Unit 3

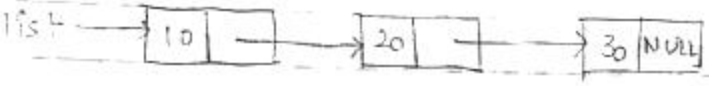
* Linked list

→ A dynamic list in which each node contained a pointer that links it to the next node is called linked list.

→ The node of a linked list can be store anywhere in the memory

→ We access the linked list by saving the address of first node in the pointer variable say list. This pointer called the External pointer variable to the list.

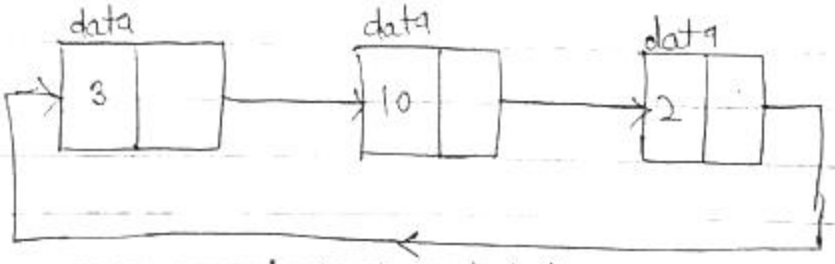
→ The link field to the last node contains NULL to indicate the end of the list.



⇒ Struct node
{
int data;
struct node *link;
};

* Circular linked list

A linked list in which the last node contains the address of first node is known as Circular linked list.

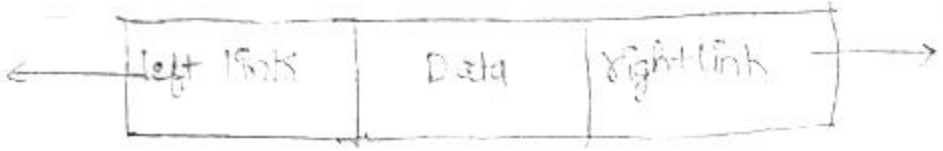


Last element points back to first.

* Doubly linked list.

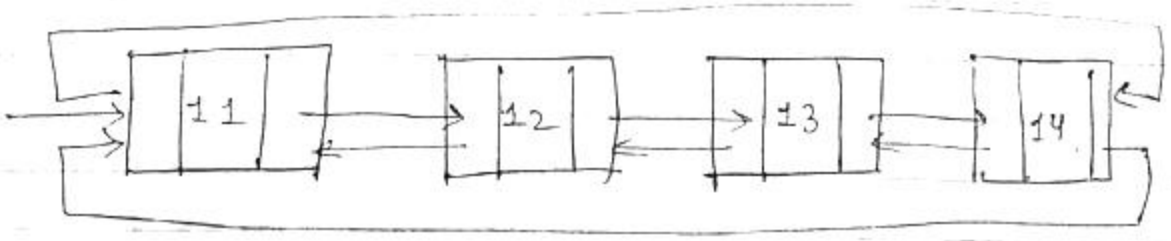
The linked list that may traverse in either direction or in both direction. A doubly linked list has two pointer — one to point to the previous node and another for next node.

- left link – link to predecessor on left node
- data field
- right link - link to successor node on right node.



⇒ Struct dnode
{
Struct dnode *llink;
Int data;
Struct dnode *rlink;
};

* Circular Doubly linked list.
In Circular Doubly linked list, Instead of storing a NULL, we will store the address of first node and last node in the right link and left link of last and first node respectively.



⇒ Struct cdnode
{
Structdnode *llink;
Int data;
Struct dnode *rlink;
};

* Applications for linked list:

→ linked list can be used to implement stacks, queues.
→ linked list can also be used to implement graph.
→ linked list is useful for dynamic memory allocation..
→ Real life examples are persons standing for food in mess.
→ Songs playing in media player.
→ The circular linked list is used in our computer os, where multiple applications are running.

* pointer

A pointer is a variable which contains the adress in memory of another variable. We can have pointer of any type. The Unary operator "&" gives the "address of variable". The Indirection or dereference operator "*" gives the content of an object pointed to by a pointer.

* Difference Between Call by Value and Call by reference.

Call by Value	Call by Reference
1. The actual arguments can be variable or constant.	1. The actual arguments can only be variable.
2. The value of actual argument are sent to formal argument which are normal variable.	2. The ~~value~~ Reference of actual argument are sent to formal argument which are pointer variable.

3. Any changes made by formal arguments will not reflect actual arguments.

```
4. #Include <stdio.h>
   #Include <conio.h>
    main()
    {
    Int X=15, y=20;
   printf("\n Two numbers before
  change function");
 printf(" X=%d y=%d", x, y);
   change (x, y)
 printf("\n Two numbers after
  change function");
 printf(" X=%d y=%d", x, y);
  }
  change (Int X, Int y)
  {
   X++;
   y++;
 printf("\n Two numbers after
 Incrementing In change function");
 Printf("%=%d y=%d", x, y);
  }
```

Output: Two numbers before change function X=15 y=20
Two numbers after Incrementing In change function X=16 y=21
Two numbers after change function X=15 y=20

3. Any changes made by formal arguments will reflect to actual arguments

```
4. #Include <stdio.h>
   #Include <conio.h>
    main()
    {
    Int X=15, y=20;
printf("\n Two numbers before change
  function");
 printf("X=%d y=%d", x, y);
  change (&x, &y)
printf("\n Two numbers after change
  function");
printf(" X=%d y=%d", x, y);
 }
Change (Int *x, Int *y)
  {
*x++;
*y++;
printf("\n Two numbers after Incrementing
In change function");
 printf("X=%d y=%d", *x, *y);
  }
```

Output: Two numbers before change function X=15 y=20
Two numbers after Incrementing In change function X=16 y=21
Two numbers after change function X=16 y=21.

* Difference Between Static memory allocation and Dynamic memory allocation.

	Static memory allocation	Dynamic memory allocation
1.	Performed at static or compile time.	1. performed at dynamic or run time.
2.	Assigned to Stack.	2. Assigned to Heap.
3.	Size must be known at compile time.	3. Size may be Unknown at Compile time.
4.	First In first Out	4. No particular order of assignment.
5.	It is best if requride size of memory known in advance	5. It is best if we don't know about how much memory requires.

* malloc() and Calloc() function.

C provide two built in functions malloc() and Calloc() to Created memory space during runtime.

1. malloc() ⇒ Malloc() function allocates space in bytes.
   Syntax → * malloc (size);

2. Calloc() ⇒ Calloc() function allocates spaces for "n" number of items, where an item is of "size" bytes and store 0 in reserved area.
   Syntax → * Calloc (n, size);