❖Linked Lists

Operations of List

* **IsEmpty:** determine whether or not the list is empty
* **InsertNode:** insert a new node at a particular position
* **FindNode:** find a node with a given value
* **DeleteNode:** delete a node with a given value
* **DisplayList**: print all the nodes in the list

Finding a node

- int FindNode(double x)
  - Search for a node with the value equal to x in the list.
  - If such a node is found, return its position. Otherwise, return 0.

```
int List::FindNode(double x) {
Node* currNode   =      head;
int currIndex        =      1;
while (currNode && currNode->data != x) {
        currNode     =      currNode->next;
        currIndex++;
}
if (currNode) return currIndex;
return 0;
}
```

Deleting a node

* int DeleteNode(double x)

  Delete a node with the value equal to x from the list.

  If such a node is found, return its position. Otherwise, return 0.

* Steps

  Find the desirable node (similar to FindNode)

  Release the memory occupied by the found node

  Set the pointer of the predecessor of the found node to the successor of the found node

* Like InsertNode, there are two special cases

  Delete first node

  Delete the node in middle or at the end of the list

```
int List::DeleteNode(double x) {
Node* prevNode      =   NULL;
Node* currNode      =   head;
int currIndex       =   1;
while (currNode && currNode->data != x) {
                        prevNode  =   currNode;
                        currNode  =   currNode->next;
                        currIndex++;
}
if (currNode) {
            if (prevNode) {
                    prevNode->next   =   currNode->next;
                        delete currNode;
                    }
                    else {
```

```
                    head = currNode->next;
                    delete currNode;
                }
                return currIndex;
}
return 0;
}
```

Printing all the elements

* void DisplayList(void)

  Print the data of all the elements

  Print the number of the nodes in the list

```
void List::DisplayList()
{
  int num = 0;
  Node* currNode = head;
 while (currNode != NULL){
cout << currNode->data << endl;
currNode = currNode->next;
num++;
  }
  cout << "Number of nodes in the list: " << num << endl;
}
```

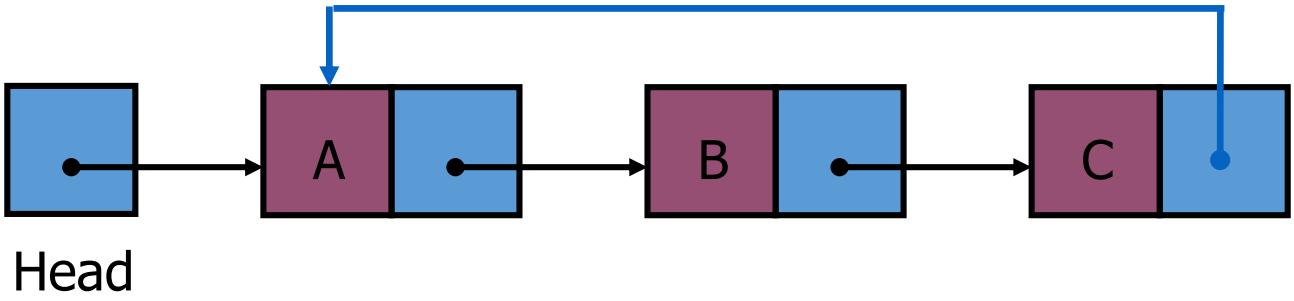
Destroying the list

- ~List(void)
  - Use the destructor to release all the memory used by the list.
  - Step through the list and delete each node one by one.

```
List::~List(void) {
  Node* currNode = head, *nextNode = NULL;
  while (currNode != NULL)
  {
nextNode = currNode->next;     // destroy the current node
delete currNode;
currNode = nextNode;
  }
}
```

Variations of Linked Lists

- *Circular linked lists*
- The last node points to the first node of the list



How do we know when we have finished traversing the list? (Tip: check if the pointer of the current node is equal to the head.)

Ex1: write program to calculate the number of even and odd numbers in linked list.

```
#include<iostream.h>
struct nod{
int info ;
nod *next ;    //odd ,even
};

main()
{ int o=0,e=0;
nod *p,*q,*f;
```

```
f=new nod;    // أنشئ أول نود وتعتبر هذه هي رأس المتسلسلة ويجب أن يبقى على أول نود
cout<<"numbers: ";
cin>>f->info;
 f->next=0;
p=f;
for(int i=1;i<=3;i++)  //  كل ما أنشئ نود جديدة أربطة بالنود السابقة وأقدم المؤشر
{
   q=new nod;
   cin>>q->info;
    q->next=0;
    p->next=q;
    p=q;
}
p=f;
while(p!=0)
{
if(p-> info %2==0)
  o++;
  else  e++;
p=p->next;
} cout<<"odd= "<<e<<"   even="<<o;
}
```

Ex2: write program to create linked list of char with 10 node then write function to add new node, function to delete node, and function to print the list.

```
#include<iostream.h>

struct node

    { int x;

    node *nex;};

  void add(node *f)

    {   node *p,*q;   p=f;

        int c;    cout<<"choose element\n";  cin>>c  ;

        while (p->x!=c)     p=p->nex;

         q=new node;     cout<<"read node value\n";

         cin>>q->x;      q->nex=p->nex;     p->nex=q;
```

```
   }
 void print(node *f)
{    node *p;     p=f;
          while( p!=0)
          { cout<<p->x;   p=p->nex;}
}
void delet(node *f)
{    node *p,*q;     p=f;
     int c;   cout<<"chose element\n";     cin>>c;
          while( p->x!=c)
          {q=p; p=p->nex; }
          q->nex=p->nex;
         }
 void main()
{       node *f,*p,*q;
        cout<<"\n enter your list\n";
f=new node;
cin>>f->x;    f->nex=0;    p=f;
for(int i=1;i<5;i++)
{  q=new node;
   cin>>q->x;    q->nex=0;
   p->nex=q;     p=q;}
   p=f;
    int y=0, c;
do
   {     cout<<”\n1= add\n”;
          cout<<”2=print\n”;
```

```
        cout<<"3=delete\n";
        int ch;        cin>>ch;
        switch(ch)
        {case 1:add(f);  break;
         case 2: print(f);break;
         case 3:delet(f);
        }
} while(y!=3);
 }
```