```c
// Write a menu driven program in C to implement a Single Linked List using
Dynamic Memory Allocation and apply various
// operations on it through separate functions as below:
// a)Create a single linked list
// b)Display the list of elements
// c)Insert a sll at the beginning of the list
// d)Insert a sll at the end of the list
// e)Insert a sll at a given position in the list
// f)Insert a sll after a given sll
// g)Delete the first sll
// h)Delete the last sll
// i)Delete a sll at a given position
// j)Delete a sll after a given sll
// k)Search an element in the list
// l)Sort the elements of the list in ascending order of their values
// m)Reverse the whole list
// n)Merge one single linked list with another single linked list to form a
larger single linked list.
//(Hints: Create the first list, create the second list, sort the elements of
the first list, sort the elements of the
// second list, then merge the two lists to form a larger single linked list.)

// C program for the all operations in
// the Singly Linked List
#include <stdio.h>
#include <stdlib.h>
// Linked List Node
struct sll
{
    int val;
    struct sll *next;
};
struct sll *start = NULL;

void createList();
void traverse();
void insertAtFront();
void insertAtEnd();
void insertAtPosition();
void insertNode();
void deleteFirst();
void deleteFirst();
void deleteEnd();
void deletePosition();
void deleteNode();
void sort();
void reverseLL();
int search();
```

```c
// void mergeList();
void mean();
void maximum();

// Driver Code
int main()
{
    int i;
    int choice;
    while (1)
    {

        printf("\n\t1 Create a single linked list\n");
        printf("\t2 Display the list of elements\n");
        printf("\t3 Insert a node at the beginning of the list\n");
        printf("\t4 Insert a node at the end of the list\n");
        printf("\t5 Insert a node at a given position in the list\n");
        printf("\t6 Insert a node after a given node\n");
        printf("\t7 Delete the first node\n");
        printf("\t8 Delete the last node\n");
        printf("\t9 Delete a node at a given position\n");
        printf("\t10 Delete a node after a given node\n");
        printf("\t11 Search an element in the list\n");
        printf("\t12 Sort the elements of the list in ascending order of their
values\n");
        printf("\t13 Reverse the whole list\n");
        printf("\t14 Merge one single linked list with another single linked
list to form a larger single linked list.\n");
        printf("\t15 Find the maximum\n");
        printf("\t16 Find the mean\n");
        printf("\t17 EXIT\n");
        printf("\nEnter Choice :\n");
        scanf("%d", &choice);

        switch (choice)
        {
        case 1:
            createList();
        case 2:
            traverse();
            break;
        case 3:
            insertAtFront();
            break;
        case 4:
            insertAtEnd();
            break;
        case 5:
```

```
            insertAtPosition();
            break;
        case 6:
            insertNode();
            break;
        case 7:
            deleteFirst();
            break;
        case 8:
            deleteEnd();
            break;
        case 9:
            deletePosition();
            break;
        case 10:
            deleteNode();
            break;
        case 11:
            i = search();
            printf("The key is found at %d index",i);
            break;
        case 12:
            sort();
            break;
        case 13:
            reverseLL();
            break;
        case 14:
            // mergeList();
            break;
        case 15:
            maximum();
            break;
        case 16:
            mean();
            break;
        case 17:
            exit(1);
            break;
        default:
            printf("Incorrect Choice\n");
        }
    }
    return 0;
}
// Function to create list with n nodes initially
void createList()
{
```

```c
    struct sll *temp;
    int i; char ch;
    temp = (struct sll*)malloc(sizeof(struct sll));
    start = temp;
    printf("\nEnter the vlaue of node\n");
    scanf("%d",&temp->val);
    temp->next = NULL;
    printf("\nEnter any character to comntinue and q for quit\n");
    scanf(" %c",&ch);
    while (ch !='q')
    {
        temp->next = (struct sll*)malloc(sizeof(struct sll));
        if(temp->next == NULL)
        {
            printf("\nMemory is not allocated\n");
            exit(1);
        }
        temp = temp->next;
        printf("\nEnter the val of node\n");
        scanf("%d",&temp->val);
        temp->next = NULL;
        printf("\nEnter q to quit or any other character to continue\n");
        scanf(" %c",&ch);
    }
}

// Function to traverse the linked list
void traverse()
{
    struct sll *temp;

    // List is empty
    if (start == NULL)
        printf("\nList is empty\n");

    // Else print the LL
    else
    {
        temp = start;
        printf("Data = ");
        while (temp != NULL)
        {
            printf("%d\t", temp->val);
            temp = temp->next;
        }
    }
}
```

```c
// Function to insert at the front
// of the linked list
void insertAtFront()
{
    int data;
    struct sll *temp;
    temp = malloc(sizeof(struct sll));
    printf("\nEnter number to"
           " be inserted : ");
    scanf("%d", &data);
    temp->val = data;

    // Pointer of temp will be
    // assigned to start
    temp->next = start;
    start = temp;
}

// Function to insert at the end of
// the linked list
void insertAtEnd()
{
    int data;
    struct sll *temp, *head;
    temp = malloc(sizeof(struct sll));

    // Enter the number
    printf("\nEnter number to"
           " be inserted : ");
    scanf("%d", &data);

    // Changes links
    temp->next = 0;
    temp->val = data;
    head = start;
    while (head->next != NULL)
    {
        head = head->next;
    }
    head->next = temp;
}

// Function to insert at any specified
// position in the linked list
void insertAtPosition()
{
    struct sll *temp, *newnode;
    int pos, data, i = 1;
```

```
    newnode = malloc(sizeof(struct sll));

    // Enter the position and data
    printf("\nEnter position and data :");
    scanf("%d %d", &pos, &data);

    // Change Links
    temp = start;
    newnode->val = data;
    newnode->next = 0;
    while (i < pos - 1)
    {
        temp = temp->next;
        i++;
    }
    newnode->next = temp->next;
    temp->next = newnode;
}

//function to insert a node at a given position
void insertNode()
{
    int key;
    printf("Enter the position\n");
    scanf("%d",&key);
    struct sll *new,*temp;
    temp=start;
    int i,sz,n;
    n=search();
    n++;
    for(sz=1; temp->next!=NULL; sz++)
        temp=temp->next;
    if(n>1 && n<=sz)
    {
        for(i=1; i<n-1; i++)
            temp=temp->next;
        new=(struct sll *)malloc(sizeof(struct sll));
        if(new==NULL)
        {
            printf("\nMemory not allocated Properly !\n");
            exit(0);
        }
        printf("\nEnter the Value : ");
        scanf("%d",&new->val);
        (new->next)=(temp->next);
        temp->next=new;
    }
    else if(n==sz+1)
```

```
        insertAtEnd();
    else
        printf("\nNode Not Found !\n");
}

// Function to delete from the front
// of the linked list
void deleteFirst()
{
    struct sll *temp;
    if (start == NULL)
        printf("\nList is empty\n");
    else
    {
        temp = start;
        start = start->next;
        free(temp);
    }
}

// Function to delete from the end
// of the linked list
void deleteEnd()
{
    struct sll *temp, *prevnode;
    if (start == NULL)
        printf("\nList is Empty\n");
    else
    {
        temp = start;
        while (temp->next != 0)
        {
            prevnode = temp;
            temp = temp->next;
        }
        free(temp);
        prevnode->next = 0;
    }
}

// Function to delete from any specified
// position from the linked list
void deletePosition()
{
    struct sll *temp, *position;
    int i = 1, pos;

    // If LL is empty
```

```
    if (start == NULL)
        printf("\nList is empty\n");

    // Otherwise
    else
    {
        printf("\nEnter index : ");

        // Position to be deleted
        scanf("%d", &pos);
        position = malloc(sizeof(struct sll));
        temp = start;

        // Traverse till position
        while (i < pos - 1)
        {
            temp = temp->next;
            i++;
        }

        // Change Links
        position = temp->next;
        temp->next = position->next;

        // Free memory
        free(position);
    }
}

void deleteNode()
{
    int key;
    printf("Enter the position\n");
    scanf("%d",&key);
    struct sll *temp;
    int n,sz;
    temp=start;
    for(sz=1; temp->next!=NULL; sz++)
        temp=temp->next;
    n=search();
    n++;
    if(n==sz)
        deleteEnd();
    else if(n>1 && n<sz)
    {
        int i;
        for(i=1; i<n; i++)
        {
```

```
            temp=start;
            start=start->next;
        }
        (temp->next)=(start->next);
        free(start);
    }
    else if(n==sz+1)
        printf("\nNo element is present !\n");
    else
        printf("\nNode is Absent !\n");

}

// Function to find the maximum element
// in the linked list
void maximum()
{
    int a[10];
    int i;
    struct sll *temp;
    temp = (struct sll  *)malloc(sizeof(struct sll));
    // If LL is empty
    if (start == NULL)
        printf("\nList is empty\n");

    // Otherwise
    else
    {
        temp = start;
        int max = temp->val;

        // Traverse LL and update the
        // maximum element
        while (temp != NULL)
        {

            // Update the maximum
            // element
            if (max < temp->val)
                max = temp->val;
            temp = temp->next;
        }
        printf("\nMaximum number is : %d ",max);
    }
}

// Function to find the mean of the
// elements in the linked list
```

```c
void mean()
{
    int a[10];
    int i;
    struct sll *temp;

    // If LL is empty
    if (start == NULL)
        printf("\nList is empty\n");

    // Otherwise
    else
    {
        temp = start;

        // Stores the sum and count of
        // element in the LL
        int sum = 0, count = 0;
        float m;

        // Traverse the LL
        while (temp != NULL)
        {

            // Update the sum
            sum = sum + temp->val;
            temp = temp->next;
            count++;
        }

        // Find the mean
        m = sum / count;

        // Print the mean val
        printf("\nMean is %f ", m);
    }
}

// Function to sort the linked list
// in ascending order
void sort()
{
    struct sll *current = start;
    struct sll *index = NULL;
    int temp;

    // If LL is empty
    if (start == NULL)
```

```
    {
        return;
    }

    // Else
    else
    {

        // Traverse the LL
        while (current != NULL)
        {
            index = current->next;

            // Traverse the LL nestedly
            // and find the minimum
            // element
            while (index != NULL)
            {

                // Swap with it the val
                // at current
                if (current->val > index->val)
                {
                    temp = current->val;
                    current->val = index->val;
                    index->val = temp;
                }
                index = index->next;
            }

            // Update the current
            current = current->next;
        }
    }
}

int search()
{
    int key;
    struct sll *temp;
    temp = start;
    printf("Enter the key\n");
    scanf("%d",&key);
    int i;
    for(i=1; temp!=NULL; i++)
    {
        if((temp->val)==key)
            return i;
```

```
            break;
        temp=temp->next;
    }
    return 0;
}
// Function to reverse the linked list
void reverseLL()
{
    struct sll *t1, *t2, *temp;
    t1 = t2 = NULL;

    // If LL is empty
    if (start == NULL)
        printf("List is empty\n");

    // Else
    else
    {

        // Traverse the LL
        while (start != NULL)
        {

            // reversing of points
            t2 = start->next;
            start->next = t1;
            t1 = start;
            start = t2;
        }
        start = t1;

        // New head Node
        temp = start;

        printf("Reversed linked "
               "list is : ");

        // Print the LL
        while (temp != NULL)
        {
            printf("%d ", temp->val);
            temp = temp->next;
        }
    }
}
```