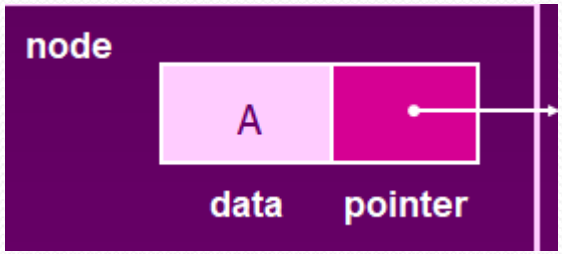
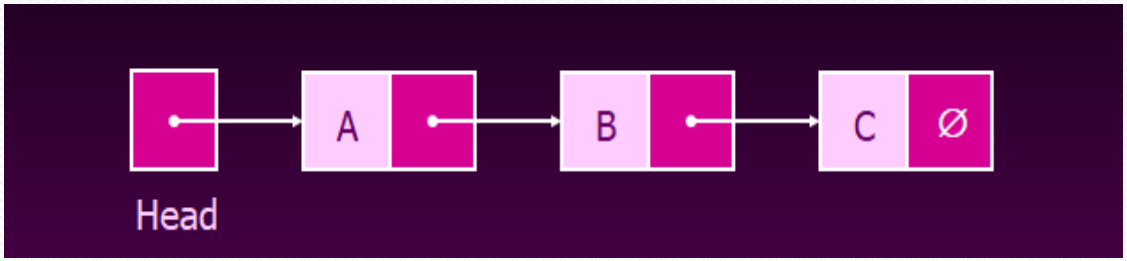
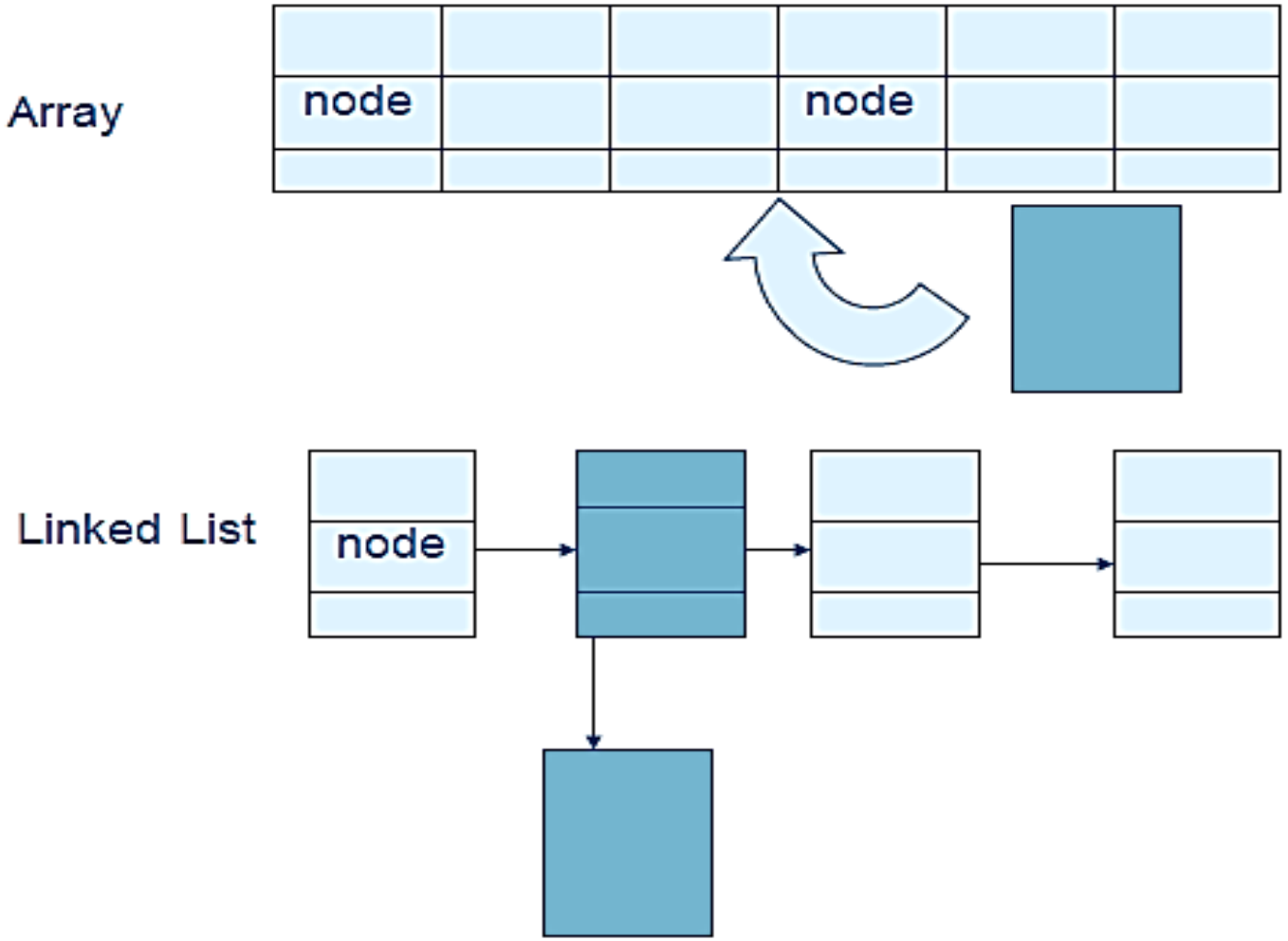
Linked Lists

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





Array vs Linked List



Types of lists

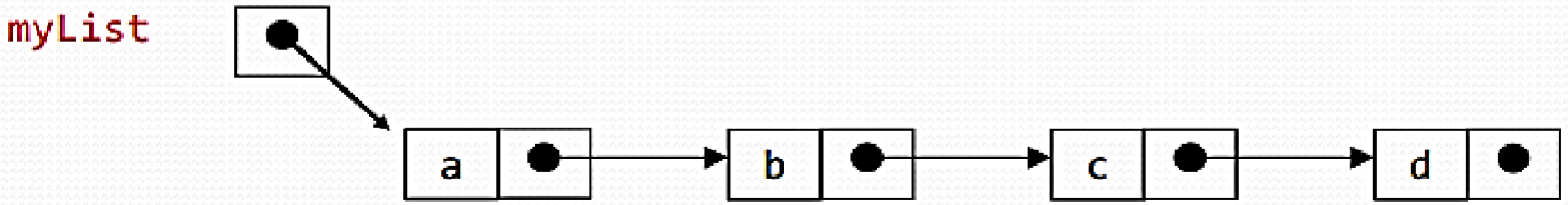
- There are two basic types of linked list
- Singly Linked list
- Doubly linked list

Singly Linked List

- Each node has only one link part
- Each link part contains the address of the next node in the list
- Link part of the last node contains NULL value which signifies the end of the node

Schematic representation

- Here is a singly-linked list (SLL):



- Each node contains a value(data) and a pointer to the next node in the list

- myList is the header pointer which points at the first node in the list

Linked Lists Declarations

- First you must declare a data structure that will be used for the nodes. For example, the following struct could be used to create a list where each node holds a float:

```
struct ListNode
{
        float value;
        ListNode *next;
};
```

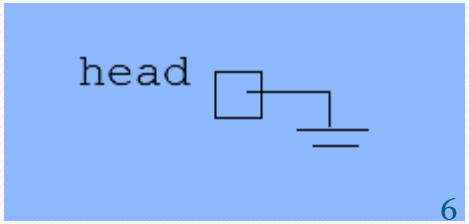
- The next step is to declare a pointer to serve as the list head, as shown below.

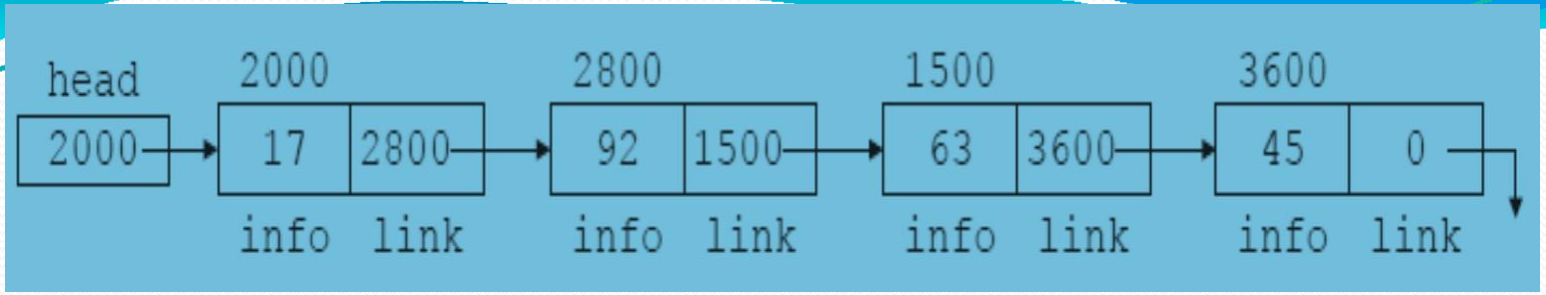
```
ListNode *head;
```

- Once you have declared a node data structure and have created a NULL head pointer, you have an empty linked list.
- The next step is to implement operations with the list.

- Empty Linked list is a single pointer having the value of NULL.

head = NULL;





Linked list with four nodes

Values of head and some of the nodes of the linked list in previous Figure

	Value	Explanation
head	2000	
head->info	17	Because head is 2000 and the info of the node at location 2000 is 17
head->link	2800	
head->link->info	92	Because head->link is 2800 and the info of the node at location 2800 is 92

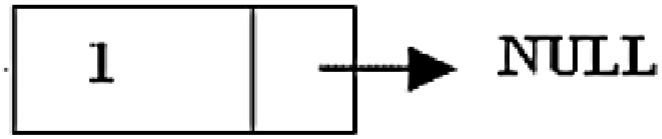
Basic Operations on a list

- Creating a List
- Inserting an element in a list
- Deleting an element from a list
- Searching a list
- Reversing a list

Creating a node

```
struct node{
        int data;              // A simple node of a linked list
        node*next;
    }*start;                   //start  points at the first node
start=NULL ;                      initialised to NULL at beginning
```

```
node* create( int  num)  //say num=1 is passed from main
{
   node*ptr;
   ptr= new  node;  //memory allocated dynamically
        if(ptr==NULL)
          'OVERFLOW' // no memory available
           exit(1);
        else
        {
           ptr->data=num;
           ptr->next=NULL;
           return ptr;
        }
}
```



To be called from main() as:-

```
void main()
{
    node* ptr;
    int data;
    cin>>data;
    ptr=create(data);
}
```

Inserting the node in a SLL

There are 3 cases here:-

- Insertion at the beginning
- Insertion at the end
- Insertion after a particular node

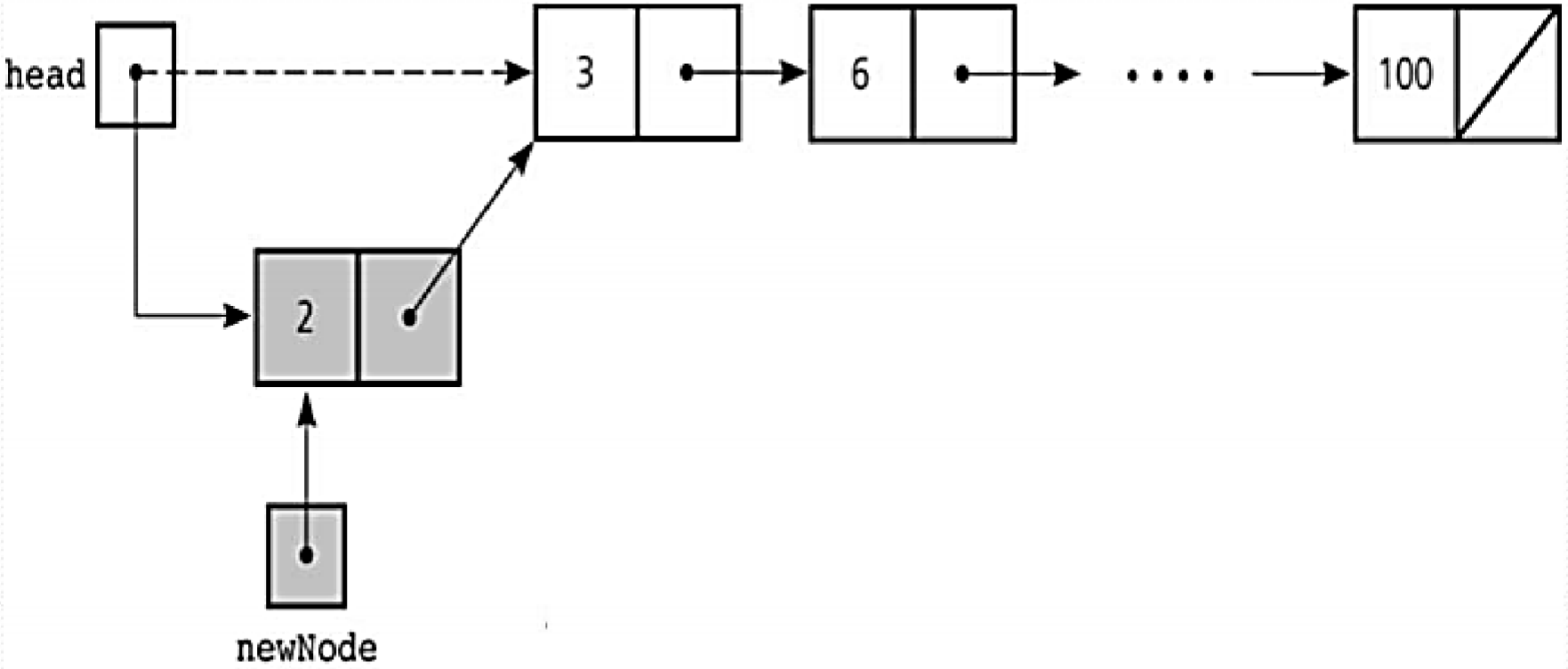
Insertion at the beginning

There are two steps to be followed:-

a) Make the next pointer of the node point towards the first node of the list

b) Make the start pointer point towards this new node

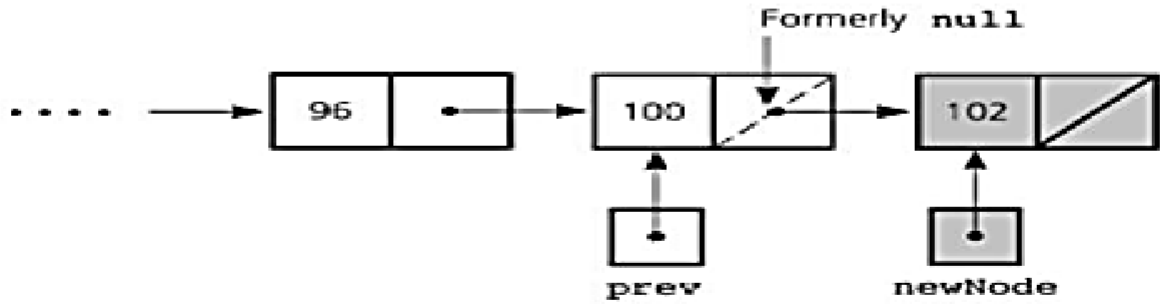
- If the list is empty simply make the start pointer point towards the new node;


head
3
6
100
2
newNode

```
void insert_beg(node* p)
{
node* temp;
            if(start==NULL)    //if the list is empty
           {
              start=p;
              cout<<"\nNode inserted successfully at the
                          beginning";
           }
          else {
                  temp=start;
                  start=p;
                  p->next=temp;  //making new node point  at
                }                          the first node of the list
}
```

Inserting at the end

Here we simply need to make the next pointer of the last node point to the new node



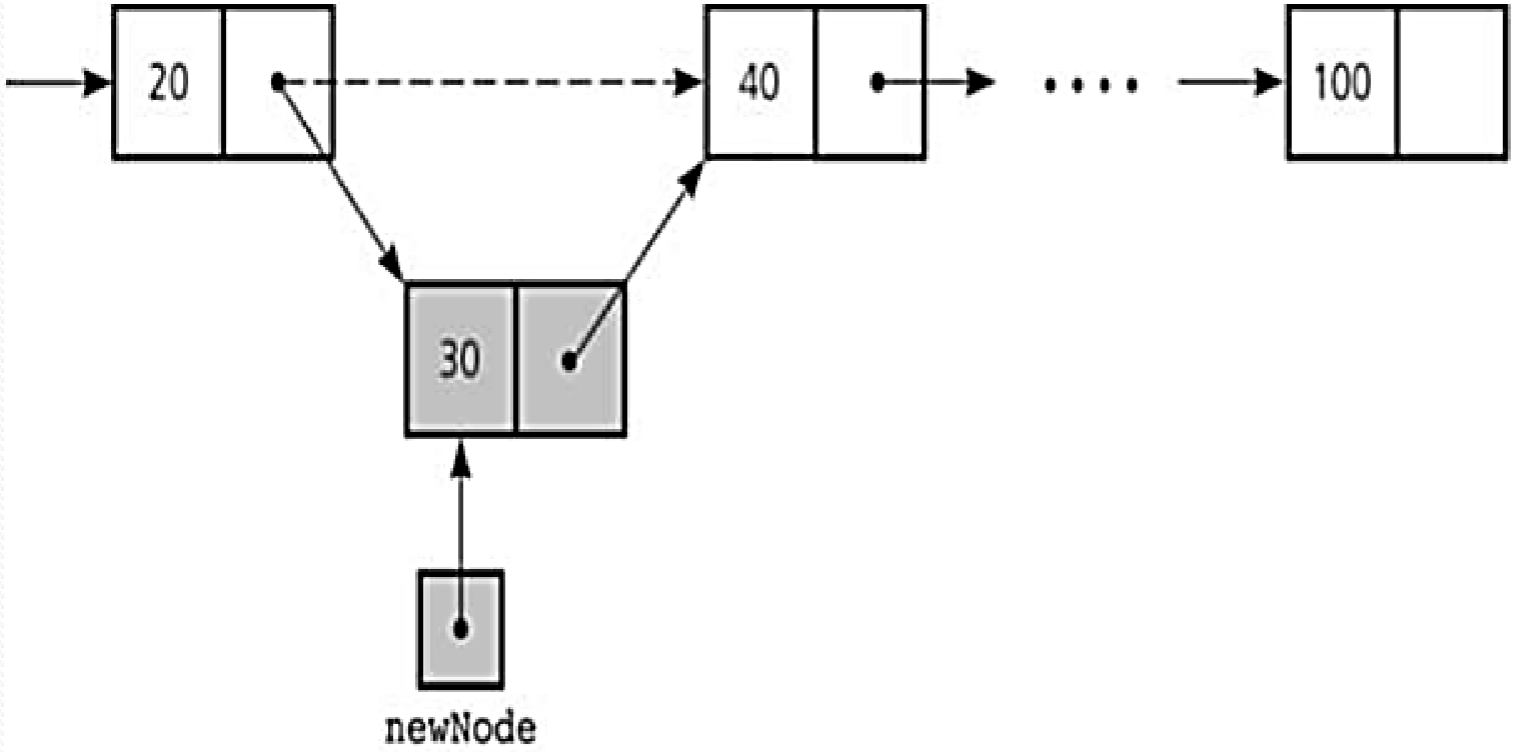
```
void insert_end(node* p)
{
node *q=start;
            if(start==NULL)
            {
              start=p;
              cout<<"\nNode inserted successfully at the end...!!!\n";
            }
          else{
                        while(q->link!=NULL)
                          q=q->link;
                 q->next=p;
                }
}
```

Inserting after an element

Here we again need to do 2 steps :-

- Make the next pointer of the node to be inserted point to the next node of the node after which you want to insert the node

- Make the next pointer of the node after which the node is to be inserted, point to the node to be inserted


20
40
100
30
newNode

```
void insert_after(int c,node* p)
{
node* q;
q=start;
      for(int i=1;i<c;i++)
      {
        q=q->link;
              if(q==NULL)
             cout<<"Less than "<<c<<" nodes in the list...!!!";
      }
 p->link=q->link;
 q->link=p;
cout<<"\nNode inserted successfully";
}
```