Shri Ramdeobaba College of Engineering and Management, Nagpur
Department of Computer Science and Engineering - Cyber Security
B.Tech. 3rd Semester, Session: 2022-2023

Student Name:	Saloni Vishwakarma
Roll No:	13
Practical No:	05
Aim:	To study and implement Doubly Linked List.

Source Code:

```
#include<stdio.h>
#include<stdlib.h>
struct node{
        int data;
        struct node* next,*prev;
};
struct node*insert_before(struct node*start)
{
   struct node *ptr,*preptr,*nn;
   int val;
   nn=(struct node*)malloc(sizeof(struct node));
   printf("\nInserting a node before a node ");
   printf("\nEnter the node data before which a new node is to be inserted: ");
   scanf("%d",&val);
   printf("\nEnter the data: ");
   scanf("%d",&nn->data);
   preptr=start;
   ptr=preptr->next;
   while(ptr->data!=val)
   {
      ptr=ptr->next;
      preptr=preptr->next;
   }
   nn->prev=preptr;
   nn->next=ptr;
   preptr->next=nn;
   ptr->prev=nn;
   return start;
}
struct node*insert_after(struct node*start)
{
   struct node *ptr,*preptr,*nn;
   int val;
   nn=(struct node*)malloc(sizeof(struct node));
   printf("\nInserting a node after a node ");
   printf("\nEnter the node data after which a new node is to be inserted: ");
   scanf("%d",&val);
   printf("\nEnter the data: ");
   scanf("%d",&nn->data);
```

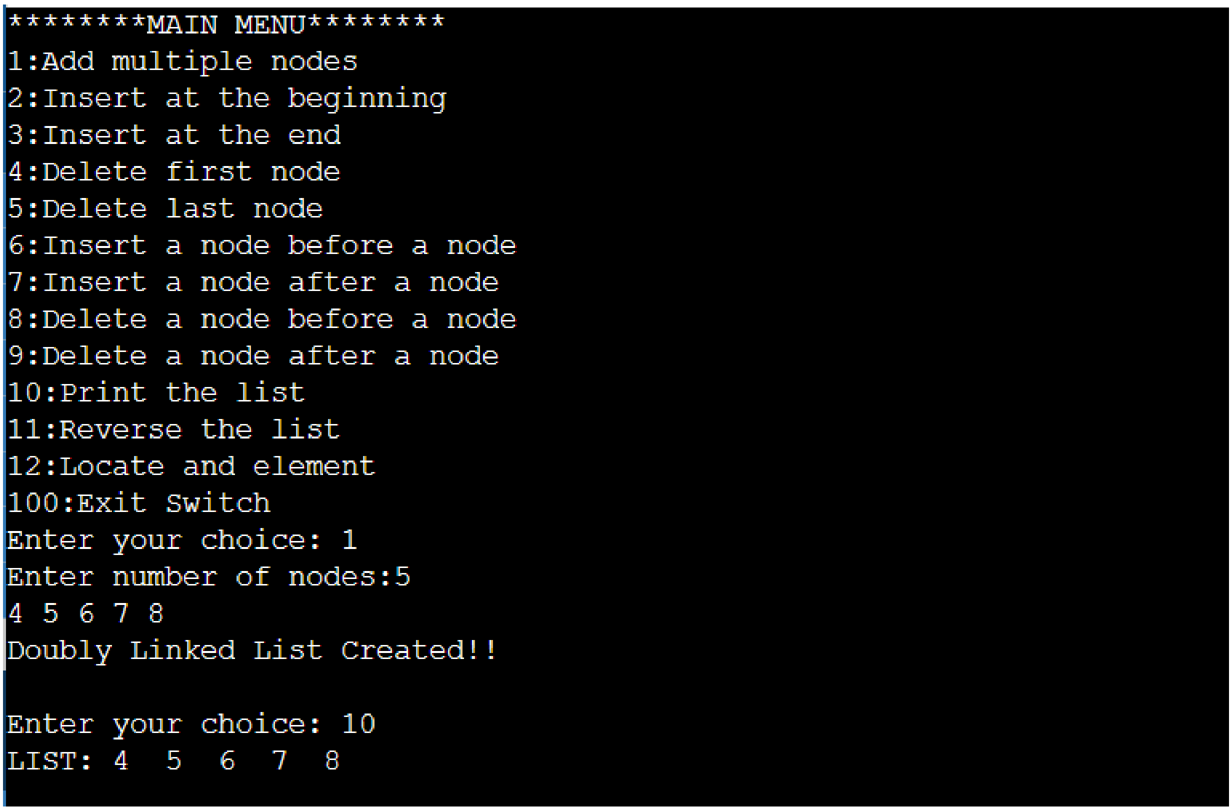
```
  preptr=start;
  ptr=preptr->next;
  while(preptr->data!=val)
  {
    ptr=ptr->next;
    preptr=preptr->next;
  }
  nn->prev=preptr;
  nn->next=ptr;
  preptr->next=nn;
  ptr->prev=nn;
  return start;
}
struct node* createNode(){
  struct node* newNode=(struct node*)malloc(sizeof(struct node));
  if(newNode==NULL){
    printf("Stack overflow\n");
  }
  else{
    return newNode;
  }
return 0;
}
struct node* newNode(struct node* start){
  struct node* newNode=createNode();
  struct node* temp;
  if(newNode == NULL){
      printf("ALLOCATION FAILED!!");
      }
  else{
      int key;
      scanf("%d",&key);

      newNode->data = key;
      newNode->next = NULL;
      newNode->prev = NULL;
      }
  if(start == NULL){
      start = newNode;
      }
  else{
      temp = start;
      while(temp->next!=NULL){
              temp = temp->next;
      }
      temp->next = newNode;
      newNode->prev = temp;
      }
return start;
}
```

```
struct node* insertatEnd(struct node* start){
   struct node* newNode=createNode();
   struct node* temp;
   if(newNode == NULL){
        printf("ALLOCATION FAILED!!");
        }
   else{
        int key;
        printf("Inserting the data at the end\n");
        printf("Enter the data to be inserted: ");
        scanf("%d",&key);

        newNode->data = key;
        newNode->next = NULL;
        newNode->prev = NULL;
        }
   if(start == NULL){
        start = newNode;
        }
   else{
        temp = start;
        while(temp->next!=NULL){
                temp = temp->next;
        }
        temp->next = newNode;
        newNode->prev = temp;
        }
return start;
}
struct node* insertatBeg(struct node* start){
   struct node* newNode=createNode();
   struct node* temp;
   if(newNode == NULL){
        printf("ALLOCATION FAILED!!");
        }
   else{
        int key;
        printf("Inserting the data at the beginning\n");
        printf("Enter the data to be inserted: ");
        scanf("%d",&key);

        newNode->data = key;
        newNode->next = NULL;
        newNode->prev = NULL;
        }
   if(start == NULL){
        start = newNode;
        }
   else{
        newNode->next = start;
        start->prev = newNode;
        start = newNode;
```

```
    }
return start;
}
struct node* deleteatBeg(struct node* start){
    if(start == NULL){
        printf("Linked List is Empty\n");
    }
    else{
       printf("Deleting the first node\n");
        if(start->next == NULL){
            free(start);
            start = NULL;
        }
        else{
            start = start->next;
            free(start->prev);
            start->prev = NULL;
        }
    }
return start;
}
struct node* deleteatEnd(struct node* start){
    if(start == NULL){
        printf("Linked List is Empty\n");
    }
    else{
       printf("Deleting the last node\n");
        if(start->next == NULL){
            free(start);
            start = NULL;
        }
        else{
            struct node* temp = start;
            while(temp->next != NULL){
                temp = temp->next;
            }
            temp->prev->next = NULL;
            free(temp);
        }
    }
return start;
}
void delete_before(struct node *start) {
  int num;
  printf("Deleting a node before a node\n");
  printf ("Enter value before which the node is to be deleted:");
  scanf ("%d", &num);
  struct node *ptr, *preptr;
  ptr = start;
  while (ptr->data != num){
      preptr = ptr;
      ptr = ptr->next;
  }
  (preptr->prev)->next = ptr;
  ptr->prev = preptr->prev;
};
void delete_after(struct node *start) {
  int num;
  printf("Deleting a node after a node\n");
  printf ("Enter value after which the node is to be deleted:");
  scanf ("%d", &num);
  struct node *ptr, *preptr;
  ptr = start;
  while (preptr->data != num){
      preptr = ptr;
      ptr = ptr->next;
    }
  preptr->next = ptr->next;
  (ptr->next)->prev = preptr;
};
void printList(struct node* start){
    struct node* ptr=start;
    if(start==NULL){
        printf("Linked list Empty\n");
    }
    else{
        printf("LIST: ");
        while(ptr!=NULL){
            printf("%d  ",ptr->data);
            ptr=ptr->next;
        }
        printf("\n");
    }
}
void reversePrint(struct node* start){
          struct node* temp = start;
          while(temp->next != NULL){
                      temp = temp->next;
          }
          printf("Reversed list is: ");
          while(temp!=start){
                      printf("%d\n", temp->data);
                      temp = temp->prev;
          }
          printf("%d\n", temp->data);
}
void locate (struct node *start){
    int num, k = 0;
    printf ("Enter data to be located:");
  scanf ("%d", &num);
  struct node *ptr;
  ptr = start;
  while (ptr->data != num){
    ptr = ptr->next;
    k++;
   }
 printf ("\nElement is at %d location\n", k + 1);
 }
 int main(){
         struct node* start=NULL;
    printf("********MAIN MENU********");
         printf("\n1:Add multiple nodes\n2:Insert at the beginning\n3:Insert at the end\n4:Delete first
 node\n5:Delete last node\n");
    printf("6:Insert a node before a node\n7:Insert a node after a node\n");
    printf("8:Delete a node before a node\n9:Delete a node after a node\n");
         printf("10:Print the list\n11:Reverse the list\n12:Locate and element\n100:Exit Switch");

    while(1){
       int choice, n, key;
       printf("\nEnter your choice: ");
         scanf("%d",&choice);
         switch(choice){
                   case 1: printf("Enter number of nodes:");
                             scanf("%d",&n);
                             for(int i=0;i<n;i++){
                             start=newNode(start);
                             }
               printf("Doubly Linked List Created!!\n");
                   break;
          case 2: start = insertatBeg(start);
             break;
          case 3: start = insertatEnd(start);
                   break;
          case 4: start = deleteatBeg(start);
                   break;
          case 5: start = deleteatEnd(start);
             break;
          case 6: start=insert_before(start);
             break;
          case 7: start=insert_after(start);
             break;
          case 8: delete_before(start);
             break;
          case 9: delete_after(start);
             break;
          case 10: printList(start);
             break;
          case 11: reversePrint(start);
             break;
          case 12: locate(start);
             break;
          case 100: exit(0);
         }
       }
         return 0;
}
```

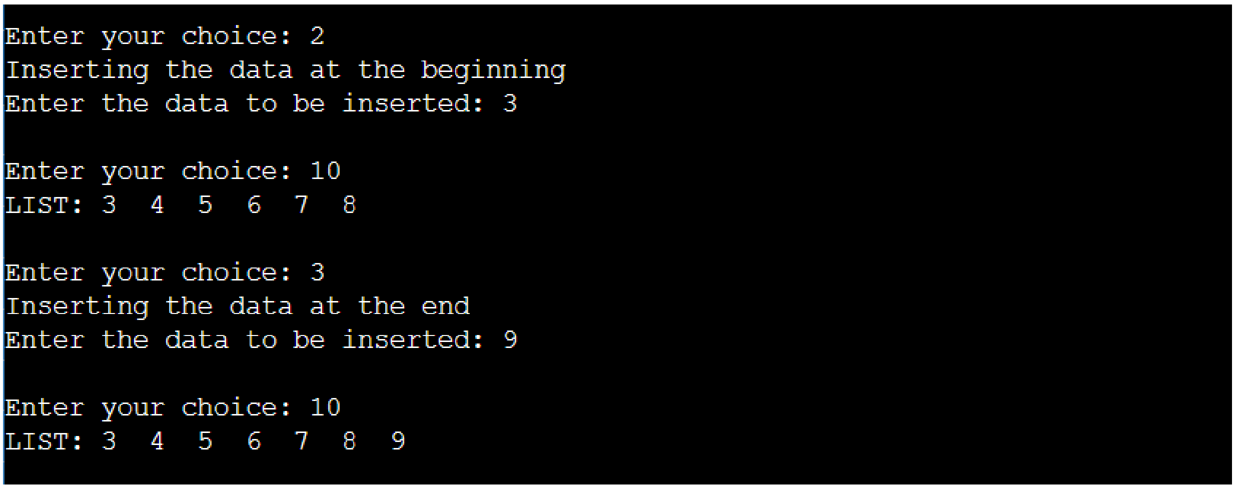
Output:

1. Creating nodes and printing the data of list.



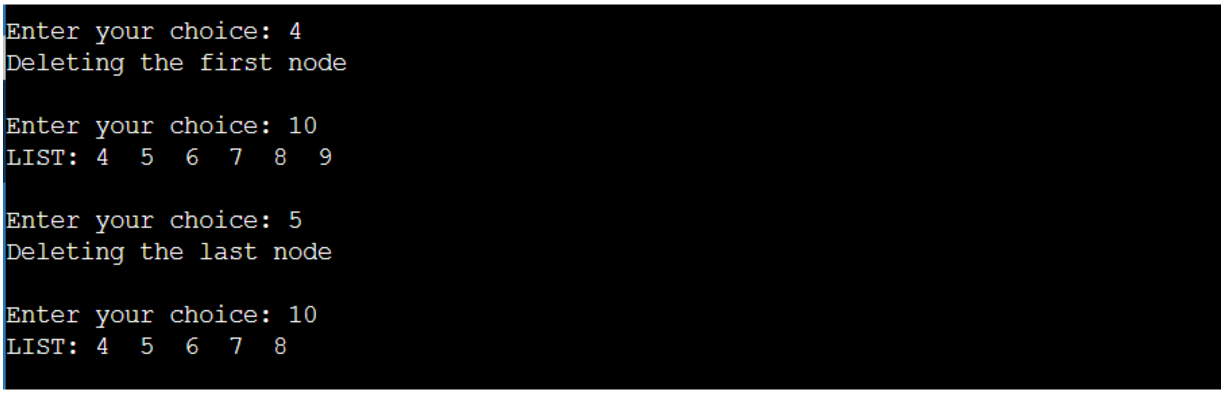
```
********MAIN MENU********
1:Add multiple nodes
2:Insert at the beginning
3:Insert at the end
4:Delete first node
5:Delete last node
6:Insert a node before a node
7:Insert a node after a node
8:Delete a node before a node
9:Delete a node after a node
10:Print the list
11:Reverse the list
12:Locate and element
100:Exit Switch
Enter your choice: 1
Enter number of nodes:5
4 5 6 7 8
Doubly Linked List Created!!

Enter your choice: 10
LIST: 4  5  6  7  8
```

2. Inserting node at the beginning and at the end.



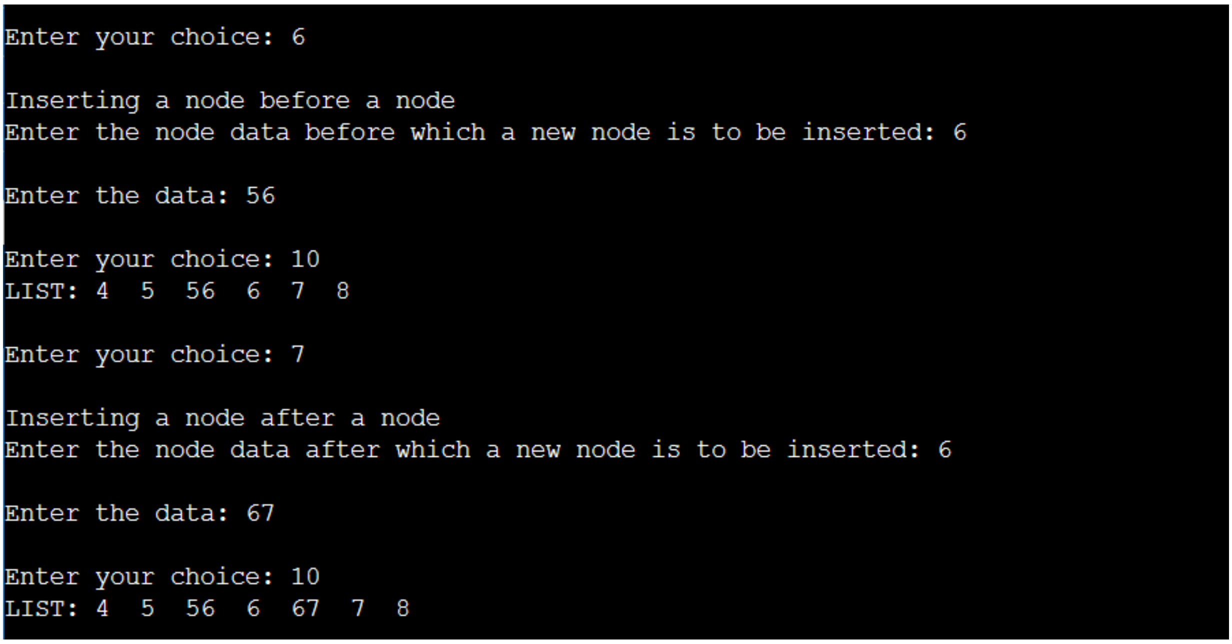
```
Enter your choice: 2
Inserting the data at the beginning
Enter the data to be inserted: 3

Enter your choice: 10
LIST: 3  4  5  6  7  8

Enter your choice: 3
Inserting the data at the end
Enter the data to be inserted: 9

Enter your choice: 10
LIST: 3  4  5  6  7  8  9
```

3. Deleting the first and last node.



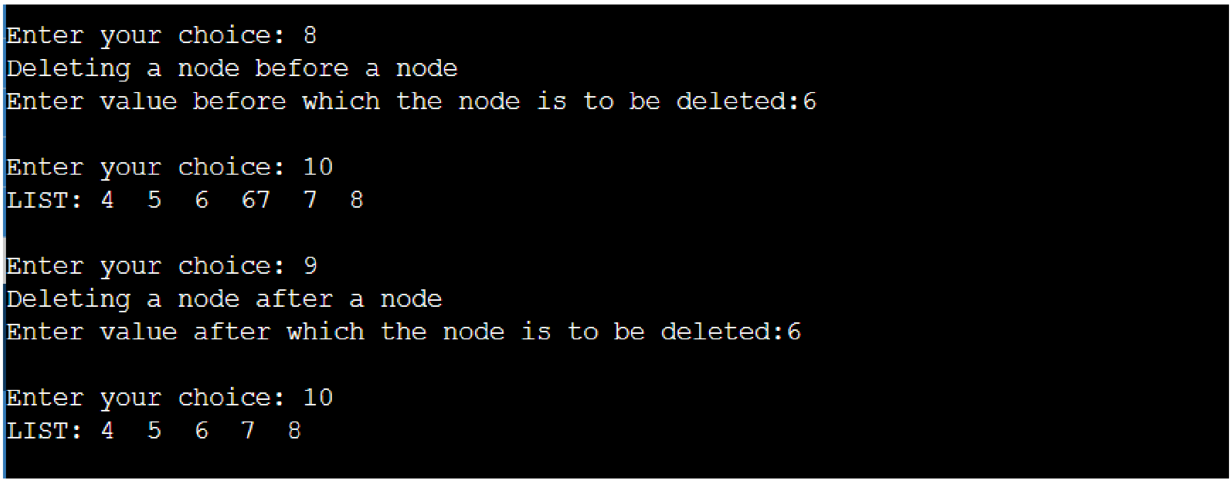
```
Enter your choice: 4
Deleting the first node

Enter your choice: 10
LIST: 4  5  6  7  8  9

Enter your choice: 5
Deleting the last node

Enter your choice: 10
LIST: 4  5  6  7  8
```

4. Inserting a node before and after a node.



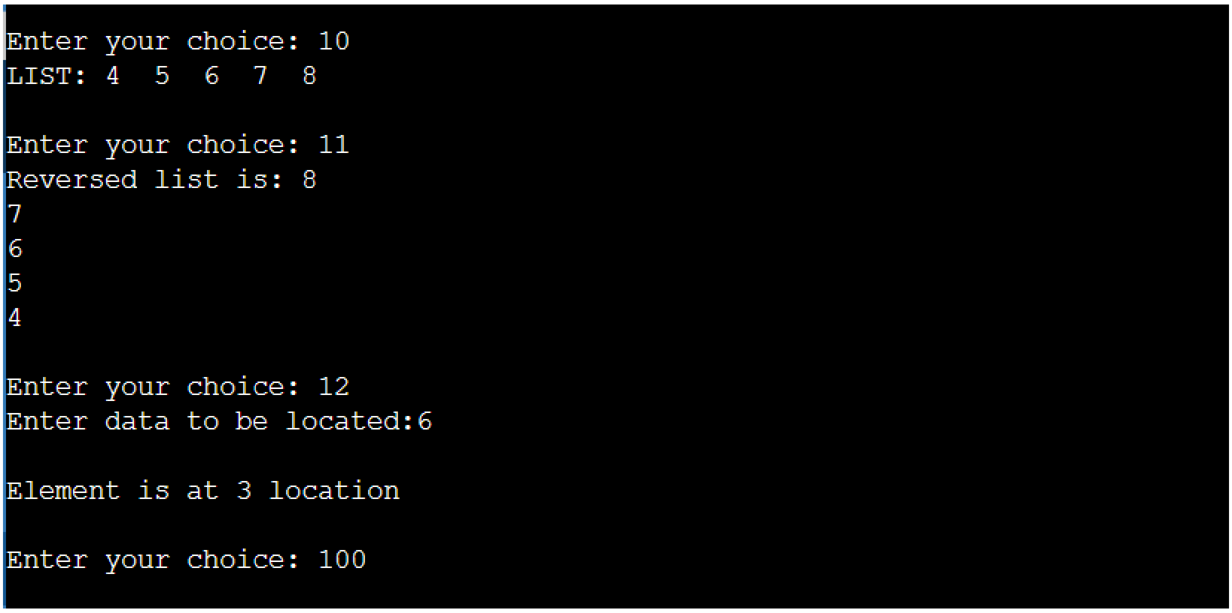
```
Enter your choice: 6

Inserting a node before a node
Enter the node data before which a new node is to be inserted: 6

Enter the data: 56

Enter your choice: 10
LIST: 4  5  56  6  7  8

Enter your choice: 7

Inserting a node after a node
Enter the node data after which a new node is to be inserted: 6

Enter the data: 67

Enter your choice: 10
LIST: 4  5  56  6  67  7  8
```

5. Deleting a node before and after a node.



```
Enter your choice: 8
Deleting a node before a node
Enter value before which the node is to be deleted:6

Enter your choice: 10
LIST: 4  5  6  67  7  8

Enter your choice: 9
Deleting a node after a node
Enter value after which the node is to be deleted:6

Enter your choice: 10
LIST: 4  5  6  7  8
```

6. Reversing the list and locating an element.



```
Enter your choice: 10
LIST: 4   5   6   7   8

Enter your choice: 11
Reversed list is: 8
7
6
5
4

Enter your choice: 12
Enter data to be located:6

Element is at 3 location

Enter your choice: 100
```

Result: The concept of Binary Search Tree has been studied and various allowable operationsof singly linked list have been implemented.