VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

NIKET DUGAR (1BM22CS180)

in partial fulfillment for the award of the degree of
BACHELOR OF ENGINEERING
in
COMPUTER SCIENCE AND ENGINEERING

B.M.S. COLLEGE OF ENGINEERING
(Autonomous Institution under VTU)
BENGALURU-560019
Dec 2023- March 2024

B. M. S. College of Engineering,
Bull Temple Road, Bangalore 560019
(Affiliated To Visvesvaraya Technological University, Belgaum)
Department of Computer Science and Engineering

This is to certify that the Lab work entitled **"DATA STRUCTURES"** carried out by NIKET DUGAR **(1BM22CS180)**, who is a bonafide student of **B. M. S. College of Engineering**. It is in partial fulfillment for the award of **Bachelor of Engineering in Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year 2023-24. The Lab report has been approved as it satisfies the academic requirements in respect of Data structures Lab - **(23CS3PCDST) work** prescribed for the said degree.

Prof. Sneha S Bagalkot , Dr. Jyothi S Nayak Assistant Professor Professor
and Head Department of CSE Department of CSE BMSCE, Bengaluru
BMSCE, Bengaluru

Index Sheet

Course outcomes:

CO1	Apply the concept of linear and nonlinear data structures.
CO2	Analyze data structure operations for a given problem
CO3	Design and develop solutions using the operations of linear and nonlinear data structure for a given specification.
CO4	Conduct practical experiments for demonstrating the operations of different data structures.

Lab program 1:

Write a program to simulate the working of stack using an array with the following:

a) Push
b) Pop
c) Display
The program should print appropriate messages for stack overflow, stack underflow.

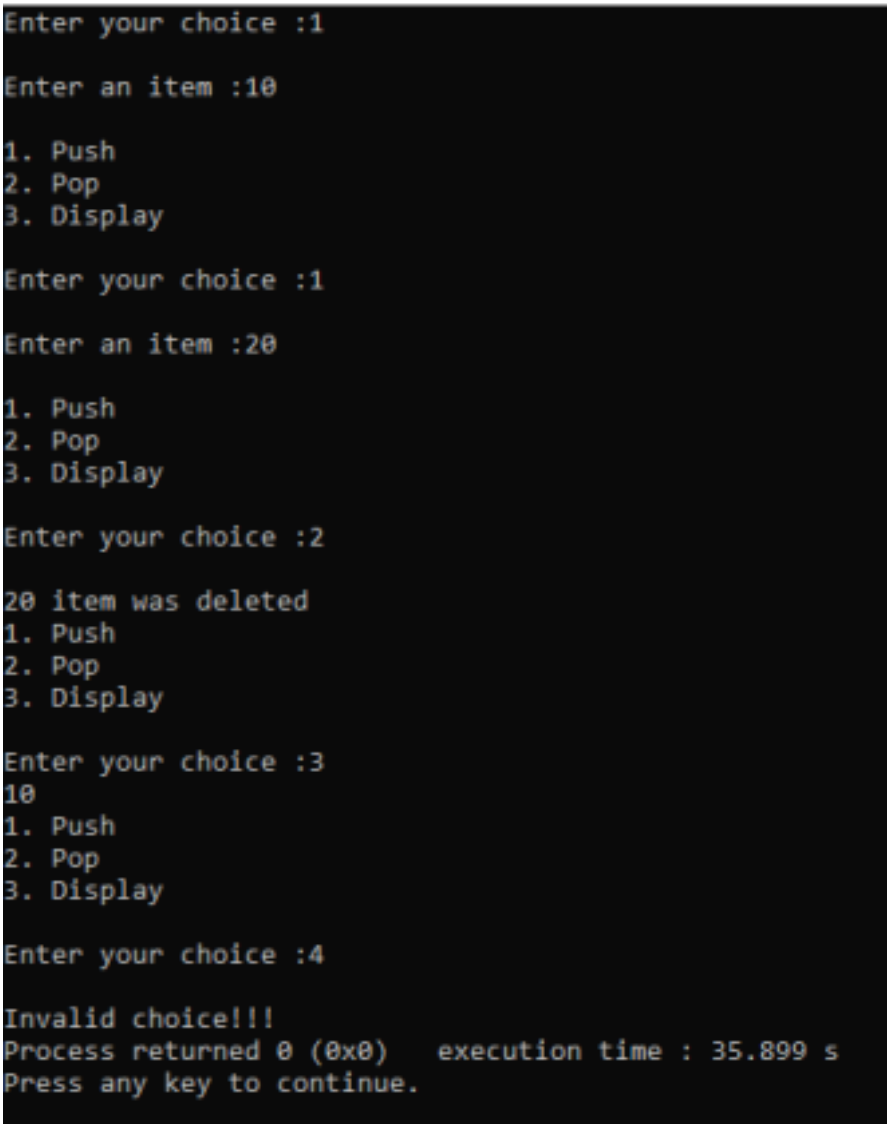
```
#include <stdio.h>
#include<stdlib.h>
#define STACK_SIZE 5
void push(int st[],int *top)
{
        int item;
        if(*top==STACK_SIZE-1)
                printf("Stack overflow\n");
        else
        {
                printf("\nEnter an item :");
                scanf("%d",&item);
                (*top)++;
                st[*top]=item;
        }
}
void pop(int st[],int *top)
{
        if(*top==-1)
                printf("Stack underflow\n");
        else
        {
                printf("\n%d item was deleted",st[(*top)--]);
        }
}
void display(int st[],int *top)
{
        int i;
        if(*top==-1)
                printf("Stack is empty\n");
        for(i=0;i<=*top;i++)
                printf("%d\t",st[i]);
}
```

```
void main()
{
        int st[10],top=-1, c,val_del;
        while(1)
        {
                printf("\n1. Push\n2. Pop\n3. Display\n");
                printf("\nEnter your choice :");
                scanf("%d",&c);
                switch(c)
                {
```



```
                        case 1: push(st,&top);
                                break;
                        case 2: pop(st,&top);
                                break;
                        case 3: display(st,&top);
                                break;
                        default: printf("\nInvalid choice!!!");
                                exit(0);
                }
        }
}
```

Output:


Enter your choice :1
Enter an item :10
1. Push
2. Pop
3. Display
Enter your choice :1
Enter an item :20
1. Push
2. Pop
3. Display
Enter your choice :2
20 item was deleted
1. Push
2. Pop
3. Display
Enter your choice :3
10
1. Push
2. Pop
3. Display
Enter your choice :4
Invalid choice!!!
Process returned 0 (0x0) execution time : 35.899 s
Press any key to continue.

Lab program 2:

WAP to convert a given valid parenthesized infix arithmetic expression to postfix expression. The expression consists of single character operands and the binary operators + (plus), - (minus), * (multiply) and / (divide)

```
#include <stdio.h>
#include <string.h>
int top;
char A[100];
void push(char x){
   if(top==9){
      printf("Stack overflow!");
   }
   else{
      A[++top] = x;
   }
}
char pop(){
   int x;
   if(top==-1){
      printf("Stack underflow!");
   }
   else{
      x = A[top];
      top--;
   }
   return x;
}
int pref(char x){
   if(x=='+' || x=='-'){
      return 1;
   }
   else if(x=='*' || x=='/'){
      return 2;
   }
   else if(x=='^'){
      return 3;
   }
   else
      return 0;
}
void main(){
   char s[] = "A*B+C*D-E", postfix[100], x, y;
```



```
   int j=0;
```

```
    for(int i=0;i<strlen(s);i++){
        if(s[i]!='+' && s[i]!='-' && s[i]!='*' && s[i]!='/'){
            postfix[j++]=s[i];
        }
        else{
            while(top!=-1 && pref(A[top]) >= pref(s[i])){
                x = pop();
                postfix[j++] = x;
            }
            push(s[i]);

        }
    }
    while(top!=-1){
        y = pop();
        postfix[j++] = y;
    }
    postfix[j] = '\0';
    printf("%s",postfix);
}
```

Output:

```
AB*CD*+E-
Process returned 9 (0x9)   execution time : 0.006 s
Press any key to continue.
```

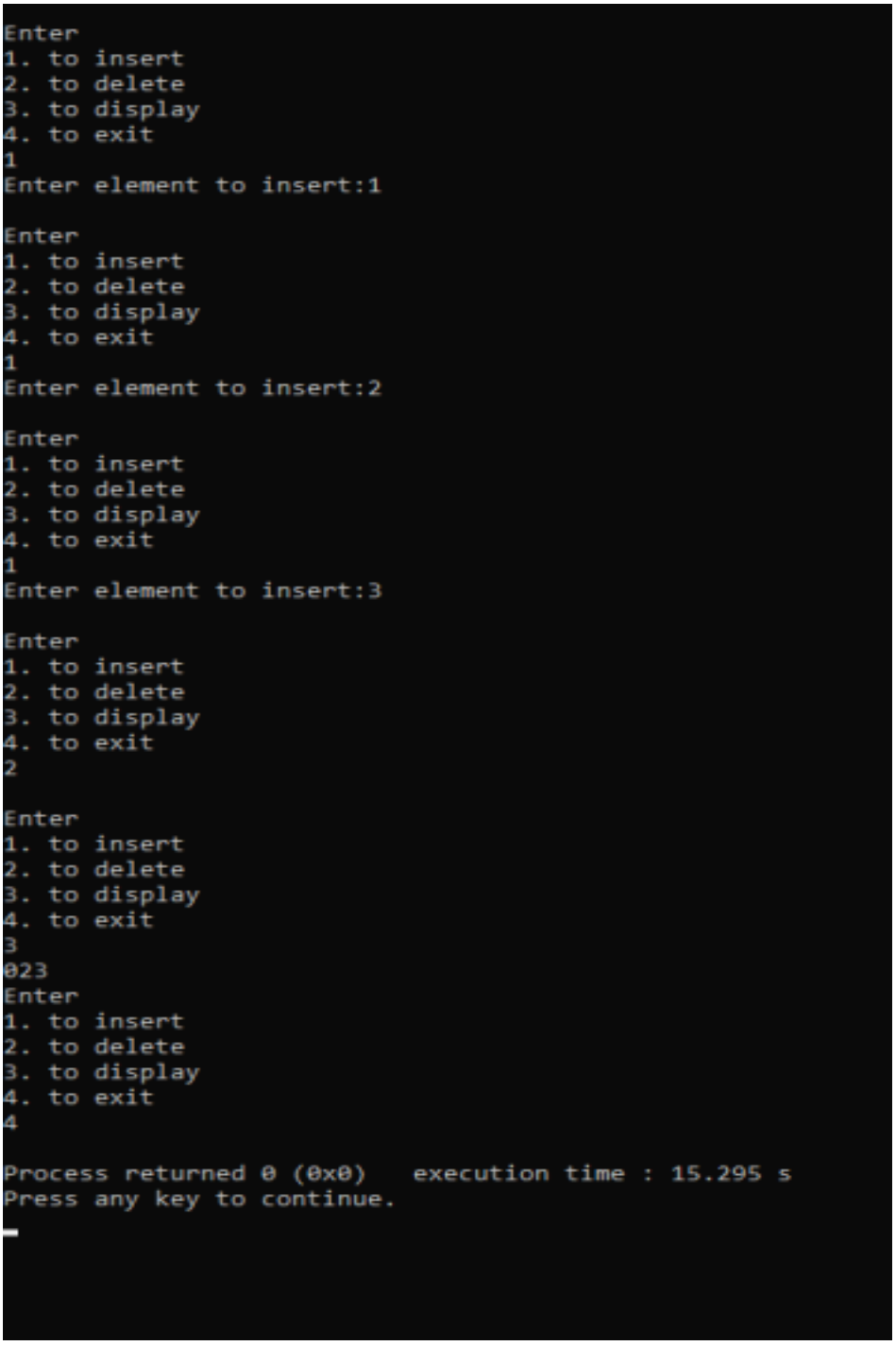
3a) WAP to simulate the working of a queue of integers using an array.Provide the following operations: Insert, Delete, Display. The program should print appropriate messages for queue empty and queue overflow conditions

```
#include <stdio.h>

int front=-1, rear=-1, Q[6];

void insert(int x){
   if(rear == 5){
      printf("Queue is full!\n");
   }
   else{
   rear++;
      Q[rear] = x;


   }
}


void delete(){
   if(front == rear){
      printf("Queue is empty\n");


   }
   else{
      for(int i = 0;i<rear;i++){
         Q[i] = Q[i+1];
      }
      rear--;
   }
}
```

```
void display(){
   if(front==rear){
      printf("Queue is empty\n");


   }
   else{
      for(int i = front;i<= rear;i++){
         printf("%d",Q[i]);
      }
   }
}


Int main()
{
   int n=0,ele,e=0;
   while(n!=4){
      printf("\nEnter\n1. to insert\n2. to delete\n3. to display\n4. to exit\n");
      scanf("%d",&n);
      switch(n){
         case 1:
            printf("Enter element to insert:");
            scanf("%d",&ele);
            insert(ele);
            break;
         case 2:
            delete();
            break;
         case 3:
            display();
            break;
```

```
    }
  }
  return 0;
}
```

Ouput



3b) WAP to simulate the working of a circular queue of integers using an array. Provide the following operations: Insert, Delete & Display.
The program should print appropriate messages for queue

empty and queue overflow conditions

```
#include <stdio.h>


int front=-1, rear=-1, Q[6];
void insert(int x){
   if((rear+1)%6 == 5){
      printf("Queue is full!\n");
   }
   else{
      rear = (rear + 1) %6;
      Q[rear] = x;
   }
}
void delete(){
   if(front == (rear+1)%6){
      printf("Queue is empty\n");
   }
   else{
      for(int i = 0;i<(rear+1)%6;i++){
         Q[i] = Q[i+1];
      }
      rear--;
   }
}
void display(){
   if(front == (rear+1)%6){
```

```
        printf("Queue is empty\n");
    }
    else{
        for(int i = front;i<= (rear+1)%6;i++){
            printf("%d",Q[i]);
        }
    }
}
void main()
{
    int n=0,ele;
    while(n!=4){
        printf("\nEnter\n1. to insert\n2. to delete\n3. to display\n4. to exit\n");
        scanf("%d",&n);
        switch(n){
            case 1:
                printf("Enter element to insert:");
                scanf("%d",&ele);
                insert(ele);
                break;
            case 2:
                delete();
                break;
            case 3:
                display();
                break;
        }
    }
```

Output

```
Enter
1. to insert
2. to delete
3. to display
4. to exit
1
Enter element to insert:1

Enter
1. to insert
2. to delete
3. to display
4. to exit
1
Enter element to insert:2

Enter
1. to insert
2. to delete
3. to display
4. to exit
1
Enter element to insert:3

Enter
1. to insert
2. to delete
3. to display
4. to exit
2

Enter
1. to insert
2. to delete
3. to display
4. to exit
3
0230
Enter
1. to insert
2. to delete
3. to display
4. to exit
4

Process returned 4 (0x4)   execution time : 13.126 s
Press any key to continue.
```

Lab program 4:

WAP to Implement Singly Linked List with following operations

a) Create a linked list.
b) Insertion of a node at first position, at any position and at end of list.
Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct Node {
    int data;
    struct Node* next;
} Node;

Node* head = NULL;

void push();
void append();
void insert();
void display();

int main() {
    int choice;
    while (1) {
        printf("1. Insert at beginning\n");
        printf("2. Insert at end\n");
        printf("3. Insert at position\n");
        printf("4. Display\n");
        printf("5. Exit\n");
        printf("Enter choice: ");
        scanf("%d", &choice);
        switch (choice) {
            case 1:
                push();
                break;
            case 2:
                append();
                break;
            case 3:
                insert();
                break;
            case 4:
                display();
                break;
            default:
```

```
            printf("Exiting the program");
            return 0;
        }
    }
}

void push() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data;
    printf("Enter data in the new node:
    "); scanf("%d", &new_data);
    temp->data = new_data;
    temp->next = head;
    head = temp;
}

void append() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data;
    printf("Enter data in the new node:
    "); scanf("%d", &new_data);
    temp->data = new_data;
    temp->next = NULL;
    if (head == NULL) {
        head = temp;
        return;
    }
    Node* temp1 = head;
    while (temp1->next != NULL) {
        temp1 = temp1->next;
    }
    temp1->next = temp;
}

void insert() {
    Node* temp = (Node*)malloc(sizeof(Node));
    int new_data, pos;
    printf("Enter data in the new node:
    "); scanf("%d", &new_data);
    printf("Enter position of the new node:
    "); scanf("%d", &pos);
    temp->data = new_data;
    temp->next = NULL;
    if (pos == 0) {
        temp->next = head;
        head = temp;
        return;
    }
    Node* temp1 = head;
    while (pos--) {
```

```
        temp1 = temp1->next;
}
Node* temp2 = temp1->next;
temp->next = temp2;
temp1->next = temp;
}

void display() {
Node* temp1 = head;
while (temp1 != NULL) {
printf("%d -> ", temp1->data);
temp1 = temp1->next;
}
printf("NULL\n");
}
```

Output

```
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 10
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 1
Enter data in the new node: 20
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 2
Enter data in the new node: 30
1. Insert at beginning
2. Insert at end
3. Insert at position
4. Display
5. Exit
Enter choice: 3
Enter data in the new node: 2
Enter position of the new node: 40

Process returned -1073741819 (0xC0000005)   execution time : 29.477 s
Press any key to continue.
```

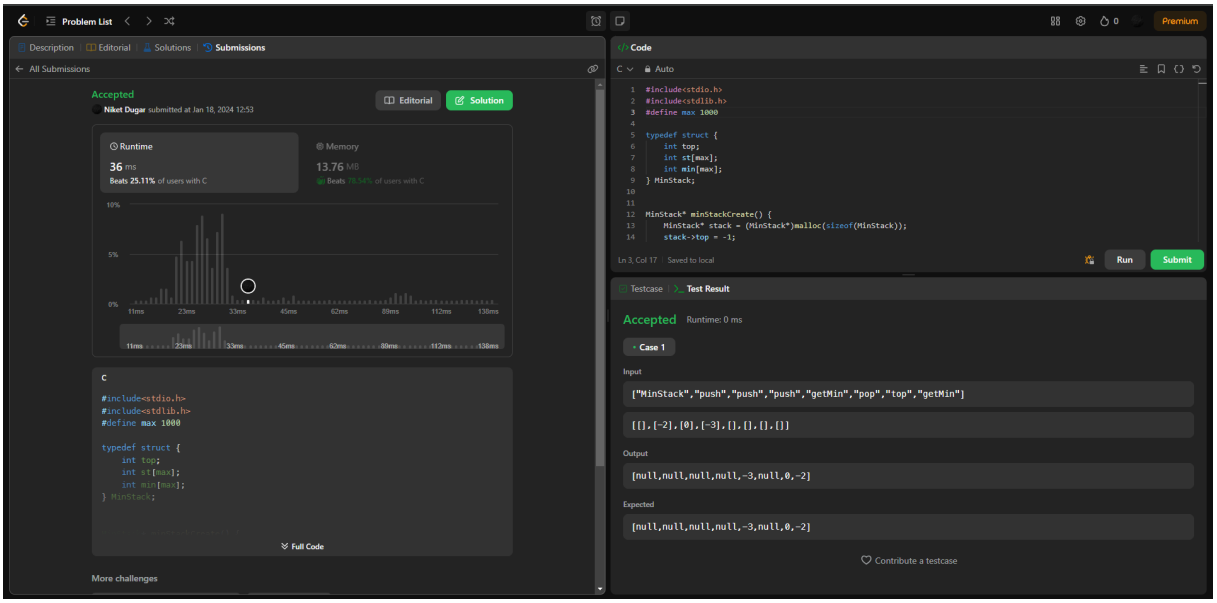
Leetcode

```
#include<stdio.h>
#include<stdlib.h>
#define max 1000

typedef struct {
    int top;
    int st[max];
    int min[max];
} MinStack;
MinStack* minStackCreate() {
    MinStack* stack = (MinStack*)malloc(sizeof(MinStack));
    stack->top = -1;
    return stack;}
void minStackPush(MinStack* obj, int val) {
    if(obj->top == max-1){
        printf("Stack Full\n");
        return;
    }
    obj->st[++obj->top] = val;
    if(obj->top > 0)
    {
        if(obj->min[obj->top - 1] < val)
            obj->min[obj->top] = obj->min[obj->top -1];
        else
            obj->min[obj->top] = val;
    }
    else
        obj->min[obj->top] = val;
```

```
}
void minStackPop(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return;
    }
    else {
        obj->top -= 1;
    }
}
int minStackTop(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("Stack empty\n");
        return -1;
    }
    return obj->st[obj->top];
}
int minStackGetMin(MinStack* obj) {
    if(obj->top == -1)
    {
        printf("min Stack empty\n");
        return -1;
    }
    return obj->min[obj->top];
}
void minStackFree(MinStack* obj) {
```

```
    free(obj);
}
```

Output



Lab program 5:
WAP to Implement Singly Linked List with following operations
a) Create a linked list.
b) Deletion of first element, specified element and last element in the list.
c) Display the contents of the linked list.

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node* next;
}node;
struct node* head = NULL;

void create(int A[], int n){
   struct node *t, *last;
   head = (struct node*)malloc(sizeof(struct node));
   head->data = A[0];
   head->next = NULL;
   last = head;
   for(int i=1; i<n; i++){
      t = (struct node*)malloc(sizeof(struct node));
      t->data = A[i];
      t->next = NULL;
      last->next = t;
      last = t;
   }
}

void fdelete(){
   struct node* p;
   p = (struct node*)malloc(sizeof(struct node));
   p = head;
   head = head->next;
   free(p);
}

void ldelete(){
   struct node*p, *p1;
   p = (struct node*)malloc(sizeof(struct node));
   p = head;
   while(p->next!=NULL){
      p1 = p;
      p = p->next;
   }
```

```
    p1->next = NULL;
    free(p);
}

void delete(int pos){
    struct node *p, *p1;
    p = (struct node*)malloc(sizeof(struct node));
    p = head;
    for(int i=0;i<pos-1;i++){
        p = p->next;
    }
    p1 = p;
    p = p->next;
    p1->next = p->next;
    free(p);

}

void display(struct node*p){
    struct node *t = p;
    while(t!=NULL){
        printf("%d ",t->data);
        t = t->next;
    }
}

int main()
{
    int A[] = {2,4,6,5,1,8};
    create(A,6);
    fdelete();
    ldelete();
    delete(2);
    display(head);
    return 0;
}
```

Output

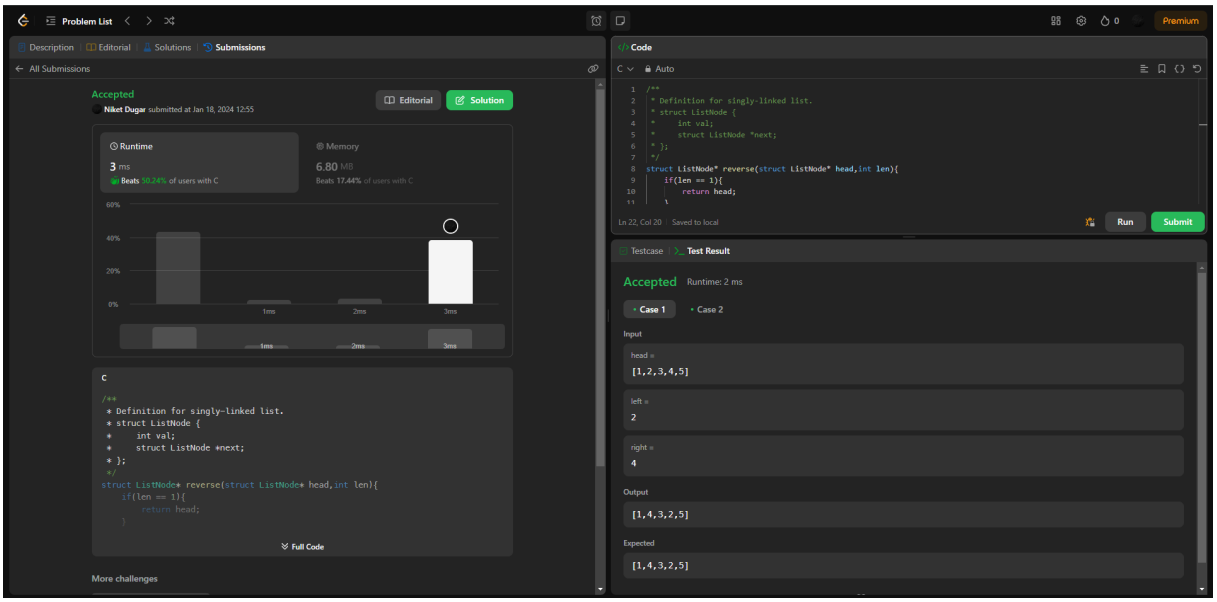
```
4 6 1
Process returned 0 (0x0)   execution time : 1.513 s
Press any key to continue.
```

LeetCode

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* reverse(struct ListNode* head,int len){
    if(len == 1){
        return head;
    }

    int count = 0;
    struct ListNode* p = head;
    while(count < len-1){
        p = p->next;
        count++;
    }
    struct ListNode* pEnd = p->next;

    struct ListNode* pPre = head;
    p = head->next;
    struct ListNode* pNext;

    count = 0;
    while(count < len-1){
        pNext = p->next;
        p->next = pPre;
        pPre = p;
        p = pNext;
        count++;
    }
    head->next = pEnd;

    return pPre;
}
struct ListNode* reverseBetween(struct ListNode* head, int
left, int right) {
    struct ListNode* p = head;struct ListNode* pPre = NULL;
    int count = 1;
    while(count < left){
        pPre = p;
        p = p->next;
        count++;
    }
    if(pPre){
        pPre->next = reverse(p,right-left+1);
    }
```

```
    else{
        head = reverse(p,right-left+1);
    }

    return head;
}
```

Output



Lab program 6:
6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists.

```
#include <stdio.h>
#include <stdlib.h>
struct node{
   int data;
   struct node* next;
};
struct node* head = NULL;

void create(int A[], int n){
   struct node *t, *last;
   head = (struct node*)malloc(sizeof(struct node));
   head->data = A[0];
   head->next = NULL;
   last = head;
   for(int i=1; i<n; i++){
      t = (struct node*)malloc(sizeof(struct node));
      t->data = A[i];
      t->next = NULL;
      last->next = t;
      last = t;
   }
}

struct node* sort(head) {
   struct node *ptr1,*ptr2;
   ptr1=head;
   int temp,count=1;
   while(ptr1->next!=NULL){
      count++;
      ptr1=ptr1->next;
   }
   ptr1=head;
   while(--count){
      ptr1=head;
      while(ptr1->next!=NULL){
         ptr2=ptr1;
         ptr1=ptr1->next;
         if(ptr1->data<ptr2->data){
            temp=ptr2->data;
            ptr2->data=ptr1->data;
            ptr1->data=temp;
         }
      }
```

```
    }
    return head;
}

void main(){
    int A[]={2,1,3,5,4,8,6,7};
    create(A,8);
    head = sort(head);
    struct node* p = head;
    while(p!=NULL){
        printf("%d",p->data);
        p = p->next;
    }
}


#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node* next;
};
struct node* head = NULL;

void create(int A[], int n){
    struct node *t, *last;
    head = (struct node*)malloc(sizeof(struct node));
    head->data = A[0];
    head->next = NULL;
    last = head;
    for(int i=1; i<n; i++){
        t = (struct node*)malloc(sizeof(struct node));
        t->data = A[i];
        t->next = NULL;
        last->next = t;
        last = t;
    }
}

struct node* reverse(struct node* p, int len){
    p = head;
    int a[len];
    a[0] = p->data;
    for(int i=1; i<len; i++){
        p = p->next;
        a[i] = p->data;
    }
    p = head;
    for(int i = len-1; i>=0; i--){
        p->data = a[i];
```

```
        p = p->next;
    }
    return head;
}

void main(){
    int A[]={2,1,3,5,4,8,6,7};
    create(A,8);
    head = reverse(head,8);
    struct node* p = head;
    while(p!=NULL){
            printf("%d ",p->data);
            p = p->next;
    }
}


#include <stdio.h>
#include <stdlib.h>
struct node{
    int data;
    struct node* next;
};
struct node* head = NULL, *head1 = NULL;
void create()
{
    head = (struct node*)malloc(sizeof(struct node));
    head1 = (struct node*)malloc(sizeof(struct node));
   struct node *last, *last1;
   int a[] = {1,2,3,4};
   int b[] = {5,6,7,8};

   head->data = a[0];
   head->next = NULL;

   head1->data = b[0];
   head1->next = NULL;

   last = head;
   last1 = head1;

   for(int i=1;i<sizeof(a)/sizeof(a[0]);i++)
   {
      struct node *t;
      t = (struct node*)malloc(sizeof(struct node));
      t->data = a[i];
      t->next = NULL;
      last->next = t;
      last = t;
   }
```

```
  for(int i=1;i<sizeof(b)/sizeof(b[0]);i++)
  {
     struct node *t1;
     t1 = (struct node*)malloc(sizeof(struct node));
     t1->data = b[i];
     t1->next = NULL;
     last1->next = t1;
     last1 = t1;
  }

}

void display_a()
{
  struct node *n = head;

  printf("A:\n");
  while(n!=NULL)
  {
     printf("%d\n",n->data);
     n = n->next;
  }
  return;
}

void display_b()
{
  struct node *n1 = head1;

  printf("B:\n");
  while(n1!=NULL)
  {
     printf("%d\n",n1->data);
     n1 = n1->next;
  }
}

void concat()
{
  struct node *n, *n1;
  if(head == NULL || head1 == NULL)
  {
     if(head == NULL)
        display_b();
     else
        display_a();
  }
  else
  {
```

```
        n=head;
        while(n!=NULL)
        {
           n1 = n;
           n = n->next;
        }
        n1->next = head1;
        display_a();
    }

}
void main()
{
    create();
    printf("After Concatenation A and B\n");
    concat();
}
```

Output

```
12345678
Process returned 0 (0x0)   execution time : 1.179 s
Press any key to continue.
```

```
7 6 8 4 5 3 1 2
Process returned 0 (0x0)   execution time : 0.947 s
Press any key to continue.
```

```
After Concatenation A and B
A:
1 2 3 4 5 6 7 8
Process returned 0 (0x0)   execution time : 0.841 s
Press any key to continue.
```

6b) WAP to Implement Single Link List to simulate Stack & Queue Operations.

```
#include<stdio.h>
#include<stdlib.h>
void push();
void pop();
void display();

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

struct node *head = NULL;

void push()
{
   int n;
   printf("Enter the insert element\n");
   scanf("%d",&n);

   struct node *new_node = (struct
node*)malloc(sizeof(struct node));
   new_node -> data = n;
   new_node->next = NULL;

   if(head == NULL)
      head = new_node;
   else
   {   struct node *p;
      p = head;
      while(p->next != NULL)
      {
         p = p->next;
      }
      p->next = new_node;
   }
   return;
}

void pop()
{
   struct node *ptr,*p2;
   if(head == NULL)
   {
      printf("List is empty\n");
      exit(0);
```

```
    }
    else
    {   if(head->next == NULL)
        {
            printf("Element %d deleted\n",head->data);
            free(head);
            head = NULL;
        }
        else
        {
            ptr = head;
            while(ptr->next != NULL)
            {
                p2 = ptr;
                ptr = ptr->next;
            }

            printf("Element %d deleted\n",ptr->data);
            p2->next = NULL;
            free(ptr);
        }
        return;
    }
}

void display()
{
    struct node *n;

    if(head == NULL)
    {
        printf("List is empty");
        exit(0);
    }

    else{
    n = head;
    while(n != NULL)
        {
            printf("%d\n",n->data);
            n = n->next;
        }
    }
}

void main()
{
    int ch;
    printf("Stack Implementation using linked list\n\n");
    while(1){
```

```
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice:\n");
        scanf("%d",&ch);
        switch(ch)
        {
           case 1:
              push();
              break;
           case 2:
              pop();
              break;
           case 3:
              display();
              break;
           case 4:
              printf("Program ends successfully!");
              exit(0);
           default:
              printf("Enter a valid number...\n");
        }
        continue;
    }
}

#include<stdio.h>
#include<stdlib.h>
void enqueue();
void dequeue();
void display();

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

struct node *head = NULL;

void enqueue()
{
    int n;
    printf("Enter the insert element\n");
    scanf("%d",&n);

    struct node *new_node =(struct node*)malloc(sizeof(struct
node));
    new_node->data = n;
    new_node->next = head;
```

```
   head = new_node;
}
void dequeue()
{
   if(head == NULL)
   {
      printf("List is empty");
      exit(0);
   }
   else
   {
      if(head->next == NULL)
      {
         printf("Element %d deleted\n",head->data);
         free(head);
         head = NULL;
      }
      else
      {
         struct node *p,*p1;
         p = head;
         while(p->next != NULL)
         {
            p1 = p;
            p = p->next;
         }
         p1->next = NULL;
         printf("Element %d deleted\n",p->data);
         free(p);
      }
   }
}

void display()
{
   struct node *n;

   if(head == NULL)
   {
      printf("List is empty");
      exit(0);
   }

   else{
   n = head;
   while(n != NULL)
      {
         printf("%d\n",n->data);
         n = n->next;
      }
```

```
    }
}

void main()
{
    int ch;
    printf("Queue Implementation using linked list\n\n");
    while(1){
        printf("1. Insert\n");
        printf("2. Delete\n");
        printf("3. Display\n");
        printf("4. Exit\n");
        printf("Enter your choice:\n");
        scanf("%d",&ch);
        switch(ch)
        {
            case 1:
                enqueue();
                break;
            case 2:
                dequeue();
                break;
            case 3:
                display();
                break;
            case 4:
                printf("Program ends successfully!");
                exit(0);
            default:
                printf("Enter a valid number...\n");
        }
        continue;
    }
}
```

Output

```
Stack Implementation using linked list

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the insert element
10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the insert element
20
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
2
Element 20 deleted
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:

3
10
```

```
Queue Implementation using linked list

1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
1
Enter the insert element
10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
2
Element 10 deleted
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:
3
List is empty
Process returned 0 (0x0)   execution time : 7.853 s
Press any key to continue.
```

Lab program 7:
WAP to Implement doubly link list with primitive operations

a) Create a doubly linked list.
b) Insert a new node to the left of the node.
c) Delete the node based on a specific value
d) Display the contents of the list

```
#include<stdio.h>
#include<stdlib.h>
typedef struct Node{
   int val;
   struct Node *prev;
   struct Node *next;
}Node;

Node *head=NULL;

void insert(){
   int num,pos;
   printf("Enter value : ");
   scanf("%d",&num);
   printf("Enter node to insert left of: ");
   scanf("%d",&pos);

   Node *ptr=(Node*)malloc(sizeof(Node));
   ptr->val=num;

   if(pos==0){
   ptr->next=head;
   ptr->prev=NULL;
      if (head != NULL){
         head->prev = ptr;
      }
   head=ptr;

   }
   Node *ptr1=head;

   if(pos!=0){
      for(int i=0;i<pos;i++){
         ptr1=ptr1->next;
      }
      ptr->next=ptr1;
      ptr->prev=ptr1->prev;
      ptr1->prev->next=ptr;
      ptr1->prev=ptr;
```

```
    }
}

void delete(){
    printf("Enter value to delete: ");
    int loc=-1,len=1,val;
    scanf("%d",&val);
    Node *ptr=head,*ptr2;
    while(ptr->next!=NULL){
        len++;
        ptr=ptr->next;
    }
    ptr=head;
    for(int i=0;i<len;i++){
        if(ptr->val=val){
            loc=i;
        }
    }

    if(loc==-1){
        printf("Delete element not in list\n");
        return;
    }
    if(loc==0){
        printf("Deleted element: %d\n",head->val);
        ptr=head;
        head=head->next;
        free(ptr);
        return;
    }
    if(loc==len){
        ptr = head;
        while (ptr->next != NULL) {
            ptr = ptr->next;
        }
        printf("Deleted element: %d\n", ptr->val);
        ptr->prev->next = NULL;
        free(ptr);
        return;
    }
    ptr=head;
    for(int i=0;i<loc-1;i++){
        ptr2=ptr;
        ptr=ptr->next;
    }
    printf("Deleted element: %d\n",ptr->val);
    ptr2->next=ptr->next;
    ptr->next->prev=ptr2;
    free(ptr);
```

```
}

void display(){
   Node *ptr=head;
   while(ptr!=NULL){
      printf("%d<->",ptr->val);
      ptr=ptr->next;
   }
   printf("NULL\n");
}

void main(){
  int choice;
   printf("1. To insert into left of Doubly Linked List\n");
   printf("2. To Delete from any point of Doubly Linked
List\n");
   printf("Enter choice: ");
   while (1) {
      scanf("%d", &choice);
      switch (choice) {
         case 1:
            insert();
            break;
         case 2:
            delete();
            break;
         case 3:
            display();
            break;
         default:
            printf("Exiting the program");
            return;
      }
      printf("Enter choice: ");
   }
}
```

Output

```
1. To insert into left of Doubly Linked List
2. To Delete from any point of Doubly Linked List
Enter choice: 1
Enter value : 10
Enter node to insert left of: 0
Enter choice: 1
Enter value : 20
Enter node to insert left of: 0
Enter choice: 2
Enter value to delete: 10
Deleted element: 10
```

Lab program 8:
Write a program

a) To construct a binary Search tree.
b) To traverse the tree using all the methods i.e.,
in-order,
preorder and post order
c) To display the elements in the tree.

```
#include <stdio.h>
#include <stdlib.h>

typedef struct tree{
   struct tree *left;
   int data;
   struct tree *right;
}tree;
tree* root = NULL;

tree* insert(tree* t, int val) {
   if (t == NULL) {
      tree* pt = (tree*)malloc(sizeof(tree));
      pt->data = val;
      pt->left = pt->right = NULL;
      return pt;
   }

   if (t->data > val) {
      t->left = insert(t->left, val);
   } else if (t->data < val) {
      t->right = insert(t->right, val);
   }

   return t;
}


void inorder(tree* t){
   if(t==NULL){
      return;
   }
   inorder(t->left);
   printf("%d ",t->data);
   inorder(t->right);
```

```
}

void preorder(tree* t){
   if(t==NULL){
      return;
   }
   printf("%d ",t->data);
   preorder(t->left);
   preorder(t->right);

}

void postorder(tree* t){
   if(t==NULL){
      return;
   }
   postorder(t->left);
   postorder(t->right);
   printf("%d ",t->data);

}
void main(){
   int A[] = {8,1,5,3,9,4,6,7};
   root = insert(root,A[0]);
   for(int i=1; i<8; i++){
      root = insert(root,A[i]);
   }
   printf("Inorder traversal\n");
   inorder(root);
   printf("\nPreorder traversal\n");
   preorder(root);
   printf("\nPostorder traversal\n");
   postorder(root);
}
```

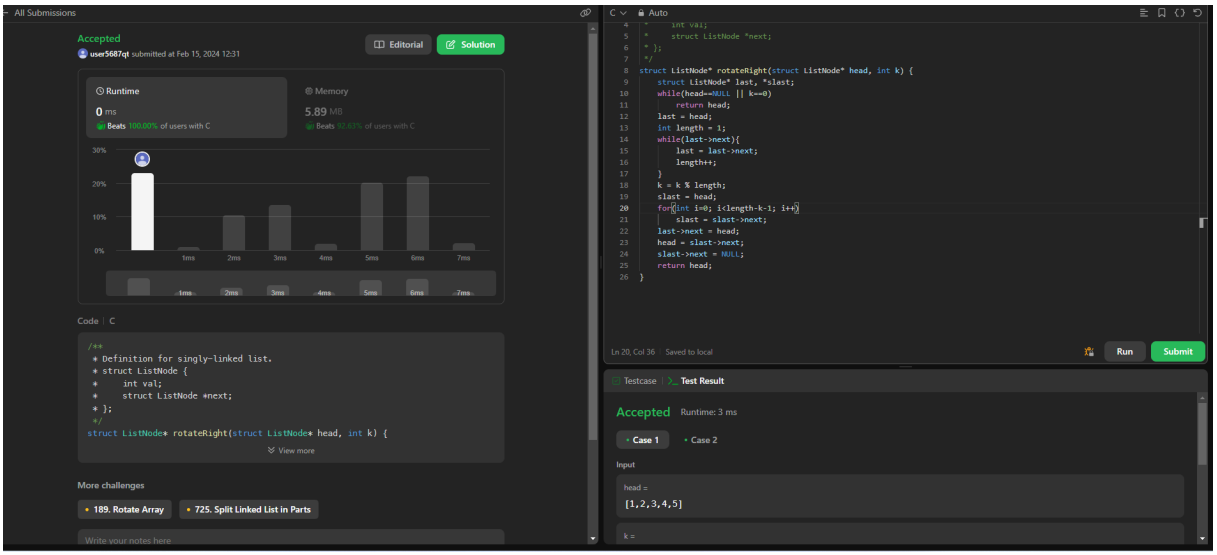
Output

```
Inorder traversal
1 3 4 5 6 7 8 9
Preorder traversal
8 1 5 3 4 6 7 9
Postorder traversal
4 3 7 6 5 1 9 8
Process returned 2 (0x2)   execution time : 1.515 s
Press any key to continue.
```

Leetcode

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* rotateRight(struct ListNode* head, int k) {
    struct ListNode* last, *slast;
    while(head==NULL || k==0)
        return head;
    last = head;
    int length = 1;
    while(last->next){
        last = last->next;
        length++;
    }
    k = k % length;
    slast = head;
    for(int i=0; i<length-k-1; i++)
        slast = slast->next;
    last->next = head;
    head = slast->next;
    slast->next = NULL;
    return head;
}
```

Output



Lab program 9:
9a) Write a program to traverse a graph using BFS method.
9b) Write a program to check whether given graph is connected or not using DFS method.

a)

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define MAX_VERTICES 50

typedef struct Graph_t {
        int V;
        bool adj[MAX_VERTICES][MAX_VERTICES];
} Graph;
Graph* Graph_create(int V)
{
        Graph* g = malloc(sizeof(Graph));
        g->V = V;

        for (int i = 0; i < V; i++) {
                for (int j = 0; j < V; j++) {
                        g->adj[i][j] = false;
                }
        }
        return g;
}

void Graph_addEdge(Graph* g, int v, int w)
{
        g->adj[v][w] = true;
        g->adj[w][v] = true;
}
void Graph_BFS(Graph* g, int s)
{
        bool visited[MAX_VERTICES];
        for (int i = 0; i < g->V; i++) {
                visited[i] = false;
        }
        int queue[MAX_VERTICES];
        int front = 0, rear = 0;
        visited[s] = true;
        queue[rear++] = s;

        while (front != rear) {
                s = queue[front++];
```

```
                printf("%d ", s);
                for (int adjacent = 0; adjacent < g->V;
                        adjacent++) {
                        if (g->adj[s][adjacent] &&
!visited[adjacent]) {
                                visited[adjacent] = true;
                                queue[rear++] = adjacent;
                        }
                }
        }
}

int main()
{

        Graph* g = Graph_create(4);
        Graph_addEdge(g, 0, 1);
        Graph_addEdge(g, 0, 2);
        Graph_addEdge(g, 1, 2);
        Graph_addEdge(g, 2, 0);
        Graph_addEdge(g, 2, 3);
        Graph_addEdge(g, 3, 3);

        printf("Following is Breadth First Traversal "
                "(starting from vertex 2) \n");
        Graph_BFS(g, 2);

        return 0;
}
```

b)

```
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>

#define N 100000

int g[N][N];
bool vis[N];

void Add_edge(int u, int v){
   g[u][v] = true;
}

bool is_connected(int n){
   for(int i=1; i<=n; i++){
      vis[i] = false;
      dfs(i,n);
   }
```

```
    for(int i=1; i<=n; i++){
        if(!vis[i])
            return false;
    }
    return true;
}

void dfs(int x, int n){
    vis[x]= true;
    for (int i = 1; i <= n; i++)
        if (g[x][i] && !vis[i])
            dfs(i, n);
}

void main()
{
    int n = 4;
    for (int i = 1; i <= n; i++)
        for (int j = 1; j <= n; j++)
            g[i][j] = 0;
    Add_edge(1, 2);
    Add_edge(2, 3);
    Add_edge(3, 4);
    if (is_connected(n))
        printf("Yes");
    else
        printf("No");
}
```

Output

```
Following is Breadth First Traversal (starting from vertex 2)
2 0 1 3
Process returned 0 (0x0)   execution time : 1.650 s
Press any key to continue.
```

```
Yes
Process returned 3 (0x3)   execution time : 1.112 s
Press any key to continue.
```

Lab program 10:
Given a File of N employee records with a set K of Keys(4-digit) which uniquely determine the records in file F. Assume that file F is maintained in memory by a Hash Table (HT) of m memory locations with L as the set of memory addresses (2-digit) of locations in HT.
Let the keys in K and addresses in L are integers. Design and develop a Program in C that uses Hash function H: K > L as H(K)=K mod m (remainder method), and implement hashing technique to map a given key K to the address space L. Resolve the collision (if any) using linear probing.

```
#include <stdio.h>
#define size 10

int H[size];

void insert(int val){
   int i = 0;
   int hkey = (val + i) % size;
   while (H[hkey] != -1) {
      i++;
      hkey = (val + i) % size;
   }
   H[hkey] = val;
}
void search(int v){
   for(int i=0; i<size; i++){
      if(H[i]==v){
         printf("Employee %d is found at position %d\n\n",
v,i);
      }
   }
}
int main(){
   int val[size] = {1345, 2347, 4642, 9871, 9855, 2638, 9853,
2986, 9874, 5530};
   for (int j = 0; j < size; j++) {
      H[j] = -1;
   }
   for (int i = 0; i < size; i++) {
      insert(val[i]);
   }
```

```
    int x;
    printf("Enter employee you want to find:");
    scanf("%d",&x);
    search(x);
    for (int i = 0; i < size; i++) {
        printf("%d %d \n",i, H[i]);
    }
    return 0;
}
```

Output

```
Enter employee you want to find:9871
Employee 9871 is found at position 1

0 5530
1 9871
2 4642
3 9853
4 9874
5 1345
6 9855
7 2347
8 2638
9 2986

Process returned 0 (0x0)   execution time : 7.371 s
Press any key to continue.
```