deletion
- Display (print all/some of its nodes)
- getSize (number of elements)
- Search List for a specific node/s.
- Sort List based on specific criterion.

Creation

```
Linked_List L;
```

```
pos P;
```

```
L=(Linked_List) malloc(sizeof(struct node));
```

```
L->next=NULL;
```

Creation

```
Linked_List L;
pos P;
L=(Linked_List) malloc(sizeof(struct node));
L->next=NULL;
int id;
float pr;
char nm[20];
printf("Enter ID , price and name \n");
scanf("%d%f%s", &id, &pr, nm);
while (id>=0){
    ptr temp;
    temp =(ptr) malloc(sizeof(struct node));
    if(temp !=NULL)
    {
        temp->ID=id;
        temp->price=pr;
        strcpy(temp->name,nm);
    }
    else
        printf("ERROR out of memory\n");

    insert(L, L, temp);

    printf("Enter ID , price and name for the next unit\n");
    scanf("%d%f%s", &id, &pr, nm);
}
```

Insert

```
void insert(Linked_List L, pos p, ptr temp)
{
    if(temp !=NULL && L != NULL)
    {
        temp->next=p->next;
        p->next=temp;
        printf("Node # %d is inserted \n", temp->ID);
    }
    else
        printf("ERROR either Linked List L or the new node is NULL \n");
}
```

Display all nodes

```
void Display_List(Linked_List L)
{
    pos p = L->next;
    while(p != NULL)
    {
        printf("%d \n", p->ID);
        p=p->next;
    }
}
```

Write a recursive
function for Display

Output?

- What is the output if we insert 5 nodes, then we call the function Display?
- What you can conclude?
- What is the time complexity for insertion?

Insert at last

```
void insert_atLast(Linked_List L, ptr newNode){  
  
    if(newNode !=NULL && L != NULL)  
    {  
        if(L->next==NULL)  
        {  
            newNode->next=L->next;  
            L->next=newNode;  
            printf("Node # %d is inserted \n", newNode->ID);  
        }  
        else  
        {  
            pos temp;  
            for(temp=L; temp->next!=NULL; temp=temp->next)  
                ;  
            temp->next=newNode;  
        }  
    }  
    else  
        printf("ERROR either Linked List L or the new node is NULL \n");  
}
```

Time complexity ?

isEmpty?

```
int isEmpty(Linked_List L)
{
    return L->next == NULL;
}
```

Time
complexity ?

Delete

- You can't delete from an empty list
- How delete operation is performed?

Time
complexity ?

- Which node to delete?
 - ith node
 - Any node with specific properties.
 - All nodes

delete_ithNode

```
void delete_ithNode(Linked_List L, int i)
{
    ptr t=L, d;
    int a=0;
    if(!isEmpty(L))
        while (a != i-1 && t->next != NULL)
    {
        ++a;
        t=t->next;
    }
    d=t->next;
    t->next=d->next;
    free(d);
}
```

Exercises

- Write delete node function based on specific node name
- Write iterative and recursive function to delete all nodes.
- What is the time complexity for your functions?

getSize

```
int getSize(Linked_List L){  
    int size=0;  
    pos t=L;  
    while(t->next != NULL){  
        ++size;  
        t=t->next;  
    }  
    return size;  
}
```

Time complexity ?

Can you return the size in O(1)

Search

- Several searching algorithm, e.g. linear search

```
pos search_List(Linked_List L, char key[30]){
    pos t=L->next;
    while(t != NULL && strcmp(key,t->name)!= 0)
    {
        t=t->next;
    }
    return t;
}
```

Exercise

- Compare between Linked Lists and Arrays for the following operations

Operation	Linked Lists	Arrays
Flexibility		
Creation		
Insertion at the last		
Insertion at the first		
Delete a specific node		
Get actual size		
Search		
Sort		
Print all nodes		