WAP to Implement singly linked list with following operations

a) Creating a linked list
b) Insertion of a node at first position, at any position and at the end of list
c) Deletion of a node at first position, at any specified element or last element in the list
d) Display the contents of the linked list.

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<process.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)
{
printf("memory full\n");
exit(0);
}
```

```
return x;
}
void freednode (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 insert_pos (int item, int pos, NODE first)
{
NODE temp;
NODE prev, cur;
int count;
temp = getnode();
temp -> info = item;
temp -> link = NULL;
if (first == NULL && pos == 1)
return temp;
if (first == NULL)
{
printf ("invaild pos \n");
return first;
}
```

```
if (pos == 1)
{
  temp -> link = first;
  return temp;
}
count = 1;
prev = NULL;
cur = first;
while (cur != NULL && count != pos)
{
prev = cur;
cur = cur -> link;
  count++;
}
if (count == pos)
{
  prev -> link = temp;
  temp -> link = cur;
  return first;
}
else
  printf("\n invalid position : item cannot be
          inserted \n");
  return first;
}
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;
}
NODE delete_front(NODE first)
{
  NODE temp;
  if (first == NULL)
  {
    printf("list is empty cannot delete\n");
    return first;
  }
  temp = first;
  temp = temp->link;
  printf("item deleted at front-end is = %d\n",
                                  first->info);
  free(first);
  return temp;
}
NODE delete_rear(NODE first)
{
  NODE cur, prev;
  if (first == NULL)
  {
    printf("list is empty cannot be deleted\n");
    return first;
  }
  if (first->link == NULL)
  {
```

```
printf ("item deleted is %d \n", first->info);
free (first);
return NULL;
}
prev NULL;
cur = first;
while (cur->link != NULL)
{ prev = cur;
  cur = cur->link;
  }
printf ("item deleted at rear-end is %d",
                                cur->info);
free (cur);
prev->link = NULL;
return first;
}
NODE delete_pos (int pos, NODE first) {
 NODE prev, cur;
 int count;
 if (first == NULL)
 {
  printf ("list is empty. cannot delete & invalid
                position \n");
  return(first); }

 if (first->link == NULL && pos == 1)
 {
  printf ("item deleted is %d \n", first->info);
  free (first);
  return NULL;
 } count = 1;
 prev = NULL;
```

```
cur = first;
while (cur != NULL && count != pos)
{
    prev = cur;
    cur = cur->link;
    count++;
}
if (count == pos)
{
    prev->link = cur->link;
    printf("item deleted is %d\n", cur->info);
    free(cur);
    return first;
}
else
    printf("Invalid position:\n");
    return first;
}
void display(NODE first)
{
    NODE temp;
    if (first == NULL)
        printf("list empty cannot display items\n");
    for(temp = first; temp != NULL; temp = temp->link)
    {
        printf("%d\n", temp->info);
    }
}
void main()
{
    int item, choice, pos;
    NODE first = NULL;
```

```
for(;;)
{
printf("\n 1: Insert_front 2: Insert_rear
\n 3: Insert_At specified Location \n 4: Delete_
front \n 5: Delete_rear \n 6: Delete_At specif
-ed location \n 7: Display - list \n 8: Exit");
print ("enter the choice\n");
scanf("%d", &choice);
switch (choice)
{
case 1: printf("enter the item at front-end\n");
    scanf("%d", &item);
    first = insert_front (first, item);
    break;
  case 2: print ("enter the item at rear-
        end \n");
        scanf ("%d", &item);
      first = insert_rear (first, item);
      break;
case 3: printf ("enter the item to be inserted
        at location \n");
      scanf("%d", &item);
      print ("enter the position \t");
      int pos;
      scanf("%d", &pos);
      first = insert_pos (item, pos, first);
      break;
case 4: first = delete_front (first);
    break;
case 5: first = delete_rear (first);
    break;
case 6: printf ("enter the position: \t");
```

```
int pos1;
scanf("%d", &pos1);
first = delete_pos(pos1, first);
    break;
case 7: display(first);
break;
default: exist(0);
break;
}
}
}
```