Lab 7 Linked lists

```
#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)
{
   printf("memory full\n");
   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 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 insert_pos(int item, int pos, NODE first)
{
    NODE temp, prev, cur;
    temp = getnode();
    temp -> info = item;
    temp -> link = NULL;
    if ( pos == 1 && first == NULL)
      return temp;
    if (first == NULL)
    {  printf("Invalid Position\n");
       return first;
    }
    if (pos == 1)
    {
       temp -> link = first;
       return temp;
```

```
int count = 1;
cur = first;
prev = NULL;
while (cur != NULL && count != pos)
{
    prev = cur;
    cur = cur -> link;
    count ++;
}
if (count == pos)
{
    prev -> link = temp;
    temp -> link = cur;
    return first;
}
printf ("Invalid position \n");
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 delete \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 cur;
    NODE prev;
    int count;
    if (first == NULL || pos <= 0)
    {   printf ("invalid position\n");
        return NULL;
    }
```

```
if (pos == 1)
{
    cur = first;
    first = first -> link;
    freenode(cur);
    printf("Node deleted successfully \n");
    return first;
}
prev = NULL;
cur = first;
count = 1;
while (cur != NULL)
{   if (count == pos) { break; }
    prev = cur;
    cur = cur -> link;
    count ++;
}
if (count != pos)
{
    printf("Invalid position\n");
    return first;
}
prev -> link = cur -> link;
freenode(cur);
return first;
}
```

```
void swap(NODE a, NODE b)
{
    int temp = a->info;
    a->info = b->info;
    b->info = temp;
}
void bubblesort(NODE first)
{
    int swapped;
    NODE cur;
    NODE prev = NULL;
    if (first == NULL)
    {
        printf("Empty linked list \n");
        return;
    }
    do
    {
        swapped = 0;
        cur = first;
        while (cur->link != prev)
        {
            if (cur->info > cur->link->info)
            {
                swap(cur, cur->link);
                swapped = 1;
            }
            cur = cur->link;
```

```
    } prev = cur;
  } while (swapped);
}
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; }
```

```
void display (NODE first)
{
    NODE temp;
    if (first == NULL)
    printf ("list cannot display items \n");
for(temp = first ; temp != NULL; temp = temp -> link)
    {
        printf ("%d\n", temp -> info);
    }
}
int main( )
{   int item, choice, pos, i, n;
    NODE first = NULL, a, b;
    for (;;) {
    printf("\n 1: Insert_front \n 2: Insert_rear \n 3.
Insert at specified position \n 4: Delete front \n 5. Delete_rear
\n 6. Delete at specified position \n 7. sort \n 8. Concatenate
two lists \n 9. Reverse the list \n 10. Display the list \n
11. Exit \n");
printf("Enter the choice \n");
scanf("%d", &choice);
```

```
switch (choice)
{
    case 1: printf("enter the item at the front-end \n");
            scanf("%d", &item);
            first = insert_front(first, item);
            break;
    case 2: printf("Enter the item at rear-end \n");
            scanf("%d", &item);
            first = insert_rear(first, item);
            break;
    case 3: printf("enter the item \n");
            scanf("%d", &item);
            printf("enter the position \n");
            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 \n");
            scanf("%d", &pos);
            first = delete_pos(pos, first); break;
```

```
case 7: bubblesort(first);
        printf("Items in sorted order are \n");
        display(first);
        break;
case 8: printf("enter the no. of nodes in 1 \n");
        scanf("%d", &n);
        a = NULL;
        for(i=0; i<n; i++)
        {
          printf("enter the item \n");
          scanf("%d", &item);
          a = insert_rear(a, item);
        }
        printf("enter the no. of nodes in 2 \n");
        scanf("%d", &n);
        b = NULL;
        for(i=0; i<n; i++)
        {
          printf("enter the item \n");
          scanf("%d", &item);
          b = insert_rear(b, item);
        }
        a = concat(a, b);
        printf("Concatenated list : \n");
        display(a);
        break;
case 9: first = revers(first);
        display(first);
        break;
case 10: printf("List : \n");
        display(first);
        break;
case 11: exit(0);
default: printf("Enter correct instruction !!!!");
        break;
}
}
return 0;
}
```

Output :

1. Insert front
2. Insert Rear
3. Insert at specified position
4. Delete front
5. Delete Rear

6. Delete at specified pos

7. Sort

8. Concatenate

9. Reverse

10. Display

11. Exit

Enter choice : 11