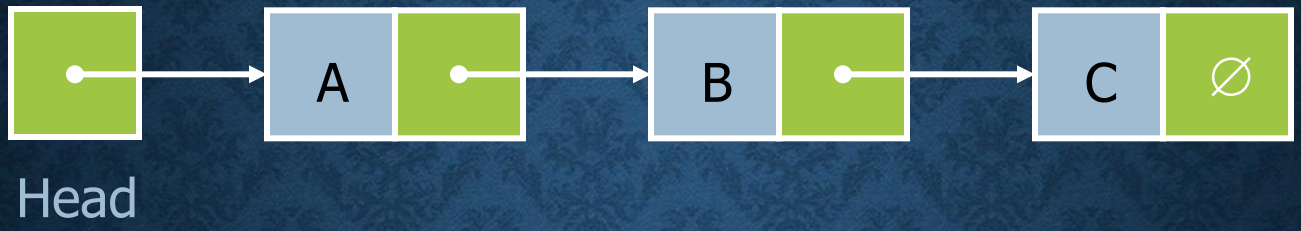
LINKED LISTS

EMAN T.MAHDI
COLLEGE OF C.S. &I.T.

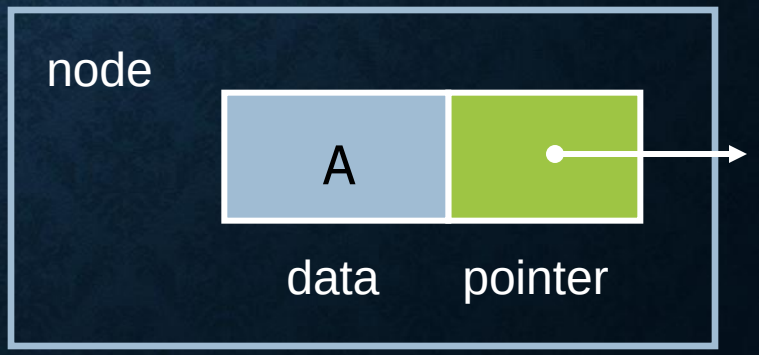
LIST OVERVIEW

- Linked lists
  - Abstract data type (ADT)
- Basic operations of linked lists
  - Insert, find, delete, print, etc.

LINKED LISTS



- A *linked list* is a series of connected *nodes*
- Each node contains at least
  - A piece of data (any type)
  - Pointer to the next node in the list
- *Head*: pointer to the first node
- The last node points to `NULL`



A SIMPLE LINKED LIST CLASS

- We use two classes: **`Node`** and **`List`**
- Declare `Node` class for the nodes
  - `data`: `double`-type data in this example
  - `next`: a pointer to the next node in the list

```
Struct node
{
   double   data;     // data
   Node*    next;     // pointer to next
};
```

A SIMPLE LINKED LIST CLASS

- Declare `List`, which contains
  - `head`: a pointer to the first node in the list.
    Since the list is empty initially, `head` is set to `NULL`
  - Operations on `List`

```
Struct node
{
double data ;
Node*next;
} node *f,*p,*q;
```

A SIMPLE LINKED LIST

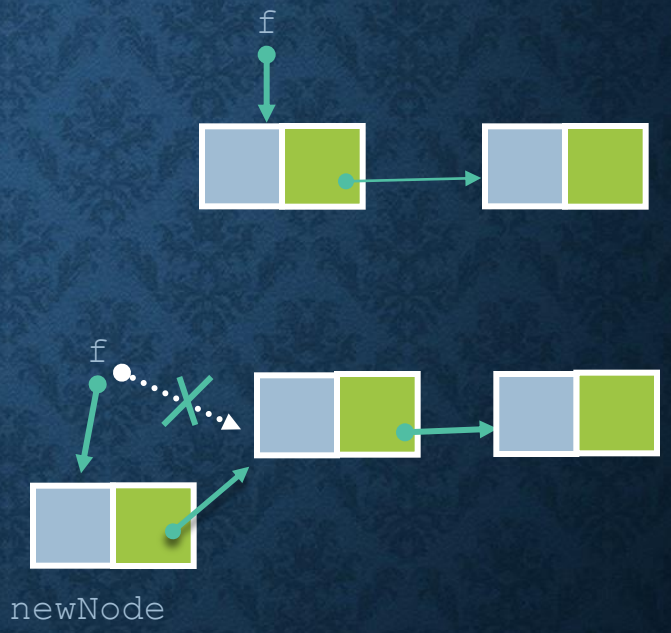
- Operations of `List`
  - Creat :creat new node a particular position
  - Add first creat a new node at the head of list
  - InsertNode: insert a new node at a particular position
  - Delete first node of list
  - Delete last node of list
  - Delete node from specific position
  - Sort list
  - Compute no. even number /prime number
  - Print all node in list

CREATE A NEW NODE

```
Void creat (int i)
{  if (f==0)
        { f=new node;
          f->info=i;
          f->next=0;
           p=f; }
    else
         { q=new node;
           q->info=i;
           q->next=0;
            p->next=q;
            p=q; }
 }
```

ADD NODE AT FIRST OF LIST

```
Void addfirst (int i)
   {
       p=new node;
       p->info=i;
        p->next=f;
         f=p;
    }
```



ADD NODE AT SPECIFIC POSITION

```
Void add_as_want (int n, int y)
   {   p=f;
       node*z;
       while(p->info !=n)
          { p=p->next;}
       if(p->info==n)
        { z=new node;
          z->info=y;
         z->next= 0;
          z->next= p->next;;
         p->next=z;
             }
       else { cout<<"location is not found";}
      }
```

DELETE FIRST NODE OF LIST

```
Void del_first()
{ if( f ->next == 0)
     delete(f);
    else
      { node*p;
        p = f;
        f = f->next;
        p = 0;
        delete(p);
       }
   }
```

DELETE NODE FROM SPECIFIC POSITION

```
Void del_as_info (int n)
{  p=f;
      while(p->info !=n)
          { q = p;
            p= p->next; }
       if(p->info ==n)
           { q->next = p->next;
              p->next=0;
             delete(p); }
       else
       { cout<<“the location is not found”;}
    }
```

PRINTING ALL THE ELEMENTS

```
Void print ()
{ int count = 0;
   p = f;
  while(p != 0)
  { count++;
    cout<<p->x;
    p = p->next; }
  cout<<“number of nodes in list is :”<<count;
  }
```

Ex: split one list into two linked list according to specific information

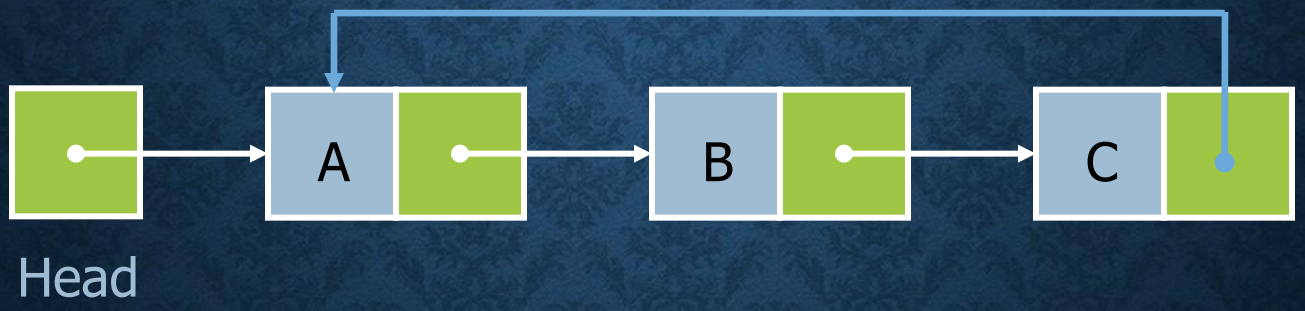
```
#include<iostream.h>
 struct node
{   int item;
     node *next;  };
 node *p,*q,*f1,*f2;
     void split ()
            {int c;     p=f1;
              cout<<"enter the item to split\n";
               cin>>c;
                  while(p->item!=c)
        p=p->next;
              If (p->item==c)
    { f2=p->next;
      p->next=0;}
             else cout<<"item not found";
              }
          void print()
 { p=f;
    while(p!=0)
  { cout<<p->item; p=p->next;} }
```

```
void main()
 {f1=new node;
   cin>>f1->item;
    f1->next=0;

      p=f1;
          for(int i=1; i<5;i++)
         {q=new node;
          cin>>q->item;
          q->next=0;
          p->next =q;
           p=q; }
      p=f1;
     split();
     print();
         }
```

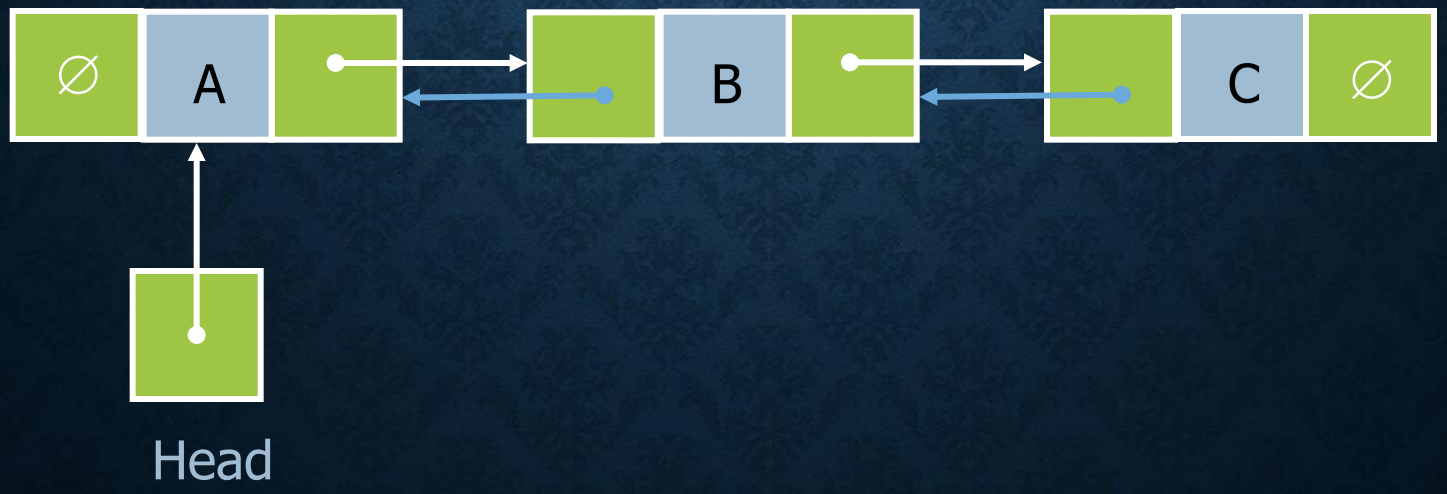
VARIATIONS OF LINKED LISTS

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



VARIATIONS OF LINKED LISTS

- *Doubly linked lists*
  - Each node points to not only successor but the predecessor
  - There are two `NULL`: at the first and last nodes in the list
  - Advantage: given a node, it is easy to visit its predecessor. Convenient to traverse lists backwards



ARRAY VERSUS LINKED LISTS

- Linked lists are more complex to code and manage than arrays, but they have some distinct advantages.
  - **Dynamic**: a linked list can easily grow and shrink in size.
    - We don't need to know how many nodes will be in the list. They are created in memory as needed.
    - In contrast, the size of a C++ array is fixed at compilation time.
  - **Easy and fast insertions and deletions**
    - To insert or delete an element in an array, we need to copy to temporary variables to make room for new elements or close the gap caused by deleted elements.
    - With a linked list, no need to move other nodes. Only need to reset some pointers.

REFERENCES

- : Introduction to Algorithms, 3rd Edition by *Thomas H.* Cormen ,*Charles E. Leiserson, Ronald L. Rivest, Clifford Stein*
- Introduction to Algorithms, 3rd Edition by *Thomas H.* Cormen ,*Charles E. Leiserson, Ronald L. Rivest, Clifford Stein*
- Elements of Programming Interviews in Java: The Insiders' Guide, by *Adnan Aziz, Tsung-Hsien Lee, Amit Prakash*
- https://github.com/careermonk/DataStructuresAndAlgorithmsMadeEasy