ASSIGNMENT NO:7 **DATE:13/03/2015**

PROGRAM TITLE:Create, Reverse and Swap the kth and (k+1)th nodes of a Linked list.

THEORY: Each node of a single linked list holds the address of only the next node in the linked list.

PROGRAM ALGORITHM:

```
Algo_insert(start,item)
{
      allocate temp node with item;
      if(start=NULL)
      {
            start=temp;
      }
      else
      {
            p=start;
            while(next(p)not equal to NULL)
      `     {
                  p=next(p);
            }
            next(p)=temp;
      }
}

Algo_swap(start,k)
{
      if (start=NULL)
      {
            print "Cannot swap as no node exists"
            return;
      }
      p=start;
      if(k=1)
      {
            swap k and k+1th nodes;
            point start to k=1th node;
      }
      else
      {
            run loop to point p to kth node;
            if(p is last node)
                  print that k+1th node doesnt exist;
            else
                  swap p and the node next to p;
      }

}

Algo_reverse(start)
{
      p=q=start;
```

```
      while(next(q)!=NULL)
      {
            p=next(q);
            next(q)=next(p);
            next(p)=start;
            start=p;
      }
}
```

PROGRAM CODE:

```
/*C Program to Create a linked list and also have functions for reversing
the linked list and swapping kth and (k+1)th node*/
#include <stdio.h>
#include <stdlib.h>
struct Node
{
      int data;
      struct Node *next;
};
typedef struct Node *NODEPTR;
NODEPTR allocate_node(int item);//Allocates memory space for a new node
int create_node(NODEPTR *start,int item);//creates a new node, can also
create start node if NULL is sent
int swapk(NODEPTR *start,int k);//swaps nodes k and k+1
int reverse(NODEPTR *start);//reverses the whole linked list;
int print(NODEPTR *start);//prints the whole linked list
//*************************************************************MAIN
FUNCTION*****************************************************************
int main()
{
    NODEPTR start=NULL;
    int ch=0,tmp;
    system("clear");
    while(ch!=5)
    {
        printf("\n\tMenu::\n\t1.Insert\n\t2.Swap elements at k and
k+1\n\t3.Reverse the list\n\t4.Print\n\t5.Exit\n\tYour choice:: ");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:printf("\n\tEnter data item:: ");
                   scanf("%d",&tmp);
                   create_node(&start,tmp);
                   break;
            case 2:printf("\n\tList before swapping");
                   print(&start);
                   printf("\n\tEnter k:: ");
                   scanf("%d",&tmp);
                   swapk(&start,tmp);
                   printf("\n\tList after swapping");
                   print(&start);
                   break;
            case 3:printf("\n\tList before reversal");
                   print(&start);
```

```
                    reverse(&start);
                    printf("\n\tList after reversal");
                    print(&start);
                    break;
             case 4:print(&start);
                    break;
             case 5:printf("\n\tProgram Terminated\n");
                    exit(0);
                    break;
             default:printf("\n\tIncorrect value entered. Enter choice
again");
         }
    }
    return 0;
}
//*************************************************************MEMBER
FUNCTIONS*****************************************************************
NODEPTR allocate_node(int item)
{
      NODEPTR temp = (NODEPTR)malloc(sizeof(struct Node));
      temp->data=item;
      temp->next=NULL;
      return temp;
}
int create_node(NODEPTR *start,int item)
{
    NODEPTR inter=allocate_node(item);
      if(*start==NULL)
      {
            *start=inter;
      }
      else
      {
            NODEPTR p=*start;
            while(p->next!=NULL)
            {
                  p=p->next;
            }
            p->next=inter;
      }
      return 0;
}
int swapk(NODEPTR *start,int k)
{
      NODEPTR p=*start;
      int i;

      /*Checks if no node exists*/
      if(p==NULL)
      {
            printf("\n\tCannot swap as no node exists.");
            return 1;
      }

      /*Checks if 1st node but not the last node*/
```

```
      if((k==1)&&((*start)->next!=NULL))
      {
            *start=p->next;
            p->next=(p->next)->next;
            (*start)->next=p;
      }
      else
      {
            NODEPTR q=*start;
            for(i=1;i<k;i++)
            {
                  q=p;
                  p=p->next;
            }
            if(p->next!=NULL)
            {
                  q->next=p->next;
                  p->next=(p->next)->next;
                  (q->next)->next=p;
            }

            /*Checks if last node*/
            else
                  printf("\n\tk+1th node doesn't exist. Swapping not
possible.");
      }
      return 0;
}
int reverse(NODEPTR *start)
{
      NODEPTR p=*start,q=*start;
      while((q->next)!=NULL)
      {
            p=q->next;
            q->next=p->next;
            p->next=*start;
            *start=p;
      }
      return 0;
}
int print(NODEPTR *start)
{
      NODEPTR p=*start;
      int i=1;
      if(*start==NULL)
    {
       printf("\n\tThe list is empty");
    }
    else
    {
      printf("\n\tThe linked list as of now is::");
      printf("\n\tPosition\tData\tAddress");
            while(p!=NULL)
            {
                  printf("\n\t\t%d:\t%d\t%p",i++,p->data,p->next);
```

```
                p=p->next;
            }
        printf("\n");
        }
        return 0;
}
```

OUTPUT:

```
Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
Your choice:: 1

Enter data item:: 10

Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
Your choice:: 1

Enter data item:: 20

Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
Your choice:: 1

Enter data item:: 30

Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
Your choice:: 1

Enter data item:: 40

Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
```

```
Your choice:: 1

Enter data item:: 50

Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
Your choice:: 2

List before swapping
The linked list as of now is::
Position   Data  Address
     1:    10    0xd7c030
     2:    20    0xd7c050
     3:    30    0xd7c070
     4:    40    0xd7c090
     5:    50    (nil)

Enter k:: 3

List after swapping
The linked list as of now is::
Position   Data  Address
     1:    10    0xd7c030
     2:    20    0xd7c070
     3:    40    0xd7c050
     4:    30    0xd7c090
     5:    50    (nil)

Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
Your choice:: 3

List before reversal
The linked list as of now is::
Position   Data  Address
     1:    10    0xd7c030
     2:    20    0xd7c070
     3:    40    0xd7c050
     4:    30    0xd7c090
     5:    50    (nil)

List after reversal
The linked list as of now is::
Position   Data  Address
     1:    50    0xd7c050
     2:    30    0xd7c070
     3:    40    0xd7c030
```

```
      4:    20    0xd7c010
      5:    10    (nil)

Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
Your choice:: 2

List before swapping
The linked list as of now is::
Position   Data  Address
      1:    50    0xd7c050
      2:    30    0xd7c070
      3:    40    0xd7c030
      4:    20    0xd7c010
      5:    10    (nil)

Enter k:: 5

k+1th node doesn't exist. Swapping not possible.
List after swapping
The linked list as of now is::
Position   Data  Address
      1:    50    0xd7c050
      2:    30    0xd7c070
      3:    40    0xd7c030
      4:    20    0xd7c010
      5:    10    (nil)

Menu::
1.Insert
2.Swap elements at k and k+1
3.Reverse the list
4.Print
5.Exit
Your choice:: 5

Program Terminated
```

DISCUSSION:

1.The complexity of creation of linked list is O(n).
2.The complexity of swapping is O(n).
3.The complexity of reversal is O(n).
4.While reversing the linked list, we have to see that the nodes are reversed and not only the data.
5.Conditions have to be placed to check if $k+1^{th}$ position exists or not while swapping.