```
#include<stdio.h>
#include<malloc.h>
struct node
{
    int data;
    struct node *next;
};
struct node *start = NULL;
struct node *create_ll(struct node*);
struct node *display(struct node *);
struct node *insert_beg(struct node *);
struct node *insert_end(struct node *);
struct node *insert_before(struct node*);
struct node *insert_after(struct node *);
struct node *delete_beg(struct node *);
struct node *delete_end(struct node *);
struct node *delete_node(struct node *);
struct node *delete_after(struct node *);
struct node *delete_list(struct node *);
struct node *sort_list(struct node *);
int main()
{
    int choice;
    printf("\n***menu***\n1.create a list\t2.display\t3.insert_beg\t4.insert_end\t5.insert_bef
```

```
\t 6. insert_after ");
printf (" \t 7. del_beg \t 8. del_end \t 9. del_node
\t 10. del_after \t 11. del_list \t 12 Sort_list
\t 13. exit \n");
do
{
    printf("\n enter the choice:");
    scanf ("%d", &choice);
    switch (choice)
    {
        case 1: start = create_ll(start);
                printf ("\n Linked list created");
                break;
        case 2: start = display (start);
                break;
        case 3: start = insert_beg (start);
                break;
        case 4: start = insert_end(start);
                break;
        case 5: start = insert_before(start);
                break;
        case 6: start = insert_after(start);
                break;
        case 7: start = delete_beg(start);
                break;
```

```
case 8: start = delete_end(start);
        break;
case 9: start = delete_node(start);
        break;
case 10: start = delete_after(start);
        break;
case 11: start = delete_list(start);
        break;
case 12: start = sort_list(start);
        break;
    }
}while(choice != 13);
return 0;
}
struct node *creat_ll(struct node *start)
{
struct node *new_node, *ptr;
int num;
printf("\n enter -1 to end");
printf("\n enter the data: ");
scanf("%d", &num);
while(num != -1)
{
    new_node = (struct node *) malloc(sizeof(
                struct node));
```

```
new_node -> data = num;
if (start == NULL)
{
    new_node -> next = NULL;
    start = new_node;
}
else
{
    ptr = start;
    while (ptr -> next != NULL)
        ptr = ptr -> next;
    ptr -> next = new_node;
    new_node -> next = NULL;
}
printf("\n Enter the data: ");
scanf("%d", &num);
}
return start;
};
struct node *display (struct node *start)
{
    struct node *ptr;
    ptr = start;
    while (ptr != NULL)
    {
```

```
        printf("%d", ptr->data);
        ptr = ptr->next;
    }
    return start;
};

struct node *insert_beg(struct node *start)
{
    struct node *new_node;
    int num;
    printf("\nEnter the data:");
    scanf("%d", &num);
    new_node = (struct node *)malloc(sizeof(struct
                node));
    new_node->data = num;
    new_node->next = start;
    start = new_node;
    return start;
};
struct node *insert_end(struct node *start)
{
    struct node *new_node, *ptr;
    int num;
    printf("\nEnter the data:");
    scanf("%d", &num);
```

```
new_node = (struct node*) malloc (sizeof (struct
                                    node));
new_node->data = num;
new_node->next = NULL;
ptr = start;
if (start == NULL)
{
    start = new_node;
}
else
{
  while (ptr->next != NULL)
          ptr = ptr->next;
  ptr->next = new_node;
  return start;
}
};
struct node *insert_before(struct node *start)
{
struct node *new_node, *ptr, *preptr;
int num, val;
printf("\nenter the data:");
scanf("%d", &num);
printf("\nenter the value before which
      data has to be inserted:");
```

```
scanf("%d", &val);
new_node = (struct node*) malloc(sizeof(
        struct node));
new_node->data = num;
ptr = start;
while (ptr->data != val)
{
        preptr = ptr;
        ptr = ptr->next;
}
preptr->next = new_node;
new_node->next = ptr;
return start;
};
struct node *insert_after(struct nod *start)
{
        struct node *new_node, *ptr, *preptr;
        int num, val;
        printf("\n enter the data:");
        scanf("%d", &num);
        printf("\nenter the value after which
                data has to be inserted");
        scanf("%d", &val);
        new_node = (struct node *) malloc(sizeof(
                struct node));
```

```
new_node -> data = num;
ptr = start;
preptr = ptr;
while (preptr -> data != val)
{
    preptr = ptr;
    ptr = ptr -> next;
}
preptr -> next = new_node;
new_node -> next = ptr;
return start;
};
struct node * delete_beg (struct node *start)
{
    struct node * ptr;
    ptr = start;
    start = start -> next;
    free(ptr);
    return start;
};
struct node *delete_end(struct node *start)
{
    struct node *ptr, *preptr;
    ptr = start;
    while (ptr -> next != NULL)
```

```
{
    preptr = ptr;
    ptr = ptr->next;
}
preptr->next = NULL;
free(ptr);
return start;
};
Struct node *delete_node(struct node *start)
{
    Struct node *ptr, *preptr;
    int val;
    printf("\nEnter the value to be deleted:");
    scanf("%d", &val);
    ptr = start;
    if(ptr->data == val)
    {
        start = delete_beg(start);
        return start;
    }
    else
    {
        while(ptr->data != val)
        {
            preptr = ptr;
```

```
        ptr = ptr->next;
    }
    prevptr->next = ptr->next;
    free(ptr);
    return start;
}
};
struct node *delete_after(struct node *start)
{
struct node *ptr, *preptr;
int val;
printf("\n Enter the value after which the
        node has to be deleted ");
scanf("%d", &val);
ptr = start;
preptr = ptr;
while (preptr->data != val)
{
        preptr = ptr;
        ptr = ptr->next;
}
preptr->next = ptr->next;
free(ptr);
return start;
};
```

```
struct node *delete_list(struct node *start)
{
    struct node *ptr;
    if(start != NULL)
    {
        ptr = start;
        while(ptr != NULL)
        {
            printf("%d is deleted in", ptr->data
                  );
            start = delete_beg(ptr);
            ptr = start;
        }
    }
    return start;
};
struct node *sort_list(struct node *start)
{
    struct node *ptr1, *ptr2;
    int temp;
    ptr1 = start;
    while(ptr1->next != NULL)
    {
        ptr2 = ptr1->next;
        while(ptr2 != NULL)
```

```
{
    if (ptr1->data > ptr2->data)
    {
        temp = ptr1->data;
        ptr1->data = ptr2->data;
        ptr2->data = temp;
    }
    ptr2 = ptr2->next;
}
ptr1 = ptr1->next;
}
display(start);
};
```