LINKED LIST.

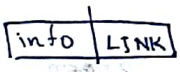
is defined as collection of sequential data items. These sequential data items are termed as "nodes"

(or) LL is a collection of nodes.

linked list uses dynamic memory allocation.

A node in a LL has 2 fields

1. info field - where info is stored.
2. link field - holds the address of next.



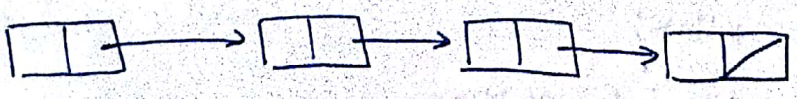
A node in a LL can be represented with the following struct def".

```
struct node
{
   int info;
   struct node * link;   → self referential structure.
};
```

There are 4 types of LL, namely.

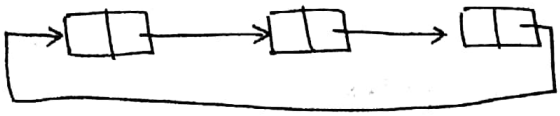
1) Singly LL

here, each node apart from info field, the link field contains only one address ie, address of the next node, and the last node link field contains null.



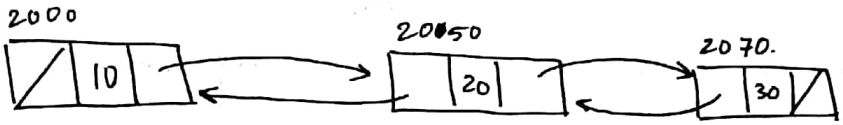
2) Circular singly LL

singly LL wherein the last node contains the address of the first node

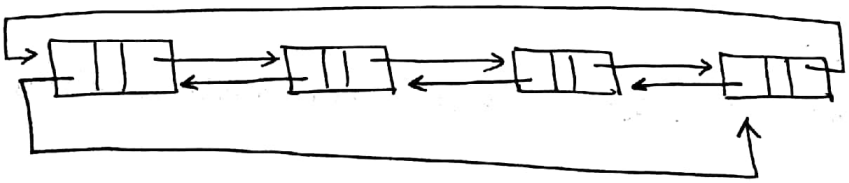


3) Doubly LL

here, each node contains the info field and the the link field contains 2 address. ie address of next node and previous node.



4) Circular double linked list



Single Linked List

To implement a single LL, we'll consider following struct defn to represent a node.

```
struct node
{
    int info;
    struct node *link;
}; typedef struct node * NODE;
```

To insert a new node at the front end.

```
NODE insertfront (NODE first , int item)
{
   NODE temp ;
   temp = (NODE) malloc (sizeof (struct node));
   temp → info = item;
   temp → link = first;
   return temp;
}
```

first is a variable of the ~~at~~ data type • NODE = (struct node *)

To insert a node at rear end.

```
NODE insertrear (NODE first ~~rear~~, int item)
{
   NODE temp; cur ,
   temp = (NODE) malloc (sizeof (struct node));
   temp → info = item;
   temp → link = NULL;
   if (first = = NULL)
        return temp;
   ~~while (cur → link ! = NULL)~~
   cur = first;
   while ( cur → link ! = NULL)
            cur = cur → link;
   cur → link = temp;
   return first;
}
```

To delete a node at front end.

```
NODE deletefront (NODE first)
{
    NODE temp;
    temp = first;
    if ( first == NULL)
    {
        pf("\n empty");
        return NULL;
    }
    first = first -> link;
    pf ("\n element del is %d ", temp -> info);
    free (temp);
    return first;
}
```

To delete a node at rear end.

```
NODE deleterear (NODE first)
{
    NODE ~~temp~~ cur, prev
    if (first == NULL)
    {
        pf("\n empty");
        return NULL;
    }
    cur = first
    prev = NULL
    while (cur -> link != NULL)
    {   prev = cur;
        cur = cur -> link;
    }
    prev -> link = NULL;
```

```
        pf(" element deleted is %d " cur->info);
        free (cur)
        return first;
}
```

To display content of single LL

```
void display (NODE first)
{
    if (first == NULL)
        pf("list empty");
    else
    {
        pf(" list content is");
        temp = first;
        while (temp != NULL)
        {
            pf("%d", temp->info);
            temp = temp->link;
        }
    }
}
```

A stack can be implemented by following funcs

1. insert front - push.
2. delete front - pop.
3. display

A queue can be implemented by the following funcs

1. insert rear - enqueue
2. delete front - dequeue.
3. display.

Lab prog 4.

to multiply 2 polynomials

ex:- $4x^3 + 2x^2 + 8x + 1$ (4 term).

4 * (x)^3 + 2 * (x)^2 + 8 * (x)^1 + 1 * (x)^0.

```
#include <stdio.h>
#include <stdlib.h>

struct node
{
   int co, po;
   struct node *link;
};
typedef struct node *NODE;

NODE insert (NODE first, int co, int po)
{
   NODE temp, cur;
   temp = (NODE) malloc (sizeof (struct node));
   temp -> co = co;
   temp -> po = po;
   temp -> link = NULL;
   if (first == NULL)
         return temp;
   cur = first;
   while (cur -> link != NULL)
         cur = cur -> link;
```

```
    cur->link = temp;
    return first;
}

void display (NODE first)
{
    NODE temp;
    if (first == NULL)
        pf("\n polynomial is empty");
    else
    {
        temp = first
        while (temp->link != NULL)
        {
            pf(" %d * (x)^ %d +", temp->co, temp->po);
            temp = temp->link;
        }
        pf ("%d * (x)^ %d \n", temp->co, temp->po);
    }
}

NODE polymultiply (NODE poly1, NODE poly2)
{
    NODE first, second;
    NODE res = NULL;
    for (first = poly1; first != NULL ; first = first->link)
        for (second = poly2; second != NULL; second = second->link).
            res = addterm (res, first->co * second->co, first->po * second->po)
    return res;
}
```

```
NODE addterm (NODE res ; int co, int po)
{
    NODE temp, cur ;    int flag = 0;
    temp = (NODE) malloc (sizeof (struct node));
    temp -> co = co ;
    temp -> po = po ;
    temp -> link = NULL;
    if (res == NULL)
            return temp;
    cur = res
    while (cur != NULL)
    {
        if (cur -> po == po)
        {
            cur -> co = cur -> co + co;
            flag = 1;
            return res;
        }
        cur = cur -> link;
    }
    if (flag == 0)
    {
        cur = res;
        while (cur -> link != NULL)
                cur = cur -> link;
        cur -> link = temp;
        return res;
    }
```

```
int main ()
{
    NODE poly1 = NULL, poly2 = NULL, poly3 = NULL;
    int m, n, co, po, i;
    printf ("\n Read no of terms for first poly:");
    scanf ("%d", &m);
    for (i=1 ; i<=m ; i++)
    {
        pf("\n Read co and po for %d term", i);
        sf ("%d %d", &co, &po);
        poly1 = insert (poly1, co, po);
    }
    printf ("\n Read no of terms for second poly:");
    scanf("%d", &n);
    for (i=1 ; i<=n ; i++)
    {
        pf ("\n Read co and po for %d term", i);
        sf ("%d %d", &co, &po);
        poly2 = insert (poly2, co, po);
    }
    printf ("\n first poly is\n");
    display (poly1);
    printf ("\n second poly is\n");
    display (poly2);
    poly = poly multiply (poly1, poly2);
    pf ("\n result poly is\n");
    display (poly);
    return 0;
}
```

WAP func to delete a node based on info field.

```
NODE deleteinfo (NODE first , int key)
{
    NODE temp; cur, prev
    if (key == first ->info)
    {
        temp = first
        first = first-> link;
        pf ("\n ele del is %d", temp -> info);
        free (temp);
       return first;
    }

  prev = NULL;
  cur = first;
  while (cur != NULL && key != cur -> info)
  {
     prev = cur ;
     cur = cur -> link;
  }
  if (cur == NULL)
  {
    pf ("\n %d has not been there ", key);
    return first ;
  }
  prev -> link = cur -> link ;
  pf ("\n %d is deleted ", cur -> info.
  free (cur);
  return first;
```

C function to delete a node based on the pos.

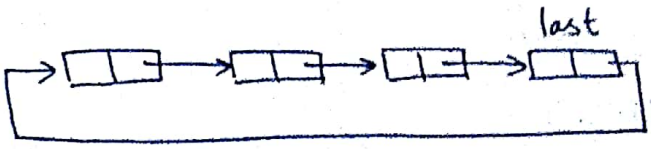
```
NODE deletepos(NODE first, int pos)
{
    NODE temp, cur, prev;
    int ct = 1;
    if (first == NULL)
    {
        pf("Nope");
        return first;
    }
    if (pos == 1)
    {
        temp = first;
        first = first -> link;
        pf("\n %d is deleted", temp -> info);
        free(temp);
        return first;
    }
    cur = first;
    while (cur != NULL && ct != pos)
    {
        prev = cur;
        cur = cur -> link
        ct = ct + 1;
    }
    if (ct != pos)
    {
        pf("\n ___ ");
        return first;
    }
    prev -> link = cur -> link;
    pf("%d is deleted", cur -> info);
    free(cur);
    return first
}
```

WAF to create an ordered list.

```
NODE insertorder(NODE first, int item)
{
   NODE temp, cur, prev;
   temp= (NODE) malloc (sizeof (struct node));
   temp -> info= item
   temp -> link = NULL
   if (first == NULL )
        return temp;
   if (item <= first ->info)
   {
       temp-> link= first ;
        return temp;
    }
    cur = first
   while (cur != NULL  &&  item > cur -> info)
   {   prev = cur;
       cur = cur -> link;
   }
   prev -> link = temp;
   temp -> link= cur;
    return first;
}
```

Circular Single linked list

In case of a circular single linked list, The link field of the last node contains address of the first node.



Let last be the reference of a circular list which holds the address of the last node in the list

If last = null, implies list is empty

If last → link = last, implies only one node

WAF to add a node at front end.

```
NODE insert front (NODE last, int item)
{
  NODE temp;
  temp = (NODE)malloc (sizeof (struct node));
  temp → info = item;
  if (last == NULL)
  {
    temp → link = temp;
    return temp;
  }
  else
  {
    temp → link = last → link
    last → link = temp;
    return last;
  }
}
```

WAF to insert at rear end.

```
NODE insert_rear (NODE last, int item)
{
    NOD temp;
    temp = (NODE) malloc (sizeof(struct node));
    temp -> info = item;
      if (last == NULL)
      {
          temp -> link = temp;
          return temp;
      }
      else
      {
          temp -> link = last -> link;
          last -> link = temp;
          return temp;
      }
}
```

WAP to delete a node from front end.

```
NODE deletefront(NODE last)
{
    NODE * temp;
    if (last == NULL)
    {
        pf(empty)
        return NULL;
    }
    if (last->link == last)
    {
        printf("%d deleted", last->info);
        free(last);
        return NULL;
    }
```

→ allocate memory for temp

```
    temp = last->link;
    last->link = temp->link;
    pf("\n %d is deleted", temp->info);
    free(temp);
    return last;
}
```

WAF to delete rear.

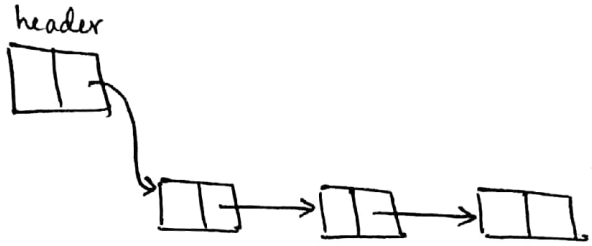
```
NODE deldelrear(NODE last)
{
    NODE cur;
    if (last == NULL)
    {
        pf(empty);
        return NULL;
    }
    if (last -> link == last)
    {
        pf("%d is deleted", last -> info);
        free(last);
        return NULL;
    }
    cur ...
    cur = last -> link;
    while (cur -> link != last)
            cur = cur -> link;
    cur -> link = last -> link;
    pf("%d ...", laust -> info);
    free(last);
    return cur;
}
```

WAF to display

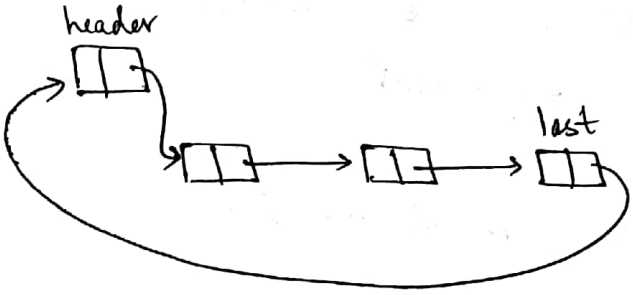
```
void display ( NODE last).
{
    NODE temp
    if (last == NULL)
    {
      pf (empty);
    }
    else
    {
      allocate .. temp.
      temp = last->link;
      while (temp != last)
      {
          print("%d\t", temp->info);
          temp = temp->link;
      }
      pf("%d\t", last->info);
    }
}
```

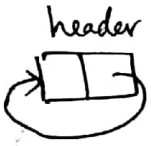
Header node

Tis a special node in case of a LL which generally contains the info about the linked list (no of nodes, objective of LL) and the link field of the header node contains address of first node in the list.



single LL





list is empty.

Circular single LL.

In case of CSLL, the list is empty if header of link points to header

In any list which has an header node before creating the list, memory should be allocated to the header node and the link field should be assigned to itself

A func to insert node at front end. in a circular list with header node.

```
NODE insertfront (NODE head, int item)
{
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    temp → info = item;
    if (head → link == head)
    {
        temp → link = head;
        head → link = temp;
        return head;
    }
    else
    {
        temp → link = head → link;
        head → link = temp;
        return head;
    }
}
```

Add a node at rear end.

```
NODE insertrear (NODE head, int item)
{
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    temp → info = item;
    if (head → link == head)
    {
        temp → link = head;
        head → link = temp;
        return head;
    }
```

```
    cur = head -> link;
    while (cur -> link != head)
        cur = cur -> link;
    ~~cur -> link = temp;~~
    temp -> link = cur -> link;
    ~~temp~~ cur -> link = temp;
    return head;
}
```

Deletion from front end

```
NODE delete front (NODE head) .
{
    NODE temp;
    if (head -> link == head)
    {
        printf ("In list empty");
        return head;
    }
    ~~temp~~ = ~~head->link~~ temp = head -> link;
    head -> link = temp -> link ;   pf (temp -> info);
    free (temp);
    return head;
}
```

Delete at rear end.

```
NODE delete rear (NODE head)
{
    NODE ~~temp~~ cur, prev.
    if (head -> link == head)
    {
        pf (empty)
    return head;
```

```
    cur = head->link.
    while (cur->link != head)
            prev = cur;
            cur = cur->link;
    prev->link = head;
    pf (cur->info);
    free(cur);
    return head;
}

// display function
void display (NODE head)
{
    if (head->link == head)
    {
        ~~head~~ print (empty)
        return;
    }
    temp = head->link.
    while (temp != head)
    {
        pf (temp->info)
        temp = temp->link;
    }
}
```

Lab program 5.

To add 2 long integers using circular list with header node.

---

① Read 2 nos & create list to hold no. (insert rear).

② check size of both, if equal. ie if not, add zeros from front

③ reverse both lists.

```
struct node
{
   int info;
   struct node * link;
};
typedef struct node * NODE;

int main()
{
   NODE h1, h2,
   int i,
   char a[100], b[100];

   h1 = (NODE) malloc (sizeof(struct node));
   h2 = (NODE) malloc (sizeof (struct node));
```

```
    h1->link = h1;
    h2->link = h2;
    pf("Read the first no");
    scanf("%s", a);
    for (i=0; a[i]!='\0'; i++)
    {
        h1 = insert rear (h1, a[i]-'0');
    }
    pf("first no is");
    display (h1);
    pf("read second no");
    scanf("%s", b);
    for (i=0; b[i]!='\0'; i++)
        h2 = insertrear (h2, b[i]-'0');
    pf("second no is");
    display (h2);
    append (h1,h2);
    pf("First no"); display (h1);
    pf("Second no"); display (h2);
    add (h1, h2);
    return 0;
}

NODE insert rear (NODE head, int item)
{
    NODE temp;
    temp = (NODE*) malloc (sizeof (struct node));
    temp -> info = item;
    if (head -> link == head)
        ;
}
```

```
void display (NODE head);
{
    :
}
void append (NODE h1 , NODE h2)
{
    int ct1=1 , ct2=1 , diff
    s1= size (h1);
    s2= size (h2);
    if (s1>s2)
    NODE temp;
    temp = h1→link;
    while(temp → link != h1)
    {
        ct1 = ct1+ 1;
        temp = temp →link;
    }
    temp = h2→ link;
    while (temp → link != h2)
    {
        ct2 = ct2 +1)
        temp = temp →link;
    }
    if (ct1 >ct2)
    {
        diff = ct1 - ct2;
        for (i = 1; i <= diff ; i++)
            h2= insert zero(h2,0);
    }
    else if (ct2 > ct1)
    {
        diff = ct2 - ct1;
        for (i=1; i<= diff ;i++)
            h1= insert zero (h1, 0);
    }
```

```
NODE insterzero (NODE head, int item)
{ ;
                                    // insert front func.
}

void add (NODE h1, NODE h2)
{
   NODE h, temp1, temp2;
   int sum = 0, carry = 0, x;
   h = (NODE) malloc (sizeof (struct node));
   h -> link = h;
   h1 = reverse (h1);
   h2 = reverse (h2);
   temp1 = h2 -> link;
   temp2 = h2 -> link;
   while (temp1 != h1 && temp2 != h2)
   {
      x = temp1 -> info + temp2 -> info + carry;
      sum = x % 10;
      carry = x / 10;
      h = insert (h, sum);
      temp1 = temp1 -> link;
      temp2 = temp2 -> link;
   }
   if (carry > 0)
      h = insert (h, carry);
   h = reverse (h);
   display (h);
}
```

```
NODE reverse(NODE head)
{
    NODE prev, cur, next;
    cur = head → link;
    prev = head.
    while (cur != head)
    {
        next = cur → link;
        cur → link = prev;
        prev = cur;
        cur = next;
    }
    head → link = prev;
    return head;
}
```

Write a C func to allocate memory to a node in LL.

```
NODE getnode()
{
    NODE temp;
    temp = (NODE) malloc (sizeof (struct node));
    return temp;
    if (temp == NULL)
    {
        pf ("\n insufficient memory");
        exit(0);
    }
    return temp;
}
```

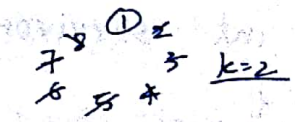
Write a func to deallocate memory for a node.

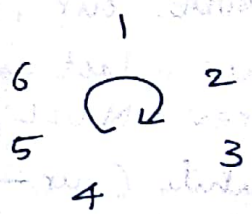
```
void freenode (NODE temp)
{
    free(temp);
}
```

JOSEPHUS PROBLEM

Let us assume that there are n number of people standing in a circle where we need to eliminate every third person until we are left out with one person.

Let the value of n be 6.





order of elimination is 3, 6, 4, 2, 5 ∴ 1 is survive.

In general, if we eliminate every k^{th} person from the given set, then J(n, k) should give us the survivor

If k=2, then J(n,k) can be obtained from the binary expansion of n by a circular left shift of the binary digits of n.

If

mory to a node

node));

Implemat ation of Josephus problem

```
struct node
{
    int info;
    struct node *link;
};
typedef struct node *NODE;

// write insertrear function on a circular list.
// write display function.

int survivor (NODE last, int k)
{
    NODE cur, prev;
    cur = last -> link;
    prev = NULL;
    while (cur -> link != cur)
    {
        for (i=0; i <k-1 ; i++)
        {
            prev = cur;
            cur = cur -> link;
        }
        prev -> link = cur -> link;
        pf (cur -> info);

        free (cur);
        cur = prev -> link.i

    }
    retur current -> info;
}
```

```
int main()
{
    int i, n, k, sur
    NODE last = NULL;
    pf("Read no of people");
    scanf("%d", &n);
    ~~insert~~ ~~last = insertrear(last,~~
    for (i=1; i<=n; i++)
        last = ~~insertrear(last~~, insertrear(last, i);
    display(last);
    pf("Read k");
    scanf("%d", &k);
    ~~last=>~~
    sur = survivor(last, k);
    printf("%d is survivor", sur);
}
```

cular list.

from DMA.

Quiz two

Doubly LL

A node ← info, llink, rlink

```
struct node
{
    int info;
    struct node *llink;
    struct node *rlink;
};  typedef struct node *NODE.
```

```
NODE insertfront (NODE first, int item)
{
  NODE temp;
  temp=(NODE) malloc (sizeof (struct node));
  temp->info = item;
  temp->llink = temp->rlink = NULL;
  if (first == NULL)
    return temp;
  temp->rlink = first;
  first->llink = temp;
  return temp;
}
```

```
NODE insertrear (NODE first, int item)
{
  NODE temp;
  temp = (NODE) malloc (sizeof (struct node));
  temp->info = item;
  temp->llink = temp->rlink = NULL;
  if (first == NULL)
    return temp;
  cur = first;
  while (cur->rlink != NULL)
    cur = cur->rlink;
  temp->llink = cur;
  cur->rlink = temp;
  return first;
}
```

```
NODE delete front(NODE first)
{
    NODE temp;
    if (first == NULL)
    {
        printf(" empty");
        return NULL;
    }
    temp = first;
    temp = temp -> rlink.
    temp -> llink = NULL;
    printf ("first -> info");
    free(first);
    return temp;
}

NODE delete rear (NODE first)
{
    NODE cur, prev;
    if (first == NULL)
    {
        printf("(" empty");
        return NULL;
    }
    if(first -> rlink == NULL)
    {
        pf(" first -> info ");
        free (first);
        return NULL;
    }
    cur = first.
    prev = NULL
    while (cur -> rlink != NULL)
    {
        prev = cur;
        cur = cur -> rlink;
    }
```

```
        pf("deleted is %d", cur->info);
        prev->rlink = NULL;
        free(cur);
        return first;
    }

    void display (NODE first)
    {
        NODE temp;
        if (first == NULL)
        {
            printf("empty");
            return;
        }
        temp = first;
        while (temp != NULL)
        {
            printf("%d\t", temp->info);
            temp = temp->rlink;
        }
    }
```

i) Function to delete a node based on info field:

ii) ~~based~~ delete based on position

iii) insert based on position.

iv) ordered doubly LL.

v) to reverse doubly LL.

vi) to find max and min.

Lab program 6

To represent a sparse matrix using doubly linked list.

Spare matrix is a matrix where 90-95% of the matrix contents are zeroes.

```
#include <stdlib.h>
#include <stdio.h>

struct node
{
    int row, col, info;
    struct node *rlink, *llink;
}; typedef struct node *NODE;

#
NODE insert (NODE first, int r, int c, int item)
{
    NODE temp, cur;
    temp = (NODE) malloc (sizeof(struct node));
    temp->info = item;
    temp->row = r;
    temp->col = c;
    temp->llink = temp->rlink = NULL;
    if (first == NULL)
        return temp;
    cur = first;
    while (cur->rlink != NULL)
        cur = cur->rlink;
```

```
        cur->rlink = temp;
        temp->llink = cur;
        return first;
}

void display (NODE first)
{
    NODE temp;
    if (first == NULL)
    {
        printf ("empty");
        return;
    }
    temp= first
    pf("\n row\t col \t value\n");
    while(temp != NULL)
    {
        pf("%d \t %d\t %d \n", temp-> row, temp->col , temp->info
        temp = temp -> rlink;
    }
}
void display matrix (NODE first , int m , int n)
{
    int i,j
    NODE temp = first
    for (i= 1 ; i<=m ; i++)
    {
        for (j=1; j<=n ; j++)
        {
            if (temp != NULL && temp->row==i &&
                    temp -> col==j)
            {
                printf ("%d\t", temp->info);
```

```
            else
                printf("0\t");
        }
        printf("\n");
    }
```

```
int main()
{
    int m,n,i,j, item;
    NODE first=NULL;
    pf("\n read order of matrix");
    sf("%d %d",&m, &n);
    pf("\n read the elements\n");
    for(i=1; i<=m; i++)
    {
        for (j=1; j<=n; j++)
        {
```

```
                sf("%d", &item);
                if (item != 0)
                    first = insert (first, i, j, item);
            }
        }
    display (first); pf("matrix is");
    display matrix (first, m, n);
    return 0;
}
```