Date
Page 35

23-11-20

Lab-7

Q-> Write a Program to implement Singly linked list with insertion at first, rear, at any position & display.

```
# 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;
    NODE prev, cur;
    int count;
    temp = getnode ();
    temp -> info = item;
    temp -> link = NULL;
    if ( first == NULL && pos == 1 )
    { 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");
        return first; }
    temp = first;
    temp = temp -> link;
    printf ("Item deleted at front-end = %d",
                first -> info);
    free (first);
    return temp;
}
Node delete_rear (Node first) {
```

```
Node cur, prev;
if (first == NULL) {
    printf("List is Empty");
    return first; }
if (first -> link == NULL) {
    printf("Item deleted: %d",
            first -> info);
    free (first);
    return NULL; }
prev = NULL;
cur = first;
while (cur -> link != NULL) {
    prev = cur;
    cur = cur -> link; }
printf("Item deleted: %d", cur ->
                        info);
free (cur);
prev -> link = NULL;
return first;
}
NODE delete_pos (int pos, NODE first) {
    NODE cur, 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");
    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 ");
    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 ("list is Empty");
        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 empty"); }
    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 ("\n1. Insert_front\n2. Insert
        rear\n3. Insert at pos\n4. Delete_front
        \n5. Delete_rear\n6. Delete at
        pos\n7. Sort\n8. Concatenate\n
        9. Reverse\n10. Display\n11. Exit");
        printf (" Enter choice : ");
        scanf ("%d", &choice);
        switch (choice) {
        case 1: printf (" Enter item at
            front end ");
            scanf ("%d", &item);
            first = insert_front (first, item);
            break;
        case 2: printf (" Enter item at
            rear end:");
            scanf ("%d", &item);
            first = insert_rear (first, item);
            break;
        case 3: printf (" Enter item : ");
            scanf ("%d", &item);
            printf (" Enter position : ");
```

```
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 position: ");
scanf ("%d", &pos);
first = delete_pos (pos, first);
break;
case 7: bubblesort (first);
printf ("Items in sorted
order : \n");
display (first);
break;
case 8: printf ("Enter No. of
nodes in 1 : ");
scanf ("%d", &n);
a = NULL;
for (i=0; i<n; i++){
printf ("Enter item: ");
scanf ("%d", &item);
a = insert_rear (a, item);
}
printf ("Enter No. of Nodes in 2:");
```

```
        scanf ("%d", &n);
        b = NULL;
        for (i=0; i<n; i++)
        { printf (" Enter item ");
          scanf ("%d", &item);
          b = insert_rear (b, item); }
        a = concat (a, b);
        printf (" Concatenated list : \n");
        display (a);
        break;
case 9: first = reverse (first);
        printf (" Reverse list : \n");
        display (first);
        break;
case 10: printf (" list : \n");
        display (first);
        break;
case 11: exit (0);
default: printf (" Enter proper
                  value !! ");
        break; }
}
return 0;
}
```

O/P →

1. Insert front
2. Insert Rear
3. Insert at specified pos
4. Delete front
5. Delete Rear
6. Delete at specified pos
7. Sort
8. Concatenate
9. Reverse
10. Display
11. Exit

Enter choice : 1

Enter item at front - end : 23