VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belgaum -590014, Karnataka.

LAB REPORT
On

DATA STRUCTURES (23CS3PCDST)

Submitted by

AYUSH GIRISH GAONKAR (1BM22CS063)

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 **AYUSH GIRISH GAONKAR (1BM22CS063)**, who is bonafide student of **B. M. S. College of Engineering**. It is in partial fulfilment 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. Namratha M
Assistant Professor
Department of CSE
BMSCE, Bengaluru

Dr. Jyothi S Nayak
Professor and Head
Department of CSE
BMSCE, Bengaluru

Index Sheet

Sl. No.	Experiment Title	Page No.
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**.**	4
2	Write a program to implement linear search in C using pass by parameters. Write a program to implement linear search in C using pass by parameters.	7
3	1. Create a user defined structure to store and retrieve student information such as name, date of birth, USN and marks. Also calculate the total marks of all the 3 subjects. 2. Create a user defined function to perform binary search on an array of characters. Use pass by parameters only. 3. Create a user defined function to add pointer variables and store and display the result.	9
4	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).	13
5	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. 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.	17
6	Hacker rank	17
7	Leetcode	26
8	Lab program 8: 1)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. 2)WAP to Implement Singly Linked List with following operations	34

	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.	
9	6a) WAP to Implement Single Link List with following operations: Sort the linked list, Reverse the linked list, Concatenation of two linked lists. 6b) WAP to Implement Single Link List to simulate Stack and Queue Operations. 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 Leetcode	42
10	Given a Linked List and a number, write a function that returns the value at the Nth node from the end of the Linked List. Program 2: Write a function that takes a list sorted in non-decreasing order and deletes any duplicate nodes from the list. The list should only be traversed once. For example if the linked list is 11->11->11->21->43->43->60 then removeDuplicates() should convert the list to 11->21->43->60. 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. 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.	48

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 size 6

void push(int);
void pop();
void display();

int stack[size], top = -1;

// Function to push element onto stack
void push(int item) {
   if(top == size - 1) {
      printf("Stack overflow\n");
   }
   else {
      stack[++top] = item;
      printf("Insertion success\n");
   }
}

// Function to pop element from stack
void pop() {
   if(top == -1) {
      printf("Stack underflow\n");
   }
   else {
      printf("The deleted elem is %d\n", stack[top]);
      top--;
   }
}

// Function to display elements of stack
void display() {
   if(top == -1) {
      printf("Stack is empty\n");
   }
   else {
      printf("The elements in the stack are:\n");
      for(int i = top; i >= 0; i--) {
         printf("%d\n", stack[i]);
      }
```

```
    }
}

int main() {
    int item, choice;
printf(“AYUSH-1BM22CS063”);
    while(1) {
        printf("Menu:\n");
        printf("1. push\n2. pop\n3. display\n4. exit\n");
        printf("Enter your choice:");
        scanf("%d", &choice);
        switch(choice) {
            case 1:
                printf("Enter the value to be inserted:");
                scanf("%d", &item);
                push(item);
                break;
            case 2:
                pop();
                break;
            case 3:
                display();
                break;
            case 4:
                exit(0);
                break;
            default:
                printf("Invalid choice\n");
        }
    }    return 0;
}
```

Output:

```
AYUSH-1BM22CS063Menu:
1. push
2. pop
3. display
4. exit
Enter your choice:1
Enter the value to be inserted:10
Insertion success
Menu:
1. push
2. pop
3. display
4. exit
Enter your choice:1
Enter the value to be inserted:20
Insertion success
Menu:
1. push
2. pop
3. display
4. exit
Enter your choice:1
Enter the value to be inserted:30
Insertion success
Menu:
1. push
2. pop
3. display
4. exit
```

```
Insertion success
Menu:
1. push
2. pop
3. display
4. exit
Enter your choice:2
The deleted elem is 30
Menu:
1. push
2. pop
3. display
4. exit
Enter your choice:2
The deleted elem is 20
Menu:
1. push
2. pop
3. display
4. exit
Enter your choice:3
The elements in the stack are:
10
Menu:
1. push
2. pop
3. display
4. exit
Enter your choice:4
```

Lab program 2:

Write a program to implement linear search in C using pass by parameters.

```
#include<stdio.h>
int linearsearch(int arr[],int n,int key){
    for(int i=0;i<n;i++)
    {
        if(arr[i]==key)
        {
            return i;
        }
     }
     return -1;
}
int main()
{
int arr[100],key,n,i,index;
printf(“ayush-1BM22CS063”);
    printf("enter the number of elements:\n");
    scanf("%d",&n);
    printf("enter the elements:\n");
    for(i=0;i<n;i++)
    {
        scanf("%d",&arr[i]);
    }
    printf("enter the key element to be found in the array:\n");
    scanf("%d",&key);
    index=linearsearch(arr,n,key);
    if(index!=-1)
    {
```

```
        printf("element %d is found at %d\n",key,index);
     }
    else
    {
        printf("element %d not found\n",key);
    }
    return 0;
}
```

Output:

```
ayush-1BM22CS063
enter the number of elements:
3
enter the elements:
1
2
3
enter the key element to be found in the array:
3
element 3 is found at 2
```

Lab program 3:

1. Create a user defined structure to store and retrieve student information such as name, date of birth, USN and marks. Also calculate the total marks of all the 3 subjects

2. Create a user defined function to perform binary search on an array of characters. Use pass by parameters only.

3. Create a user defined function to add pointer variables and store and display the result.

1)

```
#include<stdio.h>
typedef struct Student{
    char name[50];
   char dob[20];
   char usn[15];
   float marks[3];
} s1;

float calctotalmarks(s1 stud)
{
   int totalmarks=0;
   for(int i=0;i<3;i++)
   {
   totalmarks+=stud.marks[i];
   }
return totalmarks;
}
int main()
{   s1 stud;
   int n,i;
   printf("enter the no of students:\n");
   scanf("%d",&n);
```

```
    for(int i=0;i<n;i++)
       {
          printf("enter the student name:\n");
          scanf("%s",stud.name);
          printf("enter the date of birth:\n");
          scanf("%s",stud.dob);
          printf("enter the usn:\n");
          scanf("%s",stud.usn);
          printf("Enter marks for 3 subjects:\n");
          for (int i = 0; i < 3; i++) {
          printf("Enter marks for subject %d: ", i + 1);
          scanf("%f", &stud.marks[i]);
    }
    float total=calctotalmarks(stud);
    printf("\nStudent Information:\n");
    printf("Name: %s\n", stud.name);
    printf("Date of Birth: %s\n", stud.dob);
    printf("USN: %s\n", stud.usn);
    printf("Marks:\n");
    for (int i = 0; i < 3; i++) {
       printf("Subject %d: %.2f\n", i + 1, stud.marks[i]);
    }
    printf("Total Marks: %.2f\n", total);

    return 0;
}
}
```

Output:

```
enter the no of students:
1
enter the student name:
AYUSH
enter the date of birth:
13-12-2004
enter the usn:
1BM22CS063
Enter marks for 3 subjects:
Enter marks for subject 1: 43
Enter marks for subject 2: 50
Enter marks for subject 3: 45

Student Information:
Name: AYUSH
Date of Birth: 13-12-2004
USN: 1BM22CS063
Marks:
Subject 1: 43.00
Subject 2: 50.00
Subject 3: 45.00
Total Marks: 138.00
```

2)#include <stdio.h>

```
int binarySearch(char arr[], int n, char key) {
    int left = 0;
    int right = n - 1;

    while (left <= right) {
        int mid = left + (right - left) / 2;

        if (arr[mid] == key) {
            return mid; // Return the index if key is found
        } else if (arr[mid] < key) {
            left = mid + 1;
        } else {
            right = mid - 1;
        }
```

```
    }
return -1;
}

int main() {
    char arr[100];
    int n;
    char key;
    printf("Enter the number of characters in the array: ");
    scanf("%d", &n);

    printf("Enter %d characters in sorted order: ", n);
    for (int i = 0; i < n; i++) {
        scanf(" %c", &arr[i]);
    }
    printf("Enter the character to search: ");
    scanf(" %c", &key);
    int index = binarySearch(arr, n, key);
    if (index != -1) {
        printf("Character %c found at index %d\n", key, index);
    } else {
        printf("Character %c not found in the array\n", key);
    }

    return 0;
}
```

Output:

```
Enter the number of characters in the array: 5
Enter 5 characters in sorted order: a
d
e
x
y
Enter the character to search: x
Character x found at index 3
```

```
#include <stdio.h>
void addAndDisplay(int *a, int *b) {
    int result = *a + *b;
    printf("Result of addition: %d\n", result);
}
int main() {
    int num1 = 10, num2 = 20;
    addAndDisplay(&num1, &num2);
    return 0;
}
```

Output:

```
Result of addition: 30

Process returned 0 (0x0)   execution time : 0.247 s
Press any key to continue.
```

Lab program 4:

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<ctype.h>
#define stack_size 20
void push(int *top,char stack[],char ele)
{
    stack[++(*top)]=ele;
}
char pop(int *top,char stack[])
{
```

```
    return stack[(*top)--];
}
int prece(char a)
{
    if(a=='^')
    {
        return 3;
    }
    else if(a=='*' || a=='/')
    {
        return 2;
    }
    else if(a=='+'|| a=='-')
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
void intopo(char infix[],char postfix[])
{
    char ele;
    char stack[stack_size];
    int i=0,j=0,top=-1;
    while(infix[i]!='\0')
    {
        if(isalnum(infix[i]))
```

```
    {
        postfix[j]=infix[i];
        j++;
    }
    else if(infix[i]=='(')
    {
        push(&top,stack,infix[i]);
    }
    else if(infix[i]==')')
    {
        while(stack[top]!='(')
        {
            postfix[j]=pop(&top,stack);
            j++;
        }
        ele=pop(&top,stack);
    }
    else
    {
        while(prece(stack[top])>=prece(infix[i]))
        {
            postfix[j]=pop(&top,stack);
            j++;
        }
            push(&top,stack,infix[i]);
    }
     i++;
}
while(top!=-1)
```

```
    {
        postfix[j]=pop(&top,stack);
        j++;
    }
    postfix[j]='\0';
}
void main()
{
    char infix[20],postfix[20];
    printf("ayush-1BM22CS063");
    printf("enter the infix expression\n");
    scanf("%s",infix);
    intopo(infix,postfix);
    printf("the postfix expression is: %s\n",postfix);
}
```

Output:

```
ayush-1BM22CS063
enter the infix expression
a+b*c/t
the postfix expression is: abc*t/+
```

Lab program 5:

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.

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>
#include<process.h>
#define que_size 3
```

```
void enque(int que[],int *front,int *rear,int ele)
{
   if(*rear==que_size-1)
   {
      printf("\n que overflow");
   }
   else
   {
      que[++(*rear)]=ele;
   }
}
int deque(int que[],int *front,int *rear)
{
   int del_ele;
   if((*front)>*rear)
   {
      printf("\nstack underflow");
   }
   else
   {
      del_ele=que[(*front)++];
   }
   return del_ele;
}
void display(int que[],int *front,int *rear)
{
   if(*front>*rear)
   {
      printf("\nstack is empty");
```

```
    }
    else
    {
        for(int i=*front;i<*rear+1;i++)
        {
            printf("\n element is %d",que[i]);
        }
    }
}
void main()
{
    printf(“ayush-1BM22CS063\n”);
    int que[que_size],front=0,rear=-1,ele,ch,del_ele;
    do
    {
        printf("\n enter 1 for enque\n enter 2 for deque\n enter 3 for display\n");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
            {
                printf("\nenter the element to add:");
                scanf("%d",&ele);
                enque(que,&front,&rear,ele);
                printf("\nelement added to %d position",rear);
                break;
            }
        case 2:
            {
```

```
            del_ele=deque(que,&front,&rear);
            printf("\ndeleted element is %d at %d position",del_ele,front-1);
            break;
          }
        case 3:
          {
            display(que,&front,&rear);
            break;
          }

        default:
          {
            exit(0);
          }
       }
    }while(1);
}
```

Output:

```
ayush-1BM22CS063

 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

enter the element to add:10

element added to 0 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

enter the element to add:20

element added to 1 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

enter the element to add:30

element added to 2 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

enter the element to add:40

 que overflow
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
2

deleted element is 10 at 0 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
2

deleted element is 20 at 1 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
2

deleted element is 30 at 2 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
2

stack underflow
deleted element is 0 at 2 position
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
3

stack is empty
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
```

```
#include<stdio.h>
#include<process.h>
#define que_size 2
int count=0;
void enque(int que[],int *front,int *rear,int ele)
{
   if(count==que_size)
   {
      printf("\n queue overflow");
   }
   else
   {
      *rear=((*rear)+1)%que_size;
      que[*rear]=ele;
      count++;
   }
}
int deque(int que[],int *front,int *rear)
{
   int del_ele;
   if(count==0)
   {
      printf("\nstack underflow");
   }
   else
   {
      del_ele=que[(*front)];
      *front=((*front)+1)%que_size;
      count--;
```

```
    }
    return del_ele;
}
void display(int que[],int *front,int *rear)
{
    int f;
    if(count==0)
    {
        printf("\nstack is empty");
    }
{
        f=*front;
        for(int i=0;i<=count;i++)
        {
            printf("\n element is %d",que[f]);
             f=(f+1)%que_size;
        }
    }
}
void main()
{
    int que[que_size],front=0,rear=-1,ele,ch,del_ele;
    do
    {
        printf("\n enter 1 for enque\n enter 2 for deque\n enter 3 for display\n");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
```

```
        {
            printf("\n enter the element to add:");
            scanf("%d",&ele);
            enque(que,&front,&rear,ele);
            printf("\n element added ");
            break;
        }
    case 2:
        {
            del_ele=deque(que,&front,&rear);
            printf("\n deleted element is %d ",del_ele);
            break;
        }
    case 3:
        {
            display(que,&front,&rear);
            break;
        }

    default:
        {
            exit(0);
        }
    }
  }while(1);
}
```

Output:

```
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

 enter the element to add:10

 element added
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

 enter the element to add:20

 element added
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
1

 enter the element to add:20

 queue overflow
 element added
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
2

 deleted element is 10
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
2

 deleted element is 20
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
```

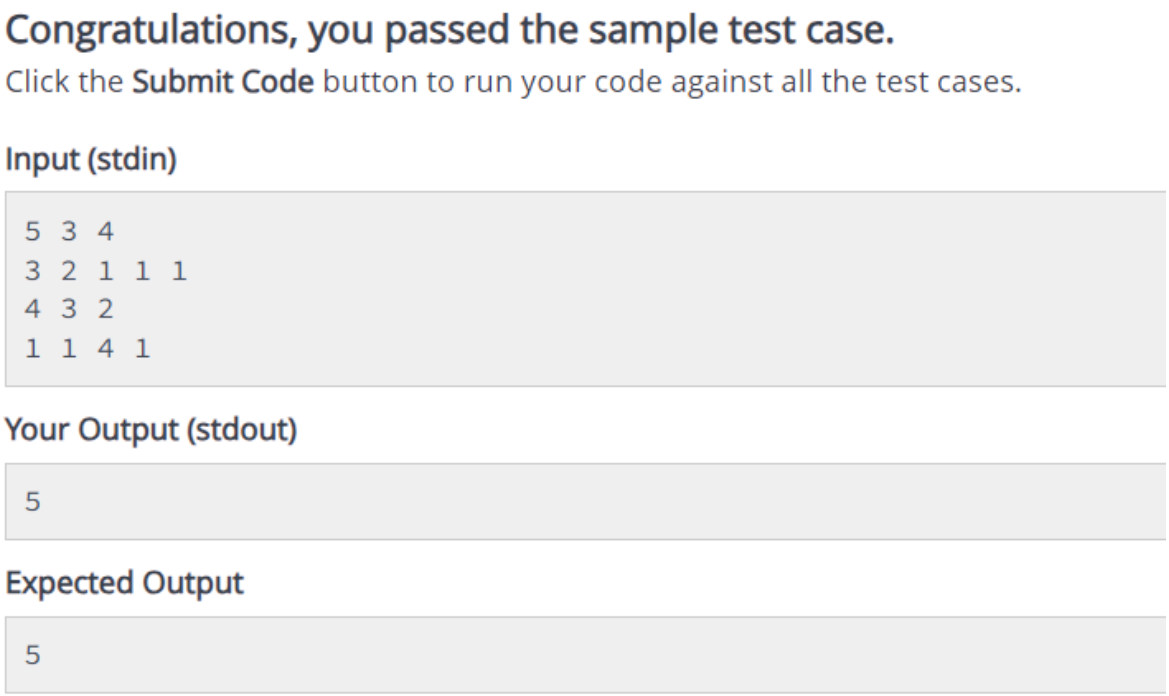
```
 deleted element is 20
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
2

stack underflow
 deleted element is 0
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
3

stack is empty
 element is 10
 enter 1 for enque
 enter 2 for deque
 enter 3 for display
```

Lab program 6:hackerrank

1)You have three stacks of cylinders where each cylinder has the same diameter, but they may vary in height. You can change the height of a stack by removing and discarding its topmost cylinder any number of times.
Find the maximum possible height of the stacks such that all of the stacks are exactly the same height. This means you must remove zero or more cylinders from the top of zero or more of the three stacks until they are all the same height, then return the height.

```
int equalStacks(int h1_count, int* h1, int h2_count, int* h2, int h3_count, int* h3) {
   int sum1 = 0, sum2 = 0, sum3 = 0;

   for (int i = 0; i < h1_count; i++) {
      sum1 += h1[i];
   }

   for (int i = 0; i < h2_count; i++) {
      sum2 += h2[i];
   }

   for (int i = 0; i < h3_count; i++) {
      sum3 += h3[i];
   }

   int top1 = 0, top2 = 0, top3 = 0;


   while (1) {

      if (sum1 == sum2 && sum2 == sum3) {
         return sum1;
      }


      if (sum1 >= sum2 && sum1 >= sum3) {
         sum1 -= h1[top1++];
      } else if (sum2 >= sum1 && sum2 >= sum3) {
         sum2 -= h2[top2++];
      } else {
         sum3 -= h3[top3++];
      }
   }
}
```

Output:



2) A queue is an abstract data type that maintains the order in which elements were added to it, allowing the oldest elements to be removed from the front and new elements to be added to the rear. This is called a *First-In-First-Out* (FIFO) data structure because the first element added to the queue (i.e., the one that has been waiting the longest) is always the first one to be removed. A basic queue has the following operations:

- ***Enqueue*: add a new element to the end of the queue.**
- ***Dequeue*: remove the element from the front of the queue and return it.**

In this challenge, you must first implement a queue using *two stacks*. Then process queries, where each query is one of the following types:

1. `1 x`: **Enqueue element into the end of the queue.**
2. `2`: **Dequeue the element at the front of the queue.**
3. `3`: **Print the element at the front of the queue.**

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

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

#define max_size 1000000
void enqueue(int stack1[max_size], int* top1, int* top2, int element) {
   stack1[++(*top1)] = element;
}
int dequeue(int stack1[max_size], int stack2[max_size], int* top1, int* top2) {
   if (*top2 == -1) {
      while (*top1 != -1) {
         stack2[++(*top2)] = stack1[(*top1)--];
      }
   }

   if (*top2 == -1) {
      return -1;
   }

   return stack2[(*top2)--];
}
int front(int stack1[max_size], int stack2[max_size], int* top1, int* top2) {
   if (*top2 == -1) {
      while (*top1 != -1) {
         stack2[++(*top2)] = stack1[(*top1)--];
      }
   }

   if (*top2 == -1) {
      return -1;
   }

   return stack2[*top2];
}
int main() {
   int stack1[max_size];
   int stack2[max_size];
   int top1=-1, top2=-1;


   int num_queries, query_type, element;
   scanf("%d", &num_queries);

   for (int i = 0; i < num_queries; ++i) {
      scanf("%d", &query_type);

      switch (query_type) {
         case 1:
            scanf("%d", &element);
            enqueue(stack1, &top1, &top2, element);
```

```
            break;
        case 2:
            dequeue(stack1, stack2, &top1, &top2);
            break;
        case 3:
            printf("%d\n", front(stack1, stack2, &top1, &top2));
            break;
      }
   }

   return 0;
}
```

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
10
1 42
2
1 14
3
1 28
3
1 60
1 78
2
2
```

Your Output (stdout)

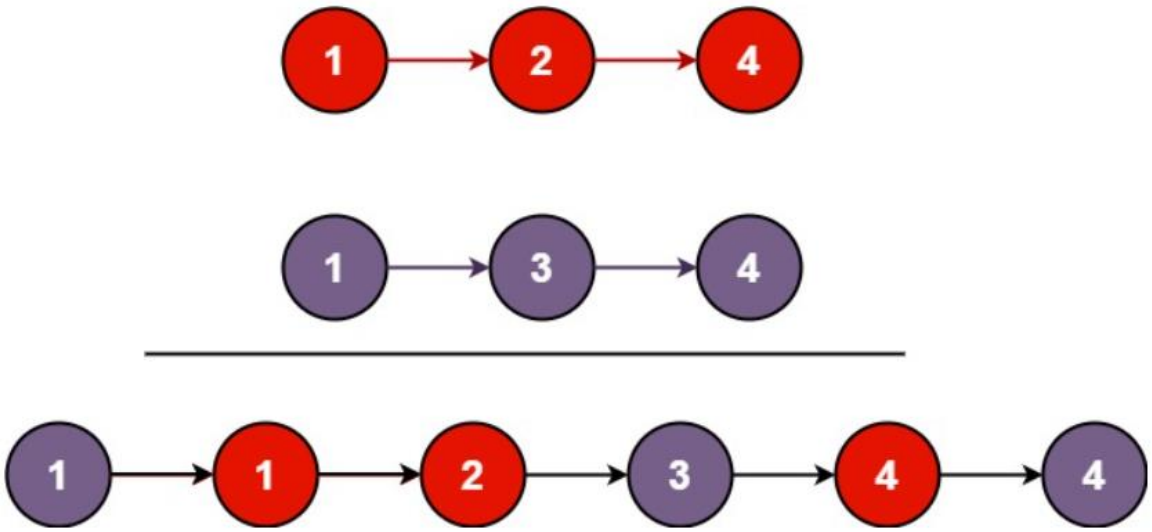
```
14
14
```

Expected Output

```
14
14
```

Lab program 7:

Merge Two Sorted Lists




```
Input: list1 = [1,2,4], list2 = [1,3,4]
Output: [1,1,2,3,4,4]
```

Example 2:

```
Input: list1 = [], list2 = []
Output: []
```

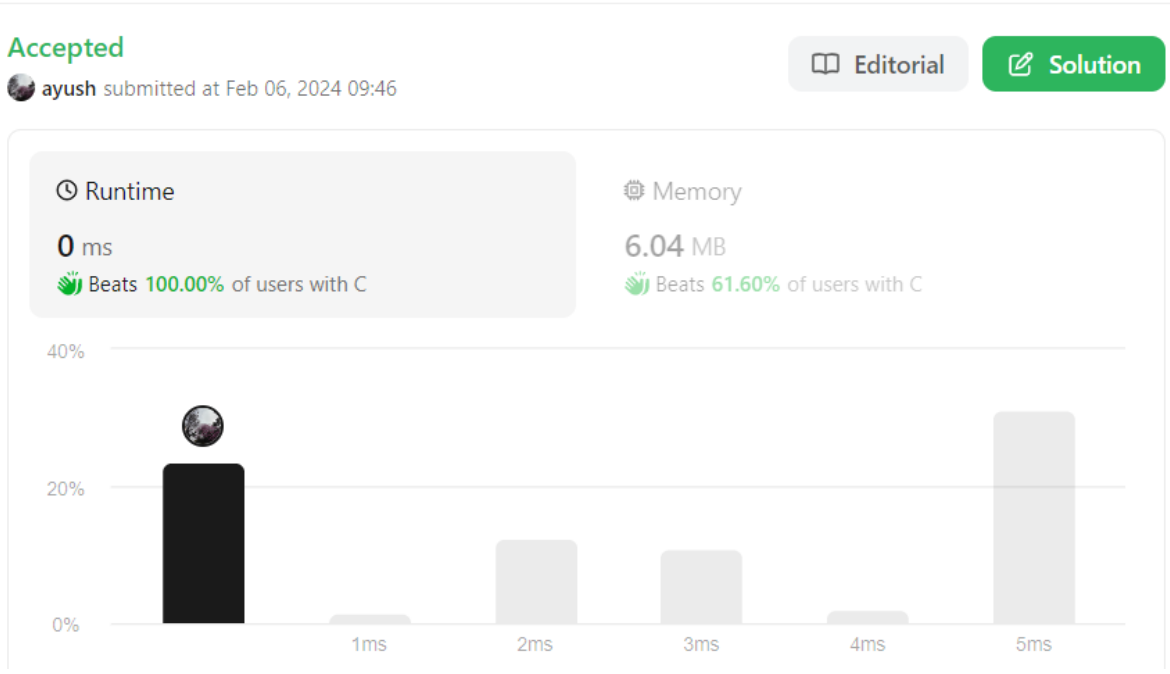
Example 3:

```
Input: list1 = [], list2 = [0]
Output: [0]
```

```
/**
 * Definition for singly-linked list.
 * struct ListNode {
 *     int val;
 *     struct ListNode *next;
 * };
 */
struct ListNode* mergeTwoLists(struct ListNode* list1, struct ListNode* list2) {
  struct ListNode* dummy = (struct ListNode*)malloc(sizeof(struct ListNode));
    dummy->val = 0;
```

```
dummy->next = NULL;


struct ListNode* current = dummy;


while (list1 != NULL && list2 != NULL) {
    if (list1->val <= list2->val) {
        current->next = list1;
        list1 = list1->next;
    } else {
        current->next = list2;
        list2 = list2->next;
    }
    current = current->next;
}


if (list1 != NULL) {
    current->next = list1;
} else {
    current->next = list2;
}


struct ListNode* mergedList = dummy->next;


free(dummy);
```

```
    return mergedList;
}
```



Lab program 8:

1)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.

2)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.

```
1)#include <stdio.h>
#include <stdlib.h>
typedef struct nodes
{
    int data;
    struct nodes *next;
} node;
```

```
node *insertbg(struct nodes *head, int value)
{
   node *new = (node *)malloc(sizeof(node));
   ;
   new->data = value;
   new->next = NULL;
   if (head == NULL)
   {
      return new;
   }
   else
   {
      new->next = head;
      head = new;
   }
   return head;
}
node *inserteg(node *head, int value)
{
   node *new, *ptr;
   new = (node *)malloc(sizeof(node));
   ptr = (node *)malloc(sizeof(node));
   new->data = value;
   new->next = NULL;
   if (head == NULL)
   {
      return new;
   }
   else
```

```
    {
        ptr = head;
        while (ptr->next != NULL)
        {
            ptr = ptr->next;
        }
        ptr->next = new;
    }
    return head;
}
node *insertatp(node *head, int pos, int value)
{
    node *new, *ptr;
    int count = 1;
    new = (node *)malloc(sizeof(node));
    ptr = (node *)malloc(sizeof(node));
    new->data = value;
    new->next = NULL;
    if (head == NULL && pos == 1)
    {
        return new;
    }
    else
    {
        ptr = head;
        while (count < pos - 1 && ptr != NULL)
        {
            ptr = ptr->next;
            count++;
```

```
        }
        if (count == pos - 1)
        {
            new->next = ptr->next;
            ptr->next = new;
        }
        if (ptr == NULL)
        {
            printf("invalid position");
        }
    }
    return head;
}
node *search(node *head,int value)
{
    node *ptr = head;
    if(ptr==NULL)
    {
        return NULL;
    }
    while(ptr!=NULL)
    {
        if(ptr->data==value)
        {
            break;
        }
        ptr = ptr->next;
    }
    if(ptr->data=value)
```

```
    {
        return ptr;
    }
}
int main()
{
    node *head = NULL;
    head = insertbg(head, 8);
    head = insertbg(head, 3);
    head = insertbg(head, 34);
    head = insertbg(head, 35);
    head = insertbg(head, 39);
    //head = inserteg(head, 39);
    //head = insertatp(head, 3, 7);
    node *ptr = (node *)malloc(sizeof(node));
    ptr = head;
    while (ptr != NULL)
    {
        printf("%d->", ptr->data);
        ptr = ptr->next;
    }
}
```

Output:

```
39->35->34->3->8->
Process returned 0 (0x0)   execution time : 0.102 s
Press any key to continue.
```

2)

```
#include <stdio.h>
#include <stdlib.h>
struct nodes
{
   int data;
   struct nodes *link;
};
struct nodes *deletebg(struct nodes *head)
{
   struct nodes *ptr;
   if (head == NULL)
   {
      printf("create a list");
   }
   else
   {
      ptr = head;
      head = head->link;
      free(ptr);
   }
   return head;
}
struct nodes *deleteeg(struct nodes *head)
{
   struct nodes *ptr, *prev;
   prev = malloc(sizeof(struct nodes));
   ptr = malloc(sizeof(struct nodes));
   if (head == NULL)
```

```
    {
        free(head);
    }
    else
    {
        ptr = head;
        while (ptr->link != NULL)
        {
            prev = ptr;
            ptr = ptr->link;
        }
        prev->link = NULL;
        free(ptr);
    }
    return head;
}
struct nodes *deletetgiven(struct nodes *head, int data)
{
    struct nodes *ptr, *ptr1, *ptr2, *ptr3;
    ptr = malloc(sizeof(struct nodes));
    ptr1 = malloc(sizeof(struct nodes));
    ptr = head;
    while (ptr->link != NULL)
    {
        ptr1 = ptr;
        ptr = ptr->link;
        if (ptr->data == data)
        {
            ptr2 = ptr->link;
```

```
            ptr3 = ptr;
            ptr = ptr1;
            ptr->link = ptr2;
        }
    }
     free(ptr3);
    return head;
}
int main()
{
    struct nodes *head;
    struct nodes *ptr;
    struct nodes *current;
    head = malloc(sizeof(struct nodes));
    head->data = 10;
    head->link = NULL;
    current = malloc(sizeof(struct nodes));
    current->data = 11;
    current->link = NULL;
    head->link = current;
    current = malloc(sizeof(struct nodes));
    current->data = 12;
    current->link = NULL;
    head->link->link = current;
    current = malloc(sizeof(struct nodes));
    current->data = 13;
    current->link = NULL;
    head->link->link->link = current;
    //head=deletebg(head);
```

```
    //head=deleteeg(head);
   // head=deletetgiven(head,11);
    ptr = head;
    while (ptr != NULL)
    {
        printf("%d->", ptr->data);
        ptr = ptr->link;
    }
}
```

Output:

```
10->11->12->13->
Process returned 0 (0x0)   execution time : 0.136 s
Press any key to continue.
```

Hackerrank:

Given the pointer to the head node of a linked list and an integer to insert at a certain position, create a new node with the given integer as its attribute, insert this node at the desired position and return the head node.
A position of 0 indicates head, a position of 1 indicates one node away from the head and so on. The head pointer given may be null meaning that the initial list is empty.

```
SinglyLinkedListNode* insertNodeAtPosition(SinglyLinkedListNode* llist, int data, int position) {
    SinglyLinkedListNode* new,*ptr;
    new=create_singly_linked_list_node(data);
    int count=0;
    ptr=llist;
    if(llist==NULL)
    {
        return new;
    }
```

```
    while(count<position-1&&ptr!=NULL)
    {
       ptr=ptr->next;
       count++;
    }
    if(ptr==NULL)
    {
       printf("invalid position");
    }
    if(count==position-1)
    {
       new->next=ptr->next;
       ptr->next=new;
    }
    return llist;
}
```

Congratulations, you passed the sample test case.

Click the **Submit Code** button to run your code against all the test cases.

Input (stdin)

```
3
16
13
7
1
2
```

Your Output (stdout)

```
16 13 1 7
```

Expected Output

```
16 13 1 7
```

Lab program 9:

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

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

6a)

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

struct nodes
{
   int data;
   struct nodes *link;
};

struct nodes *sort(struct nodes *head)
{
   struct nodes *temp1, *temp2;
   int data;

   for (temp1 = head; temp1 != NULL; temp1 = temp1->link)
   {
      for (temp2 = temp1->link; temp2 != NULL; temp2 = temp2->link)
      {
         if (temp1->data > temp2->data)
         {
            data = temp1->data;
            temp1->data = temp2->data;
            temp2->data = data;
         }
      }
   }
```

```
    return head;
}
```

Output:

```
30->50->90->
number of nodes is 3
Process returned 0 (0x0)   execution time : 1.284 s
Press any key to continue.
```

```
struct nodes *reversed(struct nodes *head)
{
   struct nodes *current, *next;
   current = head;
   if (head == NULL || head->link == NULL)
   {
      return head;
   }
   while (current->link != NULL)
   {
      next = current->link;
      current->link = next->link;
      next->link = head;
      head = next;
   }
   return head;
}
```

Output:

```
before
45->50->60->
after
60->50->45->
```

```
#include<stdio.h>
#include<stdlib.h>
struct nodes
{
   int data;
   struct nodes *link;
};
struct nodes* insertbg(struct nodes *head,int info)
```

```
{
   struct nodes *new;
   new=malloc(sizeof(struct nodes));
   new->data=info;
   new->link=head;
   head=new;
   return head;
}
struct nodes *concatenate(struct node *head1,struct node *head2)
{
   struct nodes *ptr;
   ptr=head1;
   if(head1==NULL)
   {
      return head2;
   }
   if(head2==NULL)
   {
      return head1;
   }
   while(ptr->link!=NULL)
   {
      ptr=ptr->link;
   }
   ptr->link=head2;
   return head1;
};
int main()
{
```

```
    struct nodes *head1,*head2=NULL;
    head2=insertbg(head2,89);
    head2=insertbg(head2,99);
    head2=insertbg(head2,88);
    head2=insertbg(head2,98);
    head1=malloc(sizeof(struct nodes));
    head1->data=56;
    head1->link=NULL;
    struct nodes *current;
    current=malloc(sizeof(struct nodes));
    current->data=90;
    current->link=NULL;
    head1->link=current;
    current=malloc(sizeof(struct nodes));
    current->data=98;
    current->link=NULL;
    head1->link->link=current;
    concatenate(head1,head2);
    struct nodes *ptr;
    ptr=malloc(sizeof(struct nodes));
    ptr=head1;
    while(ptr!=NULL)
    {
       printf("%d",ptr->data);
       ptr=ptr->link;
    }
}
```

Output:

```
56->90->98->98->88->99->89->
Process returned 0 (0x0)   execution time : 1.259 s
Press any key to continue.
```

6b)

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

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

node *stack_head = NULL;  // Head pointer for stack

void push(int value) {
    node *new_node = (node *)malloc(sizeof(node));
    if (new_node == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    new_node->data = value;
    new_node->next = stack_head;
    stack_head = new_node;  // Push to the top of the stack
    printf("%d pushed to stack\n", value);
}

int pop() {
    if (stack_head == NULL) {
        printf("Stack underflow.\n");
        return -1;
    }

    node *temp = stack_head;
    int value = temp->data;
```

```
    stack_head = stack_head->next;
    free(temp);
    printf("%d popped from stack\n", value);
    return value;
}

void displayStack() {
    if (stack_head == NULL) {
        printf("Stack is empty.\n");
        return;
    }

    printf("Stack elements:\n");
    node *temp = stack_head;
    while (temp != NULL) {
        printf("%d\n", temp->data);
        temp = temp->next;
    }
}

int main() {
int choice, value;
printf(“ayush-1BM22CS063\n”);

    while (1) {
        printf("\nStack Operations:\n");
        printf("1. Push\n");
        printf("2. Pop\n");
        printf("3. Display Stack\n");
        printf("4. Exit\n");
        printf("Enter your choice: ");
```

```
        scanf("%d", &choice);

        switch (choice) {
            case 1:
                printf("Enter value to push: ");
                scanf("%d", &value);
                push(value);
                break;
            case 2:
                value = pop();
                if (value != -1) {
                    printf("%d popped from stack\n", value);
                }
                break;
            case 3:
                displayStack();
            case 4:
                printf("Exiting...\n");
                exit(0);
            default:
                printf("Invalid choice.\n");
        }
    }
    return 0;
}
```

Output:

```
ayush-1BM22CS063

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 1
Enter value to push: 1
1 pushed to stack

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 1
Enter value to push: 2
2 pushed to stack

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 2
2 popped from stack
2 popped from stack

Stack Operations:
1. Push
2. Pop
3. Display Stack
4. Exit
Enter your choice: 3
Stack elements:
1
Exiting...
```

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

typedef struct node {
    int data;
```

```
    struct node *next;
} node;

node *queue_head = NULL;  // Head pointer for queue
node *queue_tail = NULL;  // Tail pointer for efficient enqueue

void enqueue(int value) {
    node *new_node = (node *)malloc(sizeof(node));
    if (new_node == NULL) {
        printf("Memory allocation failed.\n");
        return;
    }

    new_node->data = value;
    new_node->next = NULL;

    if (queue_head == NULL) {
        queue_head = queue_tail = new_node;  // First node in the queue
    } else {
        queue_tail->next = new_node;
        queue_tail = new_node;  // Update tail pointer
    }
    printf("%d enqueued to queue\n", value);
}

int dequeue() {
    if (queue_head == NULL) {
        printf("Queue underflow.\n");
        return -1;
    }

    node *temp = queue_head;
```

```
    int value = temp->data;
    queue_head = queue_head->next;
    free(temp);

    if (queue_head == NULL) {
        queue_tail = NULL;  // Reset tail pointer if queue becomes empty
    }

    printf("%d dequeued from queue\n", value);
    return value;
}

void displayQueue() {
    if (queue_head == NULL) {
        printf("Queue is empty.\n");
        return;
    }

    printf("Queue elements:\n");
    node *temp = queue_head;
    while (temp != NULL) {
        printf("%d ", temp->data);
        temp = temp->next;
    }
    printf("\n");
}

int main() {
    int choice, value;

    while (1) {
        printf("\nQueue Operations:\n");
```

```
    printf("1. Enqueue\n");
    printf("2. Dequeue\n");
    printf("3. Display Queue\n");
    printf("4. Exit\n");
    printf("Enter your choice: ");
    scanf("%d", &choice);

    switch (choice) {
        case 1:
            printf("Enter value to enqueue: ");
            scanf("%d", &value);
            enqueue(value);
            break;
        case 2:
            value = dequeue();
            if (value != -1) {
                printf("%d dequeued from queue\n", value);
            }
            break;
        case 3:
            displayQueue();
            break;
        case 4:
            printf("Exiting...\n");
            exit(0);
        default:
            printf("Invalid choice.\n");
    }
}
```

```
    return 0;
}
```

Output:

```
ayush-1BM22CS063

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 1
1 enqueued to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 2
2 enqueued to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 1
Enter value to enqueue: 3
3 enqueued to queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
1 dequeued from queue
1 dequeued from queue
```

```
Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 2
2 dequeued from queue
2 dequeued from queue

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice: 3
Queue elements:
3

Queue Operations:
1. Enqueue
2. Dequeue
3. Display Queue
4. Exit
Enter your choice:
```

3)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 value;
   struct node *prev;
   struct node *next;
} Node;
Node *insertleft(Node *head, int data, int key)
{
   Node *new,*ptr;
   new = malloc(sizeof(Node));
   new->value = data;
   new->prev = NULL;
   new->next = NULL;
   ptr = head;
   if(head==NULL)
   {
      return new;
   }
   while(ptr!=NULL)
   {
      if(ptr->value==key)
      {
         break;
```

```
        }
        ptr=ptr->next;
    }
    if(ptr->value==key)
    {
        new->prev = ptr->prev;
        (ptr->prev)->next = new;
        new->next = ptr;
        ptr->prev = new;
        return head;
    }
    printf("no values");
    return head;
}
Node *deleteval(Node *head,int key)
{
    Node *ptr;
    if(head==NULL)
    {
        printf("list empty");
        return NULL;
    }
    ptr=head;
    while(ptr!=NULL&&ptr->value!=key)
    {
        ptr=ptr->next;
    }
    if(ptr->value==key)
    {
```

```
        (ptr->next)->prev=ptr->prev;
        (ptr->prev)->next=ptr->next;
        free(ptr);
        return head;
    }
    printf("no value");
    return head;
}
int main()
{
    Node *head = malloc(sizeof(Node));
    head->value = 8;
    head->prev = NULL;
    head->next = NULL;
    Node *current = malloc(sizeof(Node));
    current->value = 10;
    current->prev = head;
    current->next = NULL;
    head->next = current;
    Node *current2 = malloc(sizeof(Node));
    current2->value = 14;
    current2->prev = current;
    current2->next = NULL;
    current->next = current2;
    insertleft(head, 15, 14);
    Node *ptr1 = head;
    while (ptr1 != NULL)
    {
        printf("%d\n", ptr1->value);
```

```
        ptr1 = ptr1->next;
    }
    deleteval(head,8);
    Node *ptr = head;
    while (ptr != NULL)
    {
        printf("%d", ptr->value);
        ptr = ptr->next;
    }
}
```

Output:

```
ayush_1BM22CS063
8
10
15
14

Process returned -1073741819 (0xC0000005)   execution time : 2.718 s
Press any key to continue.
```

Lab program 10:

Program 1:

Given a Linked List and a number , write a function that returns the value at the Nth node from the end of the Linked List.

Program 2:

Write a function that takes a list sorted in non-decreasing order and deletes any duplicate nodes from the list. The list should only be traversed once.

For example if the linked list is 11->11->11->21->43->43->60 then removeDuplicates() should convert the list to 11->21->43->60.

```
#include <stdio.h>
#include <stdlib.h>
typedef struct nodes
{
```

```
    int data;
    struct nodes *next;
} node;
node *insertbg(struct nodes *head, int value)
{
    node *new = (node *)malloc(sizeof(node));
    ;
    new->data = value;
    new->next = NULL;
    if (head == NULL)
    {
        return new;
    }
    else
    {
        new->next = head;
        head = new;
    }
    return head;
}
int given(struct nodes *head, int pos)
{
    struct nodes *ptr;
    int count=0;
    ptr = malloc(sizeof(struct nodes));
    ptr = head;
    while (ptr->next!= NULL&&count!=pos-1)
    {
      ptr=ptr->next;
```

```
    count++;
  }
  if(count==pos-1)
  {
    return ptr->data;
  }
  else{
    return NULL;
  }
}
```

Output:


```
int main()
{
  node *head = NULL;
  int n;
  head = insertbg(head, 8);
  head = insertbg(head, 3);
  head = insertbg(head, 34);
  head = insertbg(head, 35);
  head = insertbg(head, 39);
  node *ptr = (node *)malloc(sizeof(node));
  ptr = head;
  while (ptr != NULL)
  {
    printf("%d->", ptr->data);
    ptr = ptr->next;
```

```
    }
    n=given(head,3);
    printf("\nthe nth node is %d",n);
}

#include <stdio.h>
#include <stdlib.h>
typedef struct nodes
{
    int data;
    struct nodes *next;
} node;
node *insertbg(struct nodes *head, int value)
{
    node *new = (node *)malloc(sizeof(node));
    ;
    new->data = value;
    new->next = NULL;
    if (head == NULL)
    {
        return new;
    }
    else
    {
        new->next = head;
        head = new;
    }
    return head;
}
```

```
node *removeduplicates(node *head)
{
   node *ptr,*prev;
   ptr=head;
   while(ptr!=NULL&&ptr->next!=NULL){
         if(ptr->data==ptr->next->data)
         {
            prev=ptr->next;
            ptr->next=ptr->next->next;
            free(prev);
         }
         else{
            ptr=ptr->next;
         }
   }
   return head;
}
int main()
{
   node *head = NULL;
   head = insertbg(head, 60);
   head = insertbg(head, 43);
   head = insertbg(head, 43);
   head = insertbg(head, 21);
   head = insertbg(head, 11);
   head = insertbg(head, 11);
   head = insertbg(head, 11);
   node *ptr = (node *)malloc(sizeof(node));
   ptr = removeduplicates(head);
```

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

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 node
{
    int data;
    struct node *leftnode;
    struct node *rightnode;
} node;

node *create(int data)
{
    node *a;
    a = malloc(sizeof(node));
    a->data = data;
    a->leftnode = NULL;
    a->rightnode = NULL;
    return a;
}
```

```
void preordertraversal(node *root)
{
   if (root != NULL)
   {
      printf("%d\t", root->data);
      preordertraversal(root->leftnode);
      preordertraversal(root->rightnode);
   }
}

void postordertraversal(node *root)
{
   if (root != NULL)
   {
      postordertraversal(root->leftnode);
      postordertraversal(root->rightnode);
      printf("%d\t", root->data);
   }
}

void inordertraversal(node *root)
{
   if (root != NULL)
   {
      inordertraversal(root->leftnode);
      printf("%d\t", root->data);
      inordertraversal(root->rightnode);
   }
}
```

```
int isBST(node *root)
{
   static node *prev = NULL;
   if (root != NULL)
   {
      if (!isBST(root->leftnode))
      {
         return 0;
      }
      else if (prev != NULL && root->data <= prev->data)
      {
         return 0;
      }
      prev = root;
      return isBST(root->rightnode);
   }
   else
      return 1;
}

node *search_iterative(node *root, int key)
{
   if (root == NULL)
   {
      return NULL;
   }
   while (root != NULL)
   {
      if (key == root->data)
      {
```

```
            return root;
        }
        else if (key < root->data)
        {
            root = root->leftnode;
        }
        else if (key > root->data)
        {
            root = root->rightnode;
        }
    }
    return NULL; // Return NULL if the key is not found
}

void insert(node **root, int key)
{
    node *news, *prev = NULL;
    news = create(key);
    if (*root == NULL)
    {
        *root = news;
        return;
    }
    node *temp = *root; // Use a temporary variable for traversal
    while (temp != NULL)
    {
        prev = temp;
        if (key == temp->data)
        {
            printf("cannot insert");
```

```
            free(news); // Free the allocated memory before returning
            return;
        }
        else if (key < temp->data)
        {
            temp = temp->leftnode;
        }
        else if (key > temp->data)
        {
            temp = temp->rightnode;
        }
    }
    if (key < prev->data)
    {
        prev->leftnode = news;
    }
    else if (key > prev->data)
    {
        prev->rightnode = news;
    }
}

node *inorderpredecessor(node *root)
{
    root = root->leftnode;
    while (root->rightnode != NULL)
    {
        root = root->rightnode;
    }
    return root;
```

```
}

node *deletenode(node *root, int value)
{
   node *ipre;
   if (root == NULL)
   {
      return NULL;
   }
   if (value < root->data)
   {
      root->leftnode = deletenode(root->leftnode, value);
   }
   else if (value > root->data)
   {
      root->rightnode = deletenode(root->rightnode, value);
   }
   else
   {
      if (root->leftnode == NULL)
      {
         node *temp = root->rightnode;
         free(root);
         return temp;
      }
      else if (root->rightnode == NULL)
      {
         node *temp = root->leftnode;
         free(root);
         return temp;
```

```
        }
        ipre = inorderpredecessor(root);
        root->data = ipre->data;
        root->leftnode = deletenode(root->leftnode, ipre->data);
    }
    return root;
}

int main()
{
    node *root = create(50);
    node *leafl = create(45);
    node *leafr = create(60);
    node *leafll = create(35);
    node *leafrr = create(65);
    node *leaflr = create(47);
    node *leafrl = create(55);
    root->leftnode = leafl;
    root->rightnode = leafr;
    leafl->leftnode = leafll;
    leafr->rightnode = leafrr;
    leafl->rightnode = leaflr;
    leafr->leftnode = leafrl;

    insert(&root, 70);

    printf("Preorder Traversal: ");
    preordertraversal(root);
    printf("\n");

    printf("Postorder Traversal: ");
```

```
    postordertraversal(root);
    printf("\n");

    printf("Inorder Traversal: ");
    inordertraversal(root);
    printf("\n");

    printf("Is BST: %d\n", isBST(root));

    int keyToSearch = 80;
    node *n = search_iterative(root, keyToSearch);
    if (n != NULL)
    {
        printf("Element %d found\n", n->data);
    }
    else
    {
        printf("Element not found\n");
    }

    printf("Deleting node with value 50\n");
    root = deletenode(root, 50);

    printf("After deletion\n");
    printf("Preorder Traversal: ");
    preordertraversal(root);
    printf("\n");

    return 0;
}
```

Output:

```
Preorder Traversal: 50  45      35      47      60      55      65      70
Postorder Traversal: 35 47      45      55      70      65      60      50
Inorder Traversal: 35   45      47      50      55      60      65      70

Process returned 0 (0x0)   execution time : 0.834 s
Press any key to continue.
```

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.

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

#define MAX_NODES 100

struct Queue {
    int items[MAX_NODES];
    int front;
    int rear;
};

struct Graph {
    int vertices;
    int** adjMatrix;
};

struct Queue* createQueue() {
    struct Queue* queue = (struct Queue*)malloc(sizeof(struct Queue));
    queue->front = -1;
    queue->rear = -1;
    return queue;
}

void enqueue(struct Queue* queue, int value) {
    if (queue->rear == MAX_NODES - 1) {
        printf("Queue is full\n");
    } else {
        if (queue->front == -1) {
```

```
            queue->front = 0;
        }
        queue->rear++;
        queue->items[queue->rear] = value;
    }
}

int dequeue(struct Queue* queue) {
    int item;
    if (queue->front == -1) {
        printf("Queue is empty\n");
        item = -1;
    } else {
        item = queue->items[queue->front];
        queue->front++;
        if (queue->front > queue->rear) {
            queue->front = queue->rear = -1;
        }
    }
    return item;
}

bool isEmpty(struct Queue* queue) {
    return queue->front == -1;
}

struct Graph* createGraph(int vertices) {
    struct Graph* graph = (struct Graph*)malloc(sizeof(struct Graph));
    graph->vertices = vertices;

    graph->adjMatrix = (int**)malloc(vertices * sizeof(int*));
```

```
    for (int i = 0; i < vertices; i++) {
        graph->adjMatrix[i] = (int*)malloc(vertices * sizeof(int));
        for (int j = 0; j < vertices; j++) {
            graph->adjMatrix[i][j] = 0;
        }
    }

    return graph;
}

void addEdge(struct Graph* graph, int src, int dest) {
    graph->adjMatrix[src][dest] = 1;
    graph->adjMatrix[dest][src] = 1;
}

void BFS(struct Graph* graph, int startNode) {
    struct Queue* queue = createQueue();
    bool visited[MAX_NODES] = {false};

    printf("BFS traversal starting from node %d: ", startNode);

    visited[startNode] = true;
    printf("%d ", startNode);
    enqueue(queue, startNode);

    while (!isEmpty(queue)) {
        int currentNode = dequeue(queue);

        for (int i = 0; i < graph->vertices; i++) {
            if (graph->adjMatrix[currentNode][i] == 1 && !visited[i]) {
                printf("%d ", i);
                visited[i] = true;
```

```
                enqueue(queue, i);
            }
        }
    }

    printf("\n");
}

bool isCyclicUtil(struct Graph* graph, int v, bool visited[], int parent);

bool isCyclic(struct Graph* graph) {
    bool* visited = (bool*)malloc(graph->vertices * sizeof(bool));
    for (int i = 0; i < graph->vertices; i++) {
        visited[i] = false;
    }

    for (int i = 0; i < graph->vertices; i++) {
        if (!visited[i]) {
            if (isCyclicUtil(graph, i, visited, -1)) {
                free(visited);
                return true;
            }
        }
    }

    free(visited);
    return false;
}

bool isCyclicUtil(struct Graph* graph, int v, bool visited[], int parent) {
    visited[v] = true;
```

```
    for (int i = 0; i < graph->vertices; i++) {
        if (graph->adjMatrix[v][i] == 1) {
            if (!visited[i]) {
                if (isCyclicUtil(graph, i, visited, v)) {
                    return true;
                }
            } else if (i != parent) {
                return true;
            }
        }
    }

    return false;
}

int main() {
    struct Graph* graph = createGraph(4);

    addEdge(graph, 0, 1);
    addEdge(graph, 0, 2);
    addEdge(graph, 1, 2);
    addEdge(graph, 2, 0);
    addEdge(graph, 2, 3);
    addEdge(graph, 3, 3);

    int startNode = 2;
    BFS(graph, startNode);
    if (isCyclic(graph)) {
        printf("The graph contains a cycle.\n");
    } else {
        printf("The graph does not contain a cycle.\n");
```

```
    }

    return 0;
}
```

Output:

```
ayush_1BM22CS063
BFS traversal starting from node 2: 2 0 1 3
The graph contains a cycle.

Process returned 0 (0x0)   execution time : 0.136 s
Press any key to continue.
```