★ LINKED LIST :

(a) Create a linked list

(b)
- Insertion at first pos
- Insertion at any pos
- Insertion at last pos

(c) Display

(d)
- Deletion of first
- Deletion of any specified
- Deletion of last.

(e) SORT

(f) Reverse

(g) Concatenate two list.

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

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

typedef struct node *NODE;

NODE getnode() {

    NODE x;
    x = (NODE) malloc (sizeof (struct node));

    if (x == NULL)
    {
        pf ("memory full");
        exit(0);
    }

    return x;

}

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

```
NODE insert_Front (NODE first, int item) {

    NODE temp;
    temp = getnode();
    temp -> info = item;
    temp -> link = NULL;
     if (first == NULL)
            return temp;
     temp -> link = first;
     first = temp;
      return first;
    }
```

```
NODE delete_Front (NODE first) {

    NODE temp;
    ~~temp = getnodes();~~
    ~~temp -> info~~

    if (first == NULL) {
                printf("Empty LIST");
                return first;
                ;
    temp = first;
    temp = temp -> link;

    printf("Item deleted at front end is %d", ~~temp~~ first -> info);

    free (first);
      return temp;
}
```

```
NODE insert_Rear(NODE first, int item){
    NODE temp, cur;
    temp = getnode();
    temp->info = item;
    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("EMPTY LIST");
    }
    for(temp = first ; temp != NULL; temp = temp->link)
    {
        Pf("%d", temp->info);
    }
}
```

```
NODE delete_rear(NODE first) {

    NODE cur, prev;

    if (first == NULL) {
            P/("EMPTY");
            return first;
    }

    if (first -> link == NULL) {

            P/("deleted item is %d", first->info);
        free (first);
        return NULL;
    }

    prev = NULL;
    cur = first;

    while (cur -> link != NULL) {
        prev = cur;
        cur = cur -> link
    }

    P/("item deleted is %d", cur -> info);

    fu (cur);
        preve -> link = NULL;

    return first;

}
```

```
void search (int key, NODE first) {

NODE cur;
if (first == NULL) {
                pf (" list is EMPTY");
                return;
                }

cur = first;
while( cur != NULL)
    {
        if (key == cur -> info) break;
            cur = cur -> link;
    }

if (cur == NULL)
    {
        pf (" UNSUCCESSFULL");
        return;
    }
pf (" SUCCESSFULL ");
}
```

```
int count (NODE first){

    NODE cur;
    int count = 0;

    if (first == NULL){
            Pf("length zero");
            return 0;}

    cur = first

    if(cur->link == NULL){          // not necessary
            Pf("one element");
            return 1;
    }

    while (cur != NULL){

        cur = cur->link;
        count++;
    }

    return count;

}
```

```
NODE order_list (int item, NODE first) {

    NODE temp, prev, cur;
    temp = getnode();

    temp -> info = item;
    temp -> link = NULL;

// if (first == NULL) return temp;

// if (item < first -> info)
    {
        temp -> link = first;
        return temp;
    }

    prev = NULL;
    cur = first;

// end of the condition
    while (cur != NULL && item > cur -> info)
    {
        prev = cur;
        cur = cur -> link;
    }

    prev -> link = temp;
    temp -> link = cur;

    return first;
```

```
NODE delete_info (int key, NODE first)
{
NODE prev, cur;
if (first == NULL) {
                    pf("LIST EMPTY");
                    return NULL;
                   }

if (key == first -> info) {
                    cur = first;
                    first = first -> link;
                    freenode (cur);
                    return first;
                  }

prev = NULL;
cur = first;
while (cur != NULL) {
                    if (key == cur -> info) break;
                    prev = cur;
                    cur = cur -> link;
                  }

if (cur == NULL) {
                    pf("UNSUCCESSFULL");
                    return first;
                  }
```

```
prev->link = cur->link;
pf("Key deleted is %d", cur->info);
freenode(cur);
return first;
}
```

```
NODE concat(NODE first, NODE second){
NODE cur;
if(first == NULL)
        return second;
if(second == NULL)
        return first;
cur = first;
while(cur->link != NULL)
        cur = cur->link;
cur->link = second;
return first;
}
```

```
NODE reverse (NODE first)
{
    NODE cur, temp;
    cur = NULL;

    while (first != NULL)
    {
        temp = first;
        first = first -> link;
        temp -> link = cur;
        cur = temp;
    }

    return cur;
}
```

```
int main(){
    int item, choice, pos;
    NODE first=NULL;
    int n, i;
    NODE a, b;

for(;;){
    printf("\n 1. Insert_front \n 2. Delete_front \n
        3. Insert_Rear \n 4. Delete_Rear \n 5.
        Sort List \n 6. insert_at_pos \n 7.
        delete_at_pos \n 8. Display_list \n 9.
        concate \n 10. Reverse \n");
    scanf("%d", &choice);

    switch(choice){

    case 1: pf("enter item at front");
            scanf("%d", &item);
            first = insert_front(first, item);
            break;

    case 2: first = delete_front(first);
            break;

    case 3: pf("enter at rear end");
            sf("%d", &item);

            first = insert_rear(first, item);
            break;
```

```
case 4 : first = del_at_end(first);
         break;

case 5 : pf(" enter item into added list");
         sf("%d", item);
         first = add_list(item, first);
         break;

case 6: pf(" Enter pos to insert");
        sf("%d", &pos);
        pf(" enter item");
        sf("%d", &item);
        first = insert_at_pos(pos, first, item);
        break;

case 7: pf(" enter pos to be deleted \n");
        scanf("%d", &pos);
        first = delete_at_pos(pos, first);
        break;

case 8: display(first);
        break;

case 9: pf(" enter no of nodes in a list \n");
        sf("%d", &n);

        a = NULL;
```

```
for(i=0; i<n; i++) {
        pf("enter item\n");
        sf("%d", &item);
    a = insert_rear(a, item);
}

pf("enter no. of node in 2 list");
sf("%d", &n);
b = NULL;
for(i=0; i<n; i++) {
        pf("enter the item");
        sf("%d", &item");
    b = insert_rear(b, item);
}
a = concat(a,b);
display(c);
break;

case 10: first = reverse(first);
         display(first);
         break;

default: exit(0);
         break;

}

}
```