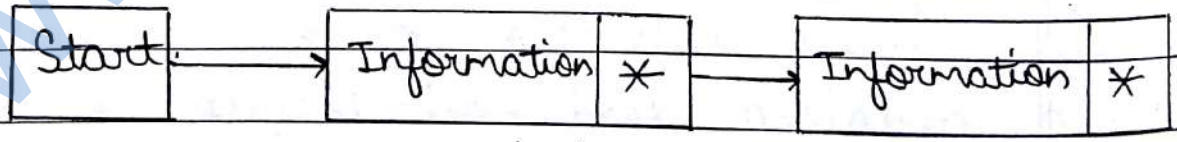
UNIT – 3

Link – List

* **Link – List** – If the memory is allocated before the execution of program, it is fixed and cannot be changed. We have to adopt an alternative strategy to allocate memory when it is required.
There is a special data-structure called link-list that provides a more flexible storage system and it does not require the use of array.
"Link-lists are special lists of some data elements link to one another. The logical ordering is represented by having each element pointing to the next element.



Nodes

In singly link-list nodes have one pointer (next) pointing to the next node

Advantages of Link-list – Link-list have many advantages –

1- Link-list are dynamic data structure, that is they can grow and shrink during the execution of program.

2- Effecient memory utiliration. Here memory is not preallocated. Memory is allocated whenever it is require and it is de-allocated (removed) when it is no longer needed.

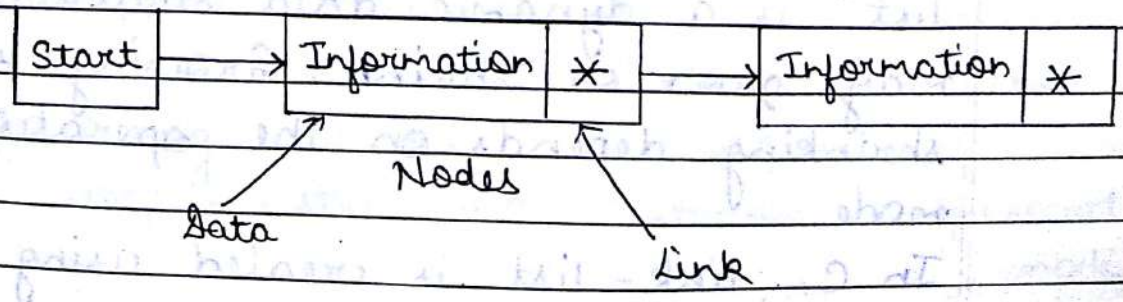
3- Insertion and deletion are easier and efficient. Link-lists provide flexibility in inserting a data item at a specified position and deletion of data-item from the given position.

Disadvantages of Link-list —

1- More memory, if number of feeds are more then more space is required.

2- Access to an orbitatory data-item is little-bit difficult and time consuming.

• **Nodes** :- A link-list is a non-sequential collection of data-items called nodes. Each node in a link-list have basically two fields —



1) The data fields contains an actual value to be stored and process.
2) The link field contains the address of the next data item in the link-list. The address used to access a particular node is known as pointer.
3) The logical and physical ordering of data-items in a link-list need to be same. But in sequential representation these ordering are the same.

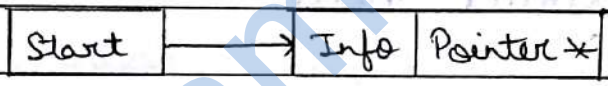
Initialization of link-list –

```
struct node
{
int info;
struct node * link;
};
struct node * tmp;
tmp = (struct node *) malloc (sizeof (struct node));
```

* Types of link-list :-

1- Singly link-list – A singly link-list is a dynamic data structure. It may grow or shrink. Growing or shrinking depends on the operations mode.
In C, link-list is created using structures

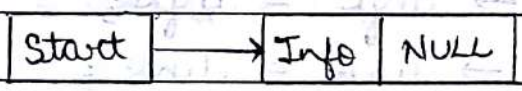
pointers and dynamic memory allocation function malloc (memory allocation). We consider head as an external pointer. This help in creating and accessing other nodes in the link-list.
When the statement tmp = (struct node *) malloc (sizeof (struct node)); is executed, a block of memory sufficient to store the node is allocated and assigns head as the starting address of the node (now head is an external pointer). This activity can be pictorially represented as-



Node

Now we can assign values to the respective field of node.

tmp = info = 10;
tmp → 10;
tmp link → NULL



Any number of nodes can be created and linked to the existing nodes.
Suppose we want to add another node to the above list, when the following statement is required -

tmp = (struct node *) malloc (sizeof (struct node));

Date / /

start → tmp · info = 100;
start → tmplink · tmp link = NULL.

* Insertion in singly link-list :-

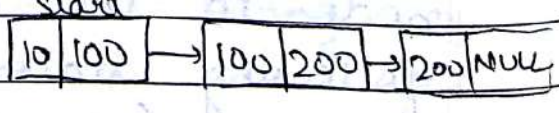
1. Insertion in an empty link-list.
2. Insertion at the beginning.
3. Insertion at the end.
4. Insertion at the n^{th} position.

1. Insertion in empty link-list —

```
struct node
{
int info;
struct node * link;
}
struct node * tmp;
tmp = (struct node*) malloc(sizeof(struct node));
tmp → info = info;
tmp → link = link;
tmp → link = NULL;
start = tmp;
```

2- Insertion at the beginning —

```
struct node
{
int info;
struct node * link;
```

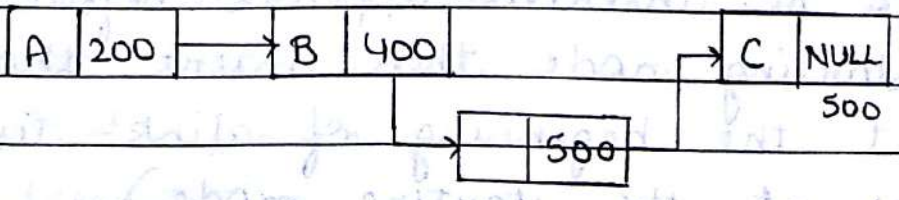


```
}
struct node * tmp;
tmp = (struct node *) malloc (sizeof (struct node));
tmp → info = 10;
tmp → link = start;
start = tmp;
```

3. Insertion at the end —

```
struct node
{
int info;
struct node * link;
}
struct node * tmp;
tmp = (struct node *) malloc (sizeof (struct
                                    node));
struct node * p;
p = start;
while (p != NULL)
p = p → link;
p → link = tmp;
tmp → link = NULL;
```

4. Insertion at n^{th} position —




```
struct node
{
```

```
int info;
struct node * link;
}
struct node * tmp;
tmp = (struct node *) malloc(sizeof(struct node));
struct node * p
p = start;
for (i = 1; i < position - 1 && p != NULL; i++)
p = p → link;
tmp → link = p → link;
p → link = tmp;
```

Inserting a new node into the link-list has the following four instances-

1. Insertion in an empty link-list.
2. Insertion at the beginning.
3. Insertion at the end.
4. Insertion at n^{th} position.

The following steps involved in deciding the position of insertion are as follows-

1.) If a link-list is empty or the node to be inserted appears before the starting node then insert that node at the beginning of link-list. (that is at the starting node).

2.) If the node to be inserted appears

after the last node in the link-list then insert that node at the end of the link-list.

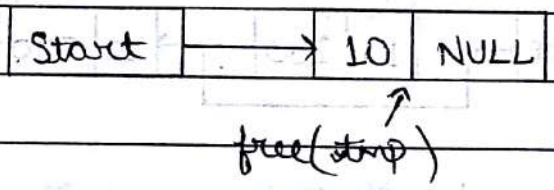
3.) If the above two conditions do not hold true then insert the new node at the specified position within the list.

* **Deletion in link-list** :- Deleting a node from link-list has the following three instances -

1- Deleting the first node of the link-list.
2- Deleting the last node of the link-list.
3- Deleting the specified node within the link-list.

~~Deleting the last node of the link-list.~~
~~Deleting the specified node within the link-list.~~

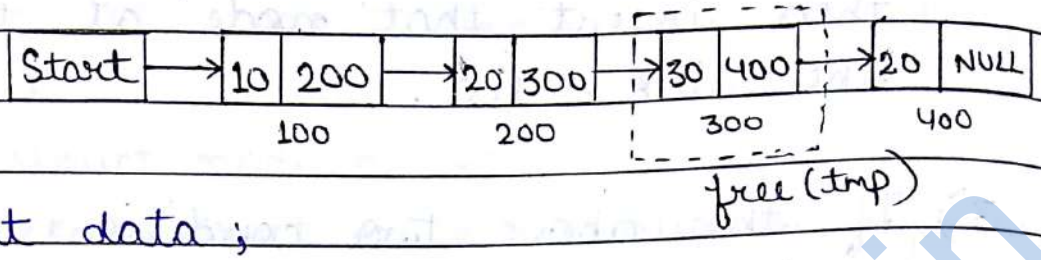
1- Deletion of one node -




```
struct node *tmp
tmp = start;
start = null;
free (tmp);
```

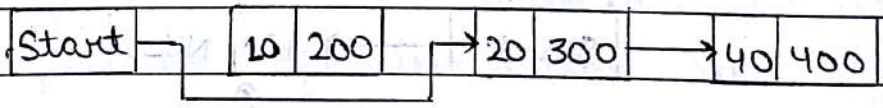
Date / /

2- Deletion of n^{th} node -




```
int data;
struct node *p;
struct node *tmp;
 p = start;
while (p→ link != NULL)
 {
if (p→ link → info == data)
 {
tmp = p→ link;
p→ link = tmp→ link;
 free (tmp);
 }
 p = p→ link;
 }
```

3- Deletion at the beginning -




```
struct node *tmp;
 tmp = start;
start = start → link;
free (tmp);
```

Date / /

* Program for matrix multiplication :-

```
#include <stdio.h>
#include <conio.h>
void main()
{
clrscr();
int a[3][3], b[3][3], c[3][3], i, j, k;
printf("Enter the data in matrix A");
for (i=0; i<3; i++)
{
for (j=0 ; j<3; j++)
{
scanf ("%d", &a[i][j]);
}
}
printf("Enter the data in matrix B");
for( i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
scanf ("%d", &b[i][j]);
}
}
printf("Multiplication of matrices A and B is:
\n");
for (i=0; i<3 ; i++)
{
for(j=0; j<3; j++)
{
c[i][j] =0;
```

Date / /

```
for(k=0 ; k<3; k++)
{
c[i][j] = a[i][j] + a[i][k] + b[k][j];
}
}
printf ("%d\n");
for(i=0; i<3; i++)
{
for(j=0; j<3; j++)
{
printf ("%d\t", c[i][j]);
}}
getch();
}
```