Singly Linked List

```
#include <stdio.h>
#include <malloc.h>
#define ISEMPTY printf("\nEMPTY LIST:");

struct node
{
   int value;
   struct node *next;
};

snode* create_node(int);
void insert_node_first();
void insert_node_last();
void insert_node_pos();
void sorted_ascend();
void delete_pos();
void search();
void update_val();
void display();
```

```
void rev_display(snode *);

typedef struct node snode;
snode *newnode, *ptr, *prev, *temp;
snode *first = NULL, *last = NULL;

/*
 * Main :contains menu
 */

 int main()
 {
   int ch;
   char ans = 'Y';

   while (ans == 'Y'||ans == 'y')
   {
      printf("\n--------------------------------\n");
      printf("\nOperations on singly linked list\n");
      printf("\n--------------------------------\n");
```

```
printf("\n1.Insert node at first");
printf("\n2.Insert node at last");
printf("\n3.Insert node at position");
printf("\n4.Sorted Linked List in Ascending Order");
printf("\n5.Delete Node from any Position");
printf("\n6.Update Node Value");
printf("\n7.Search Element in the linked list");
printf("\n8.Display List from Beginning to end");
printf("\n9.Display List from end using Recursion");
printf("\n10.Exit\n");
printf("\n~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~\n");
printf("\nEnter your choice");
scanf("%d", &ch);

switch (ch)
{
case 1:
   printf("\n...Inserting node at first...\n");
   insert_node_first();
   break;
```

```
case 2:
   printf("\n...Inserting node at last...\n");
   insert_node_last();
   break;
case 3:
   printf("\n...Inserting node at position...\n");
   insert_node_pos();
   break;
case 4:
   printf("\n...Sorted Linked List in Ascending Order...\n");
   sorted_ascend();
   break;
case 5:
   printf("\n...Deleting Node from any Position...\n");
   delete_pos();
   break;
case 6:
   printf("\n...Updating Node Value...\n");
   update_val();
   break;
```

```
case 7:
   printf("\n...Searching Element in the List...\n");
   search();
   break;
case 8:
   printf("\n...Displaying List From Beginning to End...\n");
   display();
   break;
case 9:
   printf("\n...Displaying List From End using Recursion...\n");
   rev_display(first);
   break;
case 10:
   printf("\n...Exiting...\n");
   return 0;
   break;
default:
   printf("\n...Invalid Choice...\n");
   break;
}
```

```
        printf("\nYOU WANT TO CONTINUE (Y/N)");
        scanf(" %c", &ans);
    }
    return 0;
}

/*
 * Creating Node
 */
snode* create_node(int val)
{
    newnode = (snode *)malloc(sizeof(snode));
    if (newnode == NULL)
    {
        printf("\nMemory was not allocated");
        return 0;
    }
    else
    {
        newnode->value = val;
```

```
        newnode->next = NULL;
        return newnode;
    }
}

/*
 * Inserting Node at First
 */
void insert_node_first()
{
    int val;

    printf("\nEnter the value for the node:");
    scanf("%d", &val);
    newnode = create_node(val);
    if (first == last && first == NULL)
    {
        first = last = newnode;
        first->next = NULL;
        last->next = NULL;
```

```
    }
    else
    {
        temp = first;
        first = newnode;
        first->next = temp;
    }
    printf("\n----INSERTED----");
}

/*
 * Inserting Node at Last
 */
void insert_node_last()
{
    int val;

    printf("\nEnter the value for the Node:");
    scanf("%d", &val);
    newnode = create_node(val);
```

```
    if (first == last && last == NULL)
    {
        first = last = newnode;
        first->next = NULL;
        last->next = NULL;
    }
    else
    {
        last->next = newnode;
        last = newnode;
        last->next = NULL;
    }
 printf("\n----INSERTED----");
}

/*
 * Inserting Node at position
 */
void insert_node_pos()
{
```

```
int pos, val, cnt = 0, i;


printf("\nEnter the value for the Node:");
scanf("%d", &val);
newnode = create_node(val);
 printf("\nEnter the position ");
scanf("%d", &pos);
ptr = first;
while (ptr != NULL)
{
   ptr = ptr->next;
   cnt++;
}
if (pos == 1)
{
   if (first == last && first == NULL)
   {
      first = last = newnode;
      first->next = NULL;
      last->next = NULL;
```

```
    }
    else
    {
        temp = first;
        first = newnode;
        first->next = temp;
    }
    printf("\nInserted");
}
else if (pos>1 && pos<=cnt)
{
    ptr = first;
    for (i = 1;i < pos;i++)
    {
        prev = ptr;
        ptr = ptr->next;
    }
    prev->next = newnode;
    newnode->next = ptr;
    printf("\n----INSERTED----");
```

```
    }
    else
    {
        printf("Position is out of range");
    }
}

/*
 * Sorted Linked List
 */
void sorted_ascend()
{
    snode *nxt;
    int t;

    if (first == NULL)
    {
        ISEMPTY;
        printf(":No elements to sort\n");
    }
```

```
else
{
   for (ptr = first;ptr != NULL;ptr = ptr->next)
   {
      for (nxt = ptr->next;nxt != NULL;nxt = nxt->next)
      {
         if (ptr->value > nxt->value)
         {
            t = ptr->value;
            ptr->value = nxt->value;
            nxt->value = t;
         }
      }
   }
   printf("\n---Sorted List---");
   for (ptr = first;ptr != NULL;ptr = ptr->next)
   {
      printf("%d\t", ptr->value);
   }
}
```

```
}

/*
 * Delete Node from specified position in a non-empty list
 */
void delete_pos()
{
   int pos, cnt = 0, i;

   if (first == NULL)
   {
      ISEMPTY;
      printf(":No node to delete\n");
   }
   else
   {
      printf("\nEnter the position of value to be deleted:");
      scanf(" %d", &pos);
      ptr = first;
      if (pos == 1)
```

```
{
    first = ptr->next;
    printf("\nElement deleted");
}
else
{
    while (ptr != NULL)
    {
        ptr = ptr->next;
        cnt = cnt + 1;
    }
    if (pos > 0 && pos <= cnt)
    {
        ptr = first;
        for (i = 1;i < pos;i++)
        {
            prev = ptr;
            ptr = ptr->next;
        }
        prev->next = ptr->next;
```

```
            }
            else
            {
                printf("Position is out of range");
            }
        free(ptr);
        printf("\nElement deleted");
        }
    }
}
/*
 * Updating Node value in a non-empty list
 */
void update_val()
{
    int oldval, newval, flag = 0;

    if (first == NULL)
    {
        ISEMPTY;
```

```
    printf(":No nodes in the list to update\n");
}
else
{
    printf("\nEnter the value to be updated:");
    scanf("%d", &oldval);
    printf("\nEnter the newvalue:");
    scanf("%d", &newval);
    for (ptr = first;ptr != NULL;ptr = ptr->next)
    {
        if (ptr->value == oldval)
        {
            ptr->value = newval;
            flag = 1;
            break;
        }
    }
    if (flag == 1)
    {
        printf("\nUpdated Successfully");
```

```
        }
        else
        {
            printf("\nValue not found in List");
        }
    }
}

/*
 * searching an element in a non-empty list
 */
void search()
{
    int flag = 0, key, pos = 0;

    if (first == NULL)
    {
        ISEMPTY;
        printf(":No nodes in the list\n");
    }
```

```
else
{
   printf("\nEnter the value to search");
   scanf("%d", &key);
   for (ptr = first;ptr != NULL;ptr = ptr->next)
   {
      pos = pos + 1;
      if (ptr->value == key)
      {
         flag = 1;
         break;
      }
   }
   if (flag == 1)
   {
      printf("\nElement %d found at %d position\n", key, pos);
   }
   else
   {
      printf("\nElement %d not found in list\n", key);
```

```
        }
    }
}
/*
 * Displays non-empty List from Beginning to End
 */
void display()
{
   if (first == NULL)
   {
      ISEMPTY;
      printf(":No nodes in the list to display\n");
   }
   else
   {
      for (ptr = first;ptr != NULL;ptr = ptr->next)
      {
         printf("%d\t", ptr->value);
      }
   }
}
```

```
/*
 * Display non-empty list in Reverse Order
 */
void rev_display(snode *ptr)
{
   int val;

   if (ptr == NULL)
   {
      ISEMPTY;
      printf(":No nodes to display\n");
   }
   else
   {
      if (ptr != NULL)
      {
         val = ptr->value;
         rev_display(ptr->next);
         printf("%d\t", val);
      }
```

```
    }
}
```