RUSTAMJI INSTITUTE OF TECHNOLOGY

BSF ACADEMY, TEKANPUR

Lab File for

CS303 (Data Structure)

Submitted by

Amisha Goswami   (0902CS231012)
B.Tech. Computer Science & Engineering 3rd Semester
(2023-2027 batch)

Subject Teacher   File Checked by

Dr. Jagdish Makhijani   Mr. Yashwant Pathak

TABLE OF CONTENTS

Section-A (Linked List)

Section-B (Stack)

Section-C (Queue)

Section-D (Trees)

S. No.	Practical Description	Page Nos.	COs
1	Implementation of Binary Search Tree.	44-48	CO-3
2	Conversion of BST PreOrder/PostOrder/InOrder.	49-54	CO-3
3	Implementation of Kruskal Algorithm	55-57	CO-4
4	Implementation of prime Algorithm	58-59	CO-4
5	Implementation of Dijkstra Algorithm	60-63	CO-4

Section-E (Sorting & Searching)

S. No.	Practical Description	Page Nos.	COs
1	Implementation of Sorting a. Bubble b. Selection c. Insertion d. Quick e. Merge	64-73	CO-5
2	Implementation of Binary Search on a list of numbers stored in an Array	74-76	CO-5
3	Implementation of Binary Search on a list of strings stored in an Array	77-79	CO-5
4	Implementation of Linear Search on a list of strings stored in an Array	80-81	CO-5

Section-A (Linked List)

Practical No.: 1

Program Description: Implementation of Linked List using array.

Solution:

```c
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#define MAX 10

struct
{
    int list[MAX];
    int element;
    int pos;
    int length;
};
enum boolean
{
    true, false
};
typedef enum boolean boolean;

int menu(void);
void create(void);
void insert(int, int);
void delet(int);
void find(int);
void display(void);
boolean islistfull(void);
boolean islistempty(void);


int menu()
{
    int ch;
    //clrscr();
    printf("1. Create\n2. Insert\n3. Delete\n4. Count\n5. Find\n6. Display\n7.Exit\n\n Enter your choice : ");
    scanf("%d", &ch);
    printf("\n\n");
    return ch;
}
void create(void)
{
    int element;
```

```c
    int flag=1;
    while(flag==1)
    {
        printf("Enter element : ");
        scanf("%d", &element);
        l.list[l.length] = element;
        l.length++;
        printf("To insert another element press '1' : ");
        scanf("%d", &flag);
    }
}
void display(void)
{
    int i;
    for (i=0; i<l.length; i++)
        printf("Element %d : %d \n", i+1, l.list[i]);
    printf("Press any key to continue...");
    getch();
}
void insert(int element, int pos)
{
    int i;
    if (pos == 0)
    {
        printf("\nCannot insert an element at 0th position");
        getch();
        return;
    }

    if (pos-1 > l.length)
    {
        printf("\nOnly %d elements exit. Cannot insert at %d position", l.length, pos);
        printf("\n Press any key to continue...");
        getch();
    }
    else
    {
        for (i=l.length; i>=pos-1; i--)
        {
            l.list[i+1] = l.list[i];
        }
        l.list[pos-1] = element;
        l.length++;
    }
}
void delet(int pos)
{
    int i;
```

```c
    if(pos == 0)
    {
        printf("\nCannot delete at an element 0th position");
        getch();
        return;
    }
    if (pos > l.length)
    {
        printf("\n\n Only %d elements exit. Cannot delete", l.length, pos);
        printf("\n Press any key to continue...");
        getch();
        return;
    }
    for (i=pos-1; i<l.length; i++)
    {
        l.list[i] = l.list[i+1];
    }
    l.length--;
}

void find(int element)
{
    int i;
    int flag = 1;

    for (i=0; i<l.length; i++)
    {
        if(l.list[i] == element)
        {
            printf ("%d exists at %d position",element, i+1);
            flag = 0;
            printf("\n Press any key to continue...");
            getch();
            break;
        }
    }
    if(flag == 1)
    {
        printf("Element not found.\n Press any key to continue...");
        getch();
    }
}
boolean islistfull(void)
{
    if (l.length == MAX)
        return true;
    else
        return false }
```

```c
boolean islistempty(void)
{
    if (l.length == 0)
        return true;
    else
        return false;
}

int main()
{
    int ch;
    int element;
    int pos;
    l.length = 0;
    while(1)
    {
        ch = menu();
        switch (ch)
        {
            case 1:  l.length = 0;
            create();
            break;
            case 2:
            if (islistfull() != true)
            {
                printf("Enter New element: ");
                scanf("%d", &element);
                printf("Enter the Position : ");
                scanf("%d", &pos);
                insert(element, pos);
            }
            else
            {
                printf("List is Full. Cannot insert the element");
                printf("\n Press any key to continue...");
                getch();
            }
            break;
            case 3:
            if (islistempty() != true)
            {
                printf("Enter the position of element to be deleted : ");
                scanf("%d", &pos);
                delet(pos);     }
            else
            {
                printf("List is Empty.");
                printf("\n Press any key to continue...");
```

```c
            getch();
            }
        break;
        case 4:
        printf("No of elements in the list is %d", l.length);
        printf("\n Press any key to continue...");
        getch();
        break;
        case 5:
        printf("Enter the element to be searched : ");
        scanf("%d", &element);
        find(element);
        break;
        case 6:
        display();
        break;
        case 7:
        printf("Exit");
        exit(0);
        break;
        default:  printf("Invalid Choice");
        printf("\n Press any key to continue...");
        getch();
    }
  }
}       //function to display the list of elements
```

Output:

Practical No.: 2

Program Description: Implementation of Linked List using pointers.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
struct Node;
typedef struct Node * PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;

struct Node
{
    int e;
    Position next;
};

void Insert(int x, List l, Position p)
{
    Position TmpCell;
    TmpCell = (struct Node*) malloc(sizeof(struct Node));
    if(TmpCell == NULL)
        printf("Memory out of space\n");
    else
    {
        TmpCell->e = x;
        TmpCell->next = p->next;
        p->next = TmpCell;
    }
}

int isLast(Position p)
{
    return (p->next == NULL);
}

Position FindPrevious(int x, List l)
{
    Position p = l;
    while(p->next != NULL && p->next->e != x)
        p = p->next;
    return p;
}

void Delete(int x, List l)
{
    Position p, TmpCell;
    p = FindPrevious(x, l);
```

```c
    if(!isLast(p))
    {
        TmpCell = p->next;
        p->next = TmpCell->next;
        free(TmpCell);
    }
    else
        printf("Element does not exist!!!\n");
}


void Display(List l)
{
    printf("The list element are :: ");
    Position p = l->next;
    while(p != NULL)
    {
        printf("%d -> ", p->e);
        p = p->next;
    }
}

void Merge(List l, List l1)
{
    int i, n, x, j;
    Position p;
    printf("Enter the number of elements to be merged :: ");
    scanf("%d",&n);

    for(i = 1; i <= n; i++)
    {
        p = l1;
        scanf("%d", &x);
        for(j = 1; j < i; j++)
            p = p->next;
        Insert(x, l1, p);
    }
    printf("The new List :: ");
    Display(l1);
    printf("The merged List ::");
    p = l;
    while(p->next != NULL)
    {
        p = p->next;
    }
    p->next = l1->next;
    Display(l);
}
```

```c
int main()
{
    int x, pos, ch, i;
    List l, l1;
    l = (struct Node *) malloc(sizeof(struct Node));
    l->next = NULL;
    List p = l;
    printf("LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");
    do
    {
        printf("\n\n1. INSERT\t 2. DELETE\t 3. MERGE\t 4. PRINT\t 5. QUIT\n\nEnter the choice :: ");
        scanf("%d", &ch);
        switch(ch)
        {
        case 1:
            p = l;
            printf("Enter the element to be inserted :: ");
            scanf("%d",&x);
            printf("Enter the position of the element :: ");
            scanf("%d",&pos);

            for(i = 1; i < pos; i++)
            {
                p = p->next;
            }
            Insert(x,l,p);
            break;

        case 2:
            p = l;
            printf("Enter the element to be deleted :: ");
            scanf("%d",&x);
            Delete(x,p);
            break;

        case 3:
            l1 = (struct Node *) malloc(sizeof(struct Node));
            l1->next = NULL;
            Merge(l, l1);
            break;

        case 4:
            Display(l);
            break;
        }
    }
    while(ch<5);
    return 0;}
```

Practical No.: 3

Program Description: Implementation of Doubly Linked List using Pointers

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    struct node *next;
    int data;
};
struct node *head;
void insertionFirst();
void insertionLast();
void insertionLoc();
void deleteFirst();
void deleteLast();
void deleteLoc();
void printList();
void searchList();
int main()
{
    int choice =0;
    while(choice != 9)
    {
        printf("\nDoubly Linked ListMenu\n");
        printf("\n1.Insert at begining\n");
        printf("2.Insert at last\n");
        printf("3.Insert at any random location\n");
        printf("4.Delete from Beginning\n");
        printf("5.Delete from last\n");
        printf("6.Delete the node after the given data\n");
        printf("7.Search\n");
        printf("8.Show\n");
        printf("9.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
                insertionFirst();
                break;
            case 2:
                insertionLast();
                break;
            case 3:
                insertionLoc();
```

```c
                break;
            case 4:
                deleteFirst();
                break;
            case 5:
                deleteLast();
                break;
            case 6:
                deleteLoc();
                break;
            case 7:
                searchList();
                break;
            case 8:
                printList();
                break;
            case 9:
                exit(0);
                break;
            default:
                printf("Invalid Choice!!! Please try again....");
        }
    }
    return 0;
}
void insertionFirst()
{
    struct node *ptr;
    int item;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\nOVERFLOW!!!");
    }
    else
    {
    printf("\nEnter value to insert: ");
    scanf("%d",&item);

    if(head==NULL)
    {
        ptr->next = NULL;
        ptr->prev=NULL;
        ptr->data=item;
        head=ptr;
    }
    else
    {
```

```c
      ptr->data=item;
      ptr->prev=NULL;
      ptr->next = head;
      head->prev=ptr;
      head=ptr;
   }
   printf("\nNode inserted successfully....\n");
}


}
void insertionLast()
{
   struct node *ptr,*temp;
   int item;
   ptr = (struct node *) malloc(sizeof(struct node));
   if(ptr == NULL)
   {
      printf("\nOVERFLOW!!!");
   }
   else
   {
      printf("\nEnter value to insert: ");
      scanf("%d",&item);
       ptr->data=item;
      if(head == NULL)
      {
         ptr->next = NULL;
         ptr->prev = NULL;
         head = ptr;
      }
      else
      {
        temp = head;
        while(temp->next!=NULL)
        {
           temp = temp->next;
        }
        temp->next = ptr;
        ptr ->prev=temp;
        ptr->next = NULL;
        }

   }
   printf("\nNode inserted successfully\n");
  }
void insertionLoc()
{
   struct node *ptr,*temp;
```

```c
    int item,loc,i;
    ptr = (struct node *)malloc(sizeof(struct node));
    if(ptr == NULL)
    {
        printf("\n OVERFLOW!!!");
    }
    else
    {
        temp=head;
        printf("Enter the location: ");
        scanf("%d",&loc);
        for(i=0;i<loc;i++)
        {
            temp = temp->next;
            if(temp == NULL)
            {
                printf("\nThere are less than %d elements\n", loc);
                return;
            }
        }
        printf("Enter value: ");
        scanf("%d",&item);
        ptr->data = item;
        ptr->next = temp->next;
        ptr -> prev = temp;
        temp->next = ptr;
        temp->next->prev=ptr;
        printf("\nNode inserted successfully...\n");
    }
}
void deleteFirst()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW!!!");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nNode deleted successfully....\n");
    }
    else
    {
        ptr = head;
        head = head -> next;
        head -> prev = NULL;
```

```c
        free(ptr);
        printf("\nNode deleted successfully....\n");
    }

}
void deleteLast()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\nUNDERFLOW!!!");
    }
    else if(head->next == NULL)
    {
        head = NULL;
        free(head);
        printf("\nNode deleted successfully...\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != NULL)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = NULL;
        free(ptr);
        printf("\nNode deleted successfully...\n");
    }
}
void deleteLoc()
{
    struct node *ptr, *temp;
    int val;
    printf("\nEnter the data after which the node is to be deleted : ");
    scanf("%d", &val);
    ptr = head;
    while(ptr -> data != val)
    ptr = ptr -> next;
    if(ptr -> next == NULL)
    {
        printf("\nCan't delete....\n");
    }
    else if(ptr -> next -> next == NULL)
    {
        ptr ->next = NULL;
    }
    else
```

```c
    {
        temp = ptr -> next;
        ptr -> next = temp -> next;
        temp -> next -> prev = ptr;
        free(temp);
        printf("\nNode deleted successfully...\n");
    }
}
void printList()
{
    struct node *ptr;
    printf("\nThe Doubly Linked List is\nSTART %c ",29);
    ptr = head;
    while(ptr != NULL)
    {
        printf("%d %c ",ptr->data,29);
        ptr=ptr->next;
    }
    printf("NULL\n\n");
}
void searchList()
{
    struct node *ptr;
    int item,i=0,flag;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        while (ptr!=NULL)
        {
            if(ptr->data == item)
            {
                printf("\nItem %d found at location %d ",item, i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
            }
            i++;
            ptr = ptr -> next;
        }
```

```
        if(flag==1)
        {
            printf("\nItem %d not found\n",item);
        }
    }


}
```
Output:

Practical No.: 4

Program Description: Implementation of Circular Single Linked List using Pointers.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
struct Node;
typedef struct Node * PtrToNode;
typedef PtrToNode List;
typedef PtrToNode Position;

struct Node
{
    int e;
    Position next;
};

void Insert(int x, List l, Position p)
{
    Position TmpCell;
    TmpCell = (struct Node*) malloc(sizeof(struct Node));
    if(TmpCell == NULL)
        printf("Memory out of space\n");
    else
    {
        TmpCell->e = x;
        TmpCell->next = p->next;
        p->next = TmpCell;
    }
}
int isLast(Position p, List l)
{
    return (p->next == l);
}
Position FindPrevious(int x, List l)
{
    Position p = l;
    while(p->next != l && p->next->e != x)
        p = p->next;
    return p;
}

Position Find(int x, List l)
{
    Position p = l->next;
    while(p != l && p->e != x)
        p = p->next;
    return p; }
```

File Submitted by: *Amisha Goswami (0902CS231012)*

```c
void Delete(int x, List l)
{
    Position p, TmpCell;
    p = FindPrevious(x, l);
    if(!isLast(p, l))
    {
        TmpCell = p->next;
        p->next = TmpCell->next;
        free(TmpCell);
    }
    else
        printf("Element does not exist!!!\n");
}

void Display(List l)
{
    printf("The list element are :: ");
    Position p = l->next;
    while(p != l)
    {
        printf("%d -> ", p->e);
        p = p->next;
    }
}

int main()
{
    int x, pos, ch, i;
    List l, l1;
    l = (struct Node *) malloc(sizeof(struct Node));
    l->next = l;
    List p = l;
    printf("CIRCULAR LINKED LIST IMPLEMENTATION OF LIST ADT\n\n");
    do
    {
        printf("\n\n1. INSERT\t 2. DELETE\t 3. FIND\t 4. PRINT\t 5. QUIT\n\nEnter the choice :: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                p = l;
                printf("Enter the element to be inserted :: ");
                scanf("%d",&x);
                printf("Enter the position of the element :: ");
                scanf("%d",&pos);
                for(i = 1; i < pos; i++)
                {
                    p = p->next;
```

```
        }
        Insert(x,l,p);
        break;

    case 2:
        p = l;
        printf("Enter the element to be deleted :: ");
        scanf("%d",&x);
        Delete(x,p);
        break;

    case 3:
        p = l;
        printf("Enter the element to be searched :: ");
        scanf("%d",&x);
        p = Find(x,p);
        if(p == l)
            printf("Element does not exist!!!\n");
        else
            printf("Element exist!!!\n");
        break;

    case 4:
        Display(l);
        break;
    }
}while(ch<5);
return 0;
}
```

Output:

Practical No.: 5

Program Description: Implementation of Circular Doubly Linked List using Pointers.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
struct node
{
    struct node *prev;
    struct node *next;
    int data;
};
struct node *head;
void insertion_beginning();
void insertion_last();
void deletion_beginning();
void deletion_last();
void display();
void search();
int main ()
{
int choice =0;
    while(choice != 9)
    {
        printf("\n********Main Menu********\n");
        printf("\nChoose one option from the following list ...\n");
        printf("\n=============================================\n");
        printf("\n1.Insert in Beginning\n2.Insert at last\n3.Delete from Beginning\n4.Delete from
last\n5.Search\n6.Show\n7.Exit\n");
        printf("\nEnter your choice?\n");
        scanf("\n%d",&choice);
        switch(choice)
        {
            case 1:
            insertion_beginning();
            break;
            case 2:
                insertion_last();
            break;
            case 3:
            deletion_beginning();
            break;
            case 4:
            deletion_last();
            break;
            case 5:
            search();
            break;
```

```c
        case 6:
        display();
        break;
        case 7:
        exit(0);
        break;
        default:
        printf("Please enter valid choice..");
    }
  }
}
void insertion_beginning()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *)malloc(sizeof(struct node));
  if(ptr == NULL)
  {
     printf("\nOVERFLOW");
  }
  else
  {
   printf("\nEnter Item value");
   scanf("%d",&item);
   ptr->data=item;
  if(head==NULL)
  {
    head = ptr;
    ptr -> next = head;
    ptr -> prev = head;
  }
  else
  {
     temp = head;
   while(temp -> next != head)
   {
     temp = temp -> next;
   }
   temp -> next = ptr;
   ptr -> prev = temp;
   head -> prev = ptr;
   ptr -> next = head;
   head = ptr;
  }
  printf("\nNode inserted\n");
}

}
```

```c
void insertion_last()
{
  struct node *ptr,*temp;
  int item;
  ptr = (struct node *) malloc(sizeof(struct node));
  if(ptr == NULL)
  {
    printf("\nOVERFLOW");
  }
  else
  {
    printf("\nEnter value");
    scanf("%d",&item);
    ptr->data=item;
    if(head == NULL)
    {
      head = ptr;
      ptr -> next = head;
      ptr -> prev = head;
    }
    else
    {
      temp = head;
      while(temp->next !=head)
      {
        temp = temp->next;
      }
      temp->next = ptr;
      ptr ->prev=temp;
      head -> prev = ptr;
    ptr -> next = head;
    }
  }
  printf("\nnode inserted\n");
}

void deletion_beginning()
{
  struct node *temp;
  if(head == NULL)
  {
    printf("\n UNDERFLOW");
  }
  else if(head->next == head)
  {
    head = NULL;
    free(head);
    printf("\nnode deleted\n");
```

```c
        }
        else
        {
            temp = head;
            while(temp -> next != head)
            {
                temp = temp -> next;
            }
            temp -> next = head -> next;
            head -> next -> prev = temp;
            free(head);
            head = temp -> next;
        }

}
void deletion_last()
{
    struct node *ptr;
    if(head == NULL)
    {
        printf("\n UNDERFLOW");
    }
    else if(head->next == head)
    {
        head = NULL;
        free(head);
        printf("\nnode deleted\n");
    }
    else
    {
        ptr = head;
        if(ptr->next != head)
        {
            ptr = ptr -> next;
        }
        ptr -> prev -> next = head;
        head -> prev = ptr -> prev;
        free(ptr);
        printf("\nnode deleted\n");
    }
}

void display()
{
    struct node *ptr;
    ptr=head;
    if(head == NULL)
    {
```

```c
            printf("\nnothing to print");
        }
        else
        {
            printf("\n printing values ... \n");

            while(ptr -> next != head)
            {

                printf("%d\n", ptr -> data);
                ptr = ptr -> next;
            }
            printf("%d\n", ptr -> data);
        }

}

void search()
{
    struct node *ptr;
    int item,i=0,flag=1;
    ptr = head;
    if(ptr == NULL)
    {
        printf("\nEmpty List\n");
    }
    else
    {
        printf("\nEnter item which you want to search?\n");
        scanf("%d",&item);
        if(head ->data == item)
        {
        printf("item found at location %d",i+1);
        flag=0;
        }
        else
        {
        while (ptr->next != head)
        {
            if(ptr->data == item)
            {
                printf("item found at location %d ",i+1);
                flag=0;
                break;
            }
            else
            {
                flag=1;
```

```c
        }
        i++;
        ptr = ptr -> next;
    }
    }
    if(flag != 0)
    {
        printf("Item not found\n");
    }
  }

}
```

Output:

Section-B (Stack)

Practical No.: 1

Program Description: Implementation of stack using array.

Solution:

```c
#include<stdio.h>
int stack[100],choice,n,top,x,i;
void push(void);
void pop(void);
void display(void);

void push()
{
   if(top>=n-1)
   {
      printf("\n\tSTACK is over flow");

   }
   else
   {
      printf(" Enter a value to be pushed:");
      scanf("%d",&x);
      top++;
      stack[top]=x;
   }
}
void pop()
{
   if(top<=-1)
   {
      printf("\n\t Stack is under flow");
   }
   else
   {
      printf("\n\t The popped elements is %d",stack[top]);
      top--;
   }
}
void display()
{
   if(top>=0)
   {
      printf("\n The elements in STACK \n");
      for(i=top; i>=0; i--)
         printf("\n%d",stack[i]);
      printf("\n Press Next Choice");
```

```c
        }
    else
    {
        printf("\n The STACK is empty");
    }

}

int main()
{
    //clrscr();
    top=-1;
    printf("\n Enter the size of STACK[MAX=100]:");
    scanf("%d",&n);
    printf("\n\t STACK OPERATIONS USING ARRAY");
    printf("\n\t-------------------------------");
    printf("\n\t 1.PUSH\n\t 2.POP\n\t 3.DISPLAY\n\t 4.EXIT");
    do
    {
        printf("\n Enter the Choice:");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
            {
                push();
                break;
            }
            case 2:
            {
                pop();
                break;
            }
            case 3:
            {
                display();
                break;
            }
            case 4:
            {
                printf("\n\t EXIT POINT ");
                break;
            }
            default:
            {
                printf ("\n\t Please Enter a Valid Choice(1/2/3/4)");
            }
        }
```

```
    }
    while(choice!=4);
    return 0;
}
```

Output:

Program Description: Implementation of Stack using Pointers.

Solution:

```c
#include<stdio.h>
#include<stdlib.h>
struct stackarr;
typedef struct stackarr * Stack;

struct stackarr
{
    int Capacity;
    int TopOfStack;
    int *Array;
};

void MakeEmpty(Stack s)
{
    s->TopOfStack = -1;
}

Stack CreateStack(int MaxElements)
{
    Stack s;
    s = (struct stackarr*) malloc(sizeof(struct stackarr));
    s->Array = (int *)malloc(sizeof(int) * MaxElements);
    s->Capacity = MaxElements;
    MakeEmpty(s);
    return s;
}

void DisposeStack(Stack s)
{
    if(s != NULL)
    {
        free(s->Array);
        free(s);
    }
}

int isFull(Stack s)
{
    return s->TopOfStack == s->Capacity - 1;
}

int isEmpty(Stack s)
{
    return s->TopOfStack == -1;
```

```c
}

void Push(int x, Stack s)
{
    if(isFull(s))
        printf("Full Stack\n\n");
    else
        s->Array[++s->TopOfStack] = x;
}

void Pop(Stack s)
{
    if(isEmpty(s))
        printf("Empty Stack\n\n");
    else
        s->TopOfStack--;
}

int Top(Stack s)
{
    if(isEmpty(s))
        printf("Empty Stack\n\n");
    else
        return s->Array[s->TopOfStack];
}

int TopAndPop(Stack s)
{
    if(isEmpty(s))
        printf("Empty Stack\n\n");
    else
        return s->Array[s->TopOfStack--];
}

void Display(Stack s)
{
    int i;
    if(isEmpty(s))
        printf("Empty Stack\n\n");
    else
    {
        printf("The Stack Elements are :: ");
        for(i=s->TopOfStack; i >= 0; i--)
            printf("%d  ",s->Array[i]);
        printf("\n\n");
    }
}
```

```c
int main()
{
    int n, x, ch, i;
    Stack s;
    printf("Enter the maximum number of elements in the stack :: ");
    scanf("%d", &n);
    s = CreateStack(n);
    printf("ARRAY IMPLEMENTATION OF STACK ADT\n\n");
    do
    {
        printf("\n\n1. PUSH\t 2. POP\t 3.TOP \t 4. TOPANDPOP\t 5. PRINT\t 6. QUIT\n\nEnter the choice :: ");
        scanf("%d", &ch);
        switch(ch)
        {
            case 1:
                printf("Enter the element to be pushed :: ");
                scanf("%d",&x);
                Push(x,s);
                break;

            case 2:
                Pop(s);
                break;

            case 3:
                printf("The Top element in the stack :: %d\n\n", Top(s));
                break;

            case 4:
                printf("The popped top element in the stack :: %d\n\n", TopAndPop(s));
                break;

            case 5:
                Display(s);
                break;
        }
    }while(ch<6);
    DisposeStack(s);
    return 0;
}
```
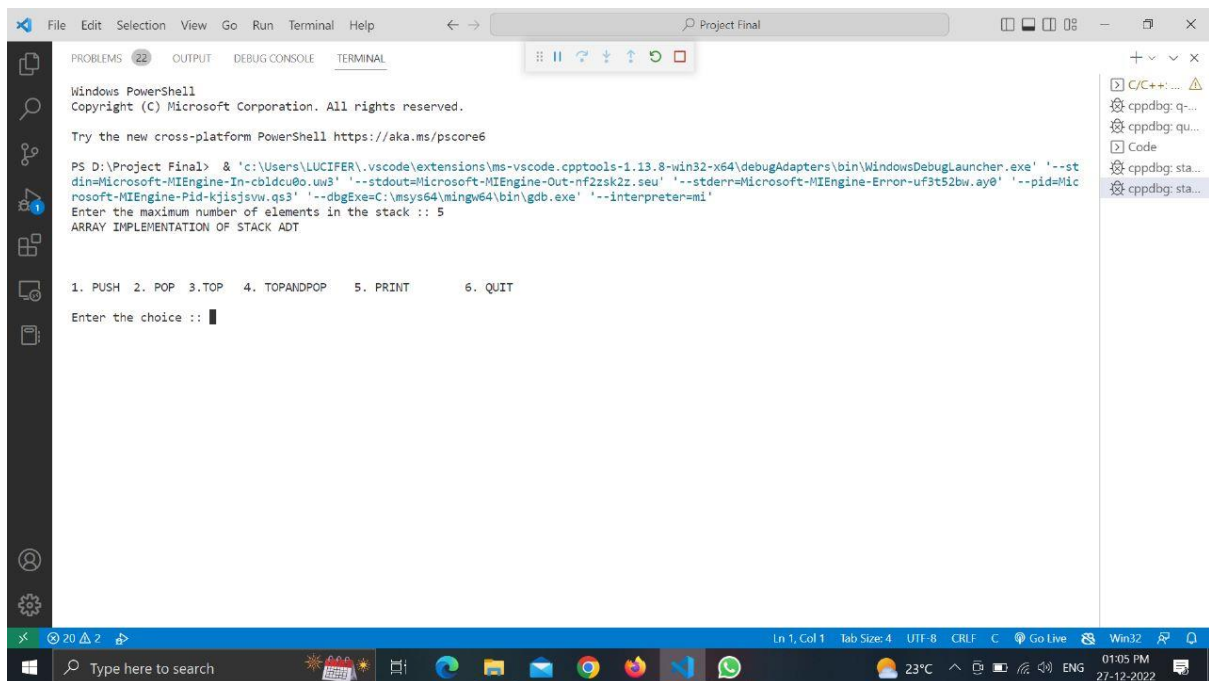
Practical No.: 3
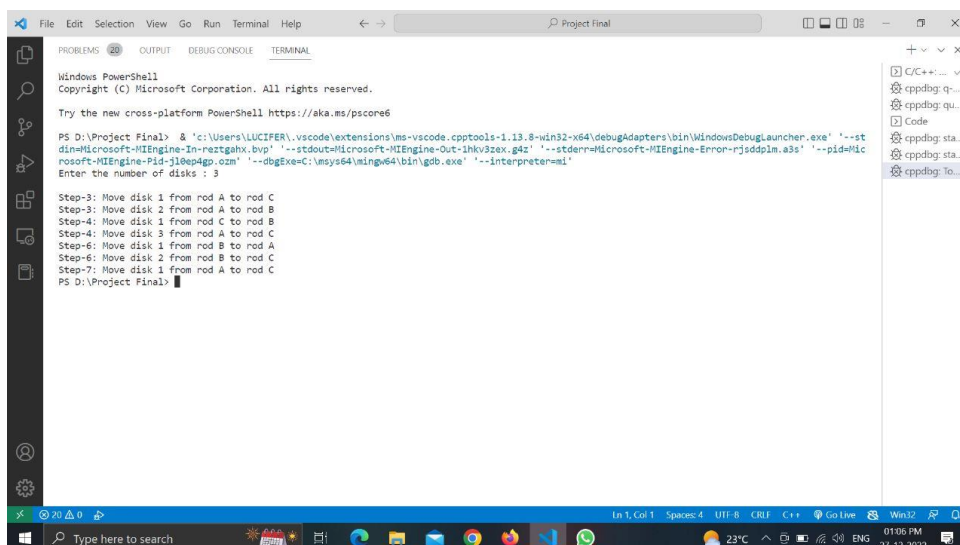
Program Description: Program for Tower of Hanoi using recursion.

Solution:

```c
#include<stdio.h>
int step=0;
// C recursive function to solve tower of hanoi puzzle
void towerOfHanoi(int n, char from_rod, char to_rod, char aux_rod)
{
    step++;
    if (n == 1)
    {
        printf("\nStep-%d: Move disk 1 from rod %c to rod %c", step,from_rod, to_rod);
        return;
    }
    towerOfHanoi(n-1, from_rod, aux_rod, to_rod);
    printf("\nStep-%d: Move disk %d from rod %c to rod %c", step,n, from_rod, to_rod);
    towerOfHanoi(n-1, aux_rod, to_rod, from_rod);
}

int main()
{
    int n; // Number of disks
    printf("Enter the number of disks : ");
    scanf("%d",&n);
    towerOfHanoi(n, 'A', 'C', 'B');  // A, B and C are names of rods
    return 0;
}
```

Output:

File Submitted by: *Amisha Goswami (0902CS231012)*

Program Description: Program to find out factorial of given number using recursion .Also show the various states of stack using in this program.

Solution:
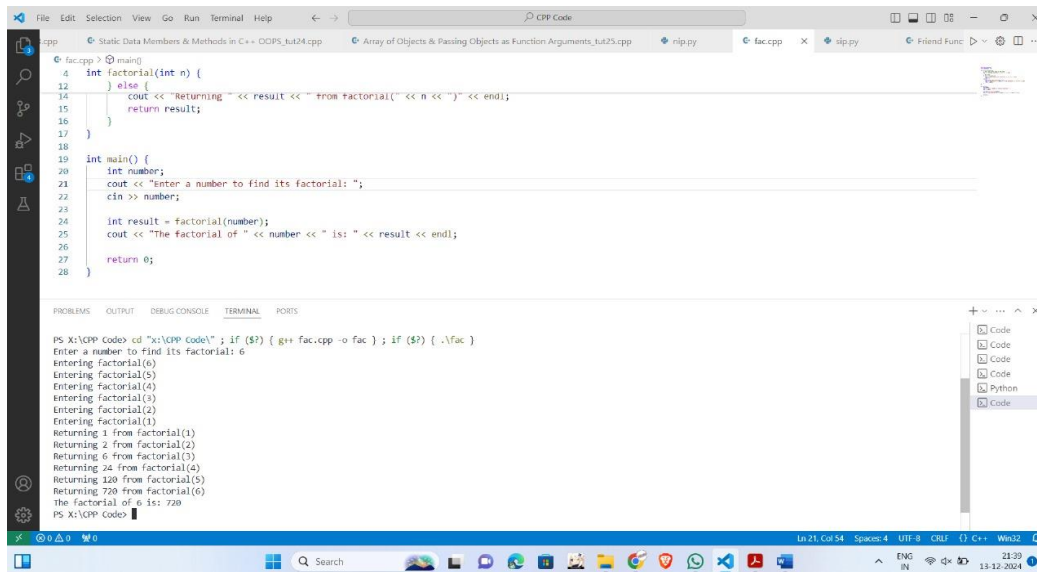
```cpp
#include <iostream>

using namespace std;


int factorial(int n) {
    // Print the current state of the stack
    cout << "Entering factorial(" << n << ")" << endl;
    // Base case
    if (n == 0 || n == 1) {
        cout << "Returning 1 from factorial(" << n << ")" << endl;
        return 1;
    } else {
        int result = n * factorial(n - 1);
        cout << "Returning " << result << " from factorial(" << n << ")" << endl;
        return result;
    }
}
int main() {
    int number;
    cout << "Enter a number to find its factorial: ";
    cin >> number;


    int result = factorial(number);
    cout << "The factorial of " << number << " is: " << result << endl;


    return 0;
}
```

Section-C (Queue)

Practical No.: 1

Program Description: Implementation of Queue using Array.

Solution:

```c
#include<stdio.h>
#define n 5
int main()
{
    int queue[n],ch=1,front=0,rear=0,i,j=1,x=n;
    printf("Queue using Array");
    printf("\n1.Insertion \n2.Deletion \n3.Display \n4.Exit");
    while(ch)
    {
        printf("\nEnter the Choice:");
        scanf("%d",&ch);
        switch(ch)
        {
        case 1:
            if(rear==x)
                printf("\n Queue is Full");
            else
            {
                printf("\n Enter no %d:",j++);
                scanf("%d",&queue[rear++]);
            }
            break;
        case 2:
            if(front==rear)
            {
                printf("\n Queue is empty");
            }
            else
            {
                printf("\n Deleted Element is %d",queue[front++]);
                x++;
            }
            break;
        case 3:
            printf("\nQueue Elements are:\n ");
            if(front==rear)
                printf("\n Queue is Empty");
            else
            {
                for(i=front; i<rear; i++)
                {
```

```c
            printf("%d",queue[i]);
            printf("\n");
        }
        break;
    case 4:
        exit(0);
    default:
        printf("Wrong Choice: please see the options");
    }
  }
 }
 return 0;
}
```

Output:

Program Description: Implementation of queue using pointers.

Solution:

```c
#include<stdio.h>
#include<malloc.h>
typedef struct queue
{
   int data;
   struct queue *next;
}NODE;

NODE * insert(NODE *rear,int data)
{
   NODE *temp;
   temp=(NODE*)malloc(sizeof(NODE));
   temp->data=data;
   temp->next=NULL;
   if(rear==NULL)
   {
     rear=temp;
   }
   else
   {
     rear->next=temp;
     rear=temp;

   }
}

NODE* delete(NODE*front)
{
   NODE *temp;
   if(front==NULL)
   {
     printf("Queue is empty\n");
   }
   else
   {
     temp=front;
     front=front->next;
     free(temp);
   }
   return(front);
}

void display(NODE *front)
{
```
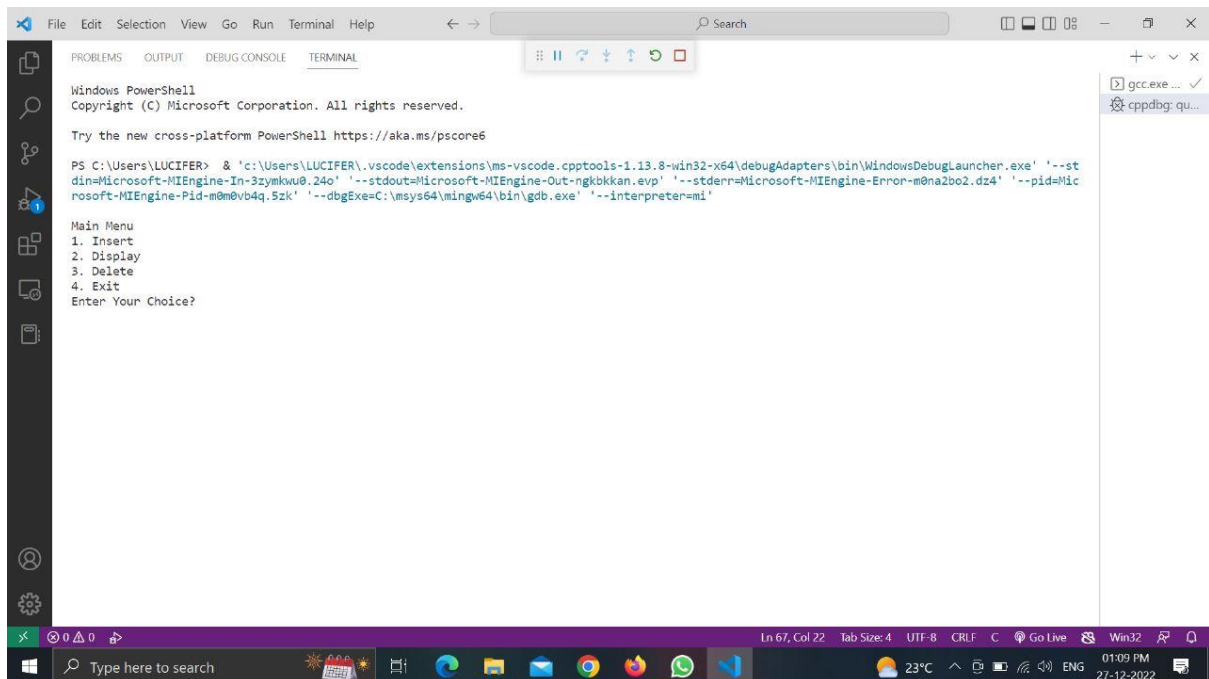
```c
    NODE *t;
    if(front==NULL)
    {
        printf("Queue is empty\n");
    }
    else
    {
        t=front;
        while(t)
        {
            printf("%d,",t->data);
            t=t->next;
        }
    }
}

void main()
{
    NODE *front=NULL,*rear=NULL;
    int ch,data;
    do{
    printf("\nMain Menu\n1] Insert\n2] Display\n3] Delete\n4] Exit\n");
    printf("Enter Ur Choice?");
    scanf("%d",&ch);
    switch(ch)
    {
        case 1:
            printf("Enter Data:");
            scanf("%d",&data);
            rear=insert(rear,data);
            if(front==NULL)
            { front=rear;
            }
            break;
        case 2:
            display(front);
            break;
        case 3:
            front=delete(front);
            if(front==NULL)
            { rear=NULL;
            }
            break;
        case 4:
            printf("Exit");
            break;
        default:
            printf("Wrong Option");   }
```

```c
    printf("\n");
  }while(ch!=4);
}
```

Output:

Practical No.: 3

Program Description: Implementation of Circular Queue using Array.

Solution:

```c
# include<stdio.h>
# define MAX 5

int cqueue_arr[MAX];
int front = -1;
int rear = -1;

/*Begin of insert*/
void insert(int item)
{
    if((front == 0 && rear == MAX-1) || (front == rear+1))
    {
        printf("Queue Overflow \n");
        return;
    }
    if (front == -1)  /*If queue is empty */
    {
        front = 0;
        rear = 0;
    }
    else
    {
        if(rear == MAX-1)   /*rear is at last position of queue */
            rear = 0;
        else
            rear = rear+1;
    }
    cqueue_arr[rear] = item ;
}
/*End of insert*/

/*Begin of del*/
void del()
{
    if (front == -1)
    {
        printf("Queue Underflow\n");
        return ;
    }
    printf("Element deleted from queue is : %d\n",cqueue_arr[front]);
    if(front == rear) /* queue has only one element */
    {
        front = -1;
        rear=-1;}
```

```c
      else
      {
         if(front == MAX-1)
            front = 0;
         else
            front = front+1;
      }
   }
}
/*End of del() */

/*Begin of display*/
void display()
{
   int front_pos = front,rear_pos = rear;
   if(front == -1)
   {
      printf("Queue is empty\n");
      return;
   }
   printf("Queue elements :\n");
   if( front_pos <= rear_pos )
      while(front_pos <= rear_pos)
      {
         printf("%d ",cqueue_arr[front_pos]);
         front_pos++;
      }
   else
   {
      while(front_pos <= MAX-1)
      {
         printf("%d ",cqueue_arr[front_pos]);
         front_pos++;
      }
      front_pos = 0;
      while(front_pos <= rear_pos)
      {
         printf("%d ",cqueue_arr[front_pos]);
         front_pos++;
      }
   }
   printf("\n");
}
/*End of display*/

/*Begin of main*/
int main()
{
   int choice,item;
```

```c
    do
    {
        printf("1.Insert\n");
        printf("2.Delete\n");
        printf("3.Display\n");
        printf("4.Quit\n");

        printf("Enter your choice : ");
        scanf("%d",&choice);

        switch(choice)
        {
            case 1 :
                printf("Input the element for insertion in queue : ");
                scanf("%d", &item);

                insert(item);
                break;
            case 2 :
                del();
                break;
            case 3:
                display();
                break;
            case 4:
                break;
                default:
                printf("Wrong choice\n");
        }
    }while(choice!=4);

    return 0;
}
/*End of main*/
```
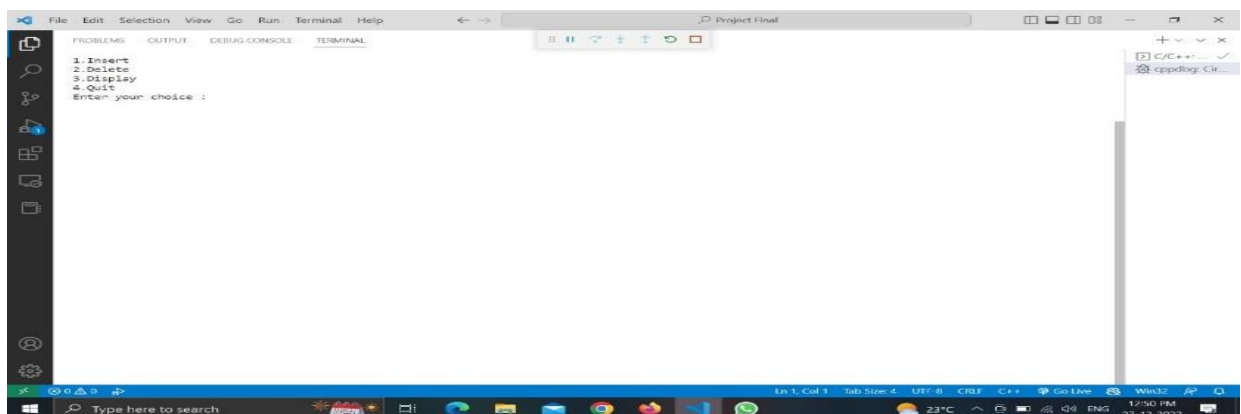
Output:

File Submitted by: *Amisha Goswami (0902CS231012)*

Section-D (Trees)

Practical No.: 1

Program Description: Implementation of Binary Search Tree.

Solution:

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

typedef struct treeNode
{
    int data;
    struct treeNode *left;
    struct treeNode *right;

}treeNode;

treeNode* FindMin(treeNode *node)
{
    if(node==NULL)
    {
        return NULL;
    }
    if(node->left)
        return FindMin(node->left);
    else
        return node;
}
treeNode* FindMax(treeNode *node)
{
    if(node==NULL)
  {
    return NULL;
  }
  if(node->right)
    FindMax(node->right);
  else
    return node;
}

treeNode * Insert(treeNode *node,int data)
{
    if(node==NULL)
    {
        treeNode *temp;
        temp = (treeNode *)malloc(sizeof(treeNode));
        temp -> data = data;
```

```c
    temp -> left = temp -> right = NULL;
        return temp;
    }


    if(data >(node->data))
    {
        node->right = Insert(node->right,data);
    }
    else if(data < (node->data))
    {
        node->left = Insert(node->left,data);
    }
    printf("\nInsertion Successful....\n");
    return node;
}

treeNode * Delete(treeNode *node, int data)
{
    treeNode *temp;
    if(node==NULL)
    {
        printf("Element Not Found");
    }
    else if(data < node->data)
    {
        node->left = Delete(node->left, data);
    }
    else if(data > node->data)
    {
        node->right = Delete(node->right, data);
    }
    else
    {
        if(node->right && node->left)
        {
            temp = FindMin(node->right);
            node -> data = temp->data;
        node -> right = Delete(node->right,temp->data);
        }
        else
        {
            temp = node;
            if(node->left == NULL)
                node = node->right;
            else if(node->right == NULL)
                node = node->left;
            free(temp);
        }    }
```

```c
        return node;

}

treeNode * Find(treeNode *node, int data)
{
    if(node==NULL)
    }
        return NULL;
    }
    if(data > node->data)
    {
        return Find(node->right,data);
    }
    else if(data < node->data)
    {
        return Find(node->left,data);
    }
    else
    {
        return node;
    }
}



int main()
{
    treeNode *root = NULL;
    treeNode * temp;
    int choice,val;
    while(1)
    {
        printf("\nTree Menu");
        printf("\n1. Insert Node");
        printf("\n2. Delete Node");
        printf("\n3. Search an Element");
        printf("\n4. Find Minimum Element");
        printf("\n5. Find Maximum Element");
        printf("\n6. Exit");
        printf("\n\nEnter Your Choice (0-9) : ");
        scanf("%d",&choice);
        switch(choice)
        {
            case 1:
                printf("\nInserting Node in Binary Search Tree\n");
                printf("Enter value to insert in new node : ");
                scanf("%d",&val);
                root=Insert(root,val);
```
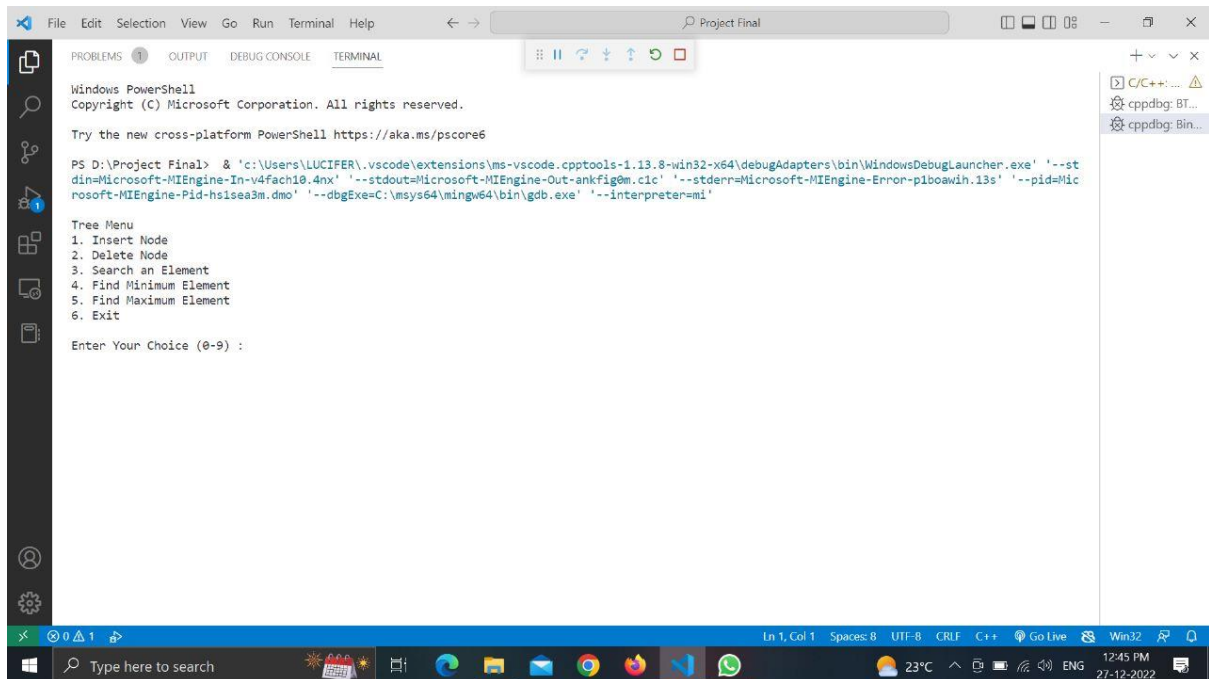
```c
            break;
        case 2:
            printf("\nDeleting a Node in Binary Search Tree\n");
            printf("Enter value of node to Delete : ");
            scanf("%d",&val);
            root=Delete(root,val);
            break;
        case 3:
            printf("\nSearch a Node");
            printf("\nEnter value to search : ");
            scanf("%d",&val);
            temp = Find(root,val);
            if(temp==NULL)
            {
                printf("Element %d not found\n",val);
            }
            else
            {
                printf("Element %d Found \n",val);
            }
            break;
        case 4:
            temp = FindMin(root);
            printf("Minimum element is %d\n",temp->data);
            break;
        case 5:
            temp = FindMax(root);
            printf("Maximum element is %d\n",temp->data);
            break;
        case 6:
            printf("\nThanks for using Tree Program...\n");
            exit(1);
            break;
        default:
            printf("\nInvalid Choice. Please Try Again....\n");
            break;
        }
    }
}
```

Practical No.: 2

Program Description: Conversion of BST PreOrder/PostOrder/InOrder.

Solution:

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

typedef struct treeNode
{
    int data;
    struct treeNode *left;
    struct treeNode *right;

}treeNode;

treeNode* FindMin(treeNode *node)
{
    if(node==NULL)
    {

        return NULL;
    }
    if(node->left)
        return FindMin(node->left);
    else
        return node;
}
treeNode* FindMax(treeNode *node)
{
    if(node==NULL)
  {

    return NULL;
  }
  if(node->right)
    FindMax(node->right);
  else
    return node;
}

treeNode * Insert(treeNode *node,int data)
{
    if(node==NULL)
    {
        treeNode *temp;
        temp = (treeNode *)malloc(sizeof(treeNode));
```

```c
        temp -> data = data;
        temp -> left = temp -> right = NULL;     return temp;
    }
    if(data >(node->data))
    {
        node->right = Insert(node->right,data);
    }
    else if(data < (node->data))
    {
        node->left = Insert(node->left,data);
    }
    printf("\nInsertion Successful....\n");
    return node;
}

treeNode * Delete(treeNode *node, int data)
{
    treeNode *temp;
    if(node==NULL)
    {
        printf("Element Not Found");
    }
    else if(data < node->data)
    {
        node->left = Delete(node->left, data);
    }
    else if(data > node->data)
    {
        node->right = Delete(node->right, data);
    }
    else
    {
        if(node->right && node->left)
        {
            temp = FindMin(node->right);
            node -> data = temp->data;
            node -> right = Delete(node->right,temp->data);
        }
        else
        {
            temp = node;
            if(node->left == NULL)
                node = node->right;
            else if(node->right == NULL)
                node = node->left;
            free(temp);
        }
    }
```

```c
    return node;}

treeNode * Find(treeNode *node, int data)
{
    if(node==NULL)
    {
        return NULL;
    }
    if(data > node->data)
    }
        return Find(node->right,data);
    }
    else if(data < node->data)
    {
        return Find(node->left,data);
    }
    else
    {
        return node;
    }
}

void PrintInorder(treeNode *node)
{
    if(node==NULL)
    {
        return;
    }
    PrintInorder(node->left);
    printf("%d ",node->data);
    PrintInorder(node->right);
}

void PrintPreorder(treeNode *node)
{
    if(node==NULL)
    {
        return;
    }
    printf("%d ",node->data);
    PrintPreorder(node->left);
    PrintPreorder(node->right);
}

void PrintPostorder(treeNode *node)
{
    if(node==NULL)
    {
```

```c
            return;
        }
        PrintPostorder(node->left);
        PrintPostorder(node->right);
        printf("%d ",node->data);
}

int main()
{
    treeNode *root = NULL;
    treeNode * temp;
    int choice,val;
    while(1)
    {
            printf("\nTree Menu");
            printf("\n1. Insert Node");
            printf("\n2. Delete Node");
            printf("\n3. Pre Order Traversal");
            printf("\n4. Post Order Traversal");
            printf("\n5. In Order Traversal");
            printf("\n6. All Traversal");
            printf("\n7. Search an Element");
            printf("\n8. Find Minimum Element");
            printf("\n9. Find Maximum Element");
            printf("\n0. Exit");
            printf("\n\nEnter Your Choice (0-9) : ");
            scanf("%d",&choice);
            switch(choice)
            {
                case 1:
                    printf("\nInserting Node in Binary Search Tree\n");
                    printf("Enter value to insert in new node : ");
                    scanf("%d",&val);
                    root=Insert(root,val);
                    break;
                case 2:
                    printf("\nDeleting a Node in Binary Search Tree\n");
                    printf("Enter value of node to Delete : ");
                    scanf("%d",&val);
                    root=Delete(root,val);
                    break;
                case 3:
                    PrintPreorder(root);
                    break;
                case 4:
                    PrintPostorder(root);
                    break;
                case 5:
```
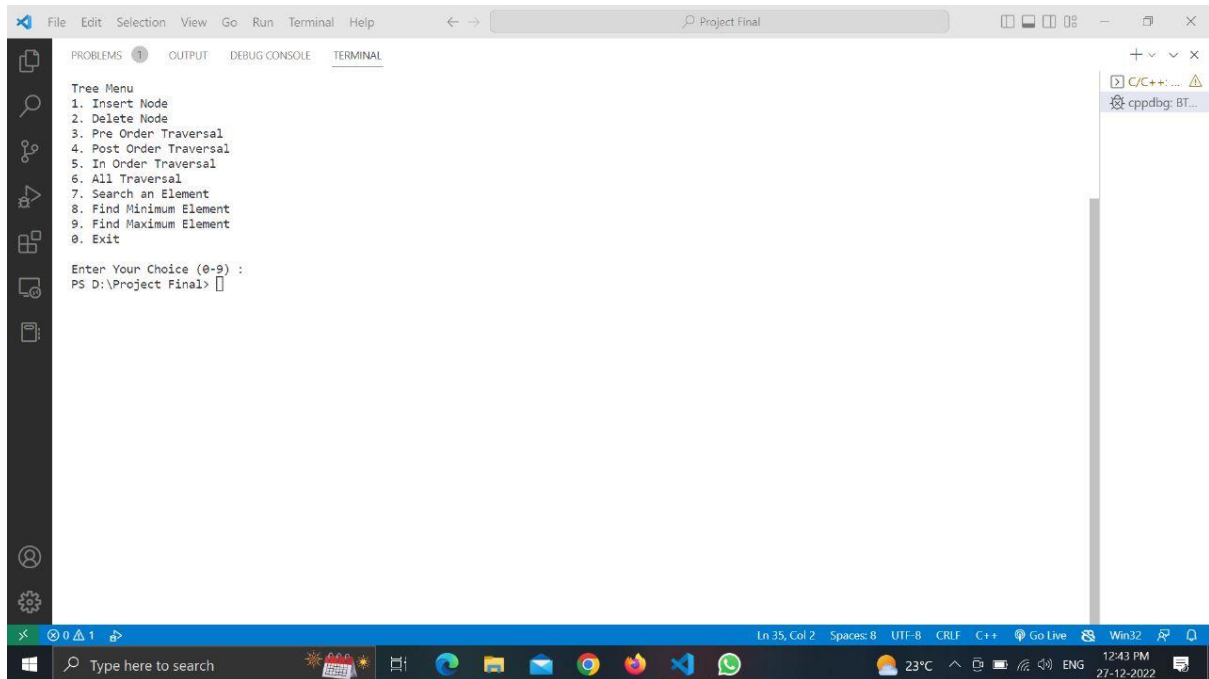
```c
            PrintInorder(root);
            break;
        case 6:
            printf("\nPre Order Traversal:");
            PrintPreorder(root);
            printf("\nIn Order Traversal:");
            PrintInorder(root);
            printf("\nPost Order Traversal:");
            PrintPostorder(root);
            break;
        case 7:
            printf("\nSearch a Node");
            printf("\nEnter value to search : ");
            scanf("%d",&val);
            temp = Find(root,val);
            if(temp==NULL)
            {
                printf("Element %d not found\n",val);
            }
            else
            {
                printf("Element %d Found \n",val);
            }
            break;
        case 8:
            temp = FindMin(root);
            printf("Minimum element is %d\n",temp->data);
            break;
        case 9:
            temp = FindMax(root);
            printf("Maximum element is %d\n",temp->data);
            break;
        case 0:
            printf("\nThanks for using Tree Program...\n");
            exit(1);
            break;
        default:
            printf("\nInvalid Choice. Please Try Again....\n");
            break;
        }
    }
}
```

Output:

File Submitted by: *Amisha Goswami (0902CS231012)*

Practical No.: 3

Program Description: Implementation of Kruskal Algorithm.

Solution:

```cpp
#include <iostream>

#include <vector>

#include <algorithm>

using namespace std;

struct Edge {

    int src, dest, weight;

};

struct Subset {

    int parent;

    int rank;

};

bool compareEdges(Edge a, Edge b) {

    return a.weight < b.weight;

}


int find(Subset subsets[], int i) {

    if (subsets[i].parent != i) {

        subsets[i].parent = find(subsets, subsets[i].parent);

    }

    return subsets[i].parent;

}

void unionSets(Subset subsets[], int x, int y) {

    int xroot = find(subsets, x);

    int yroot = find(subsets, y);

    if (subsets[xroot].rank < subsets[yroot].rank) {

        subsets[xroot].parent = yroot;

    } else if (subsets[xroot].rank > subsets[yroot].rank) {

        subsets[yroot].parent = xroot;
```

```cpp
    } else {
        subsets[yroot].parent = xroot;
        subsets[xroot].rank++;
    }
}
 void kruskalMST(vector<Edge>& edges, int V) {
    vector<Edge> result; // To store the resultant MST
    int e = 0; // Initial count of edges in MST
    sort(edges.begin(), edges.end(), compareEdges);
    Subset* subsets = new Subset[V];
    for (int v = 0; v < V; v++) {
        subsets[v].parent = v;
        subsets[v].rank = 0;
    }
    for (auto edge : edges) {
        int x = find(subsets, edge.src);
        int y = find(subsets, edge.dest);

        if (x != y) {
            result.push_back(edge);
            unionSets(subsets, x, y);
            e++;
        }
        if (e == V - 1) {
            break;
        }
    }
    cout << "Edges in the Minimum Spanning Tree:\n";
    for (auto edge : result) {
        cout << edge.src << " -- " << edge.dest << " == " << edge.weight << endl;
    }
```

```
        delete[] subsets;

}

int main() {

    int V = 4; // Number of vertices

    vector<Edge> edges = {

        {0, 1, 10},

        {0, 2, 6},

        {0, 3, 5},

        {1, 3, 15},

        {2, 3, 4}

    };

    kruskalMST(edges, V);


    return 0;

}
```

OUTPUT:
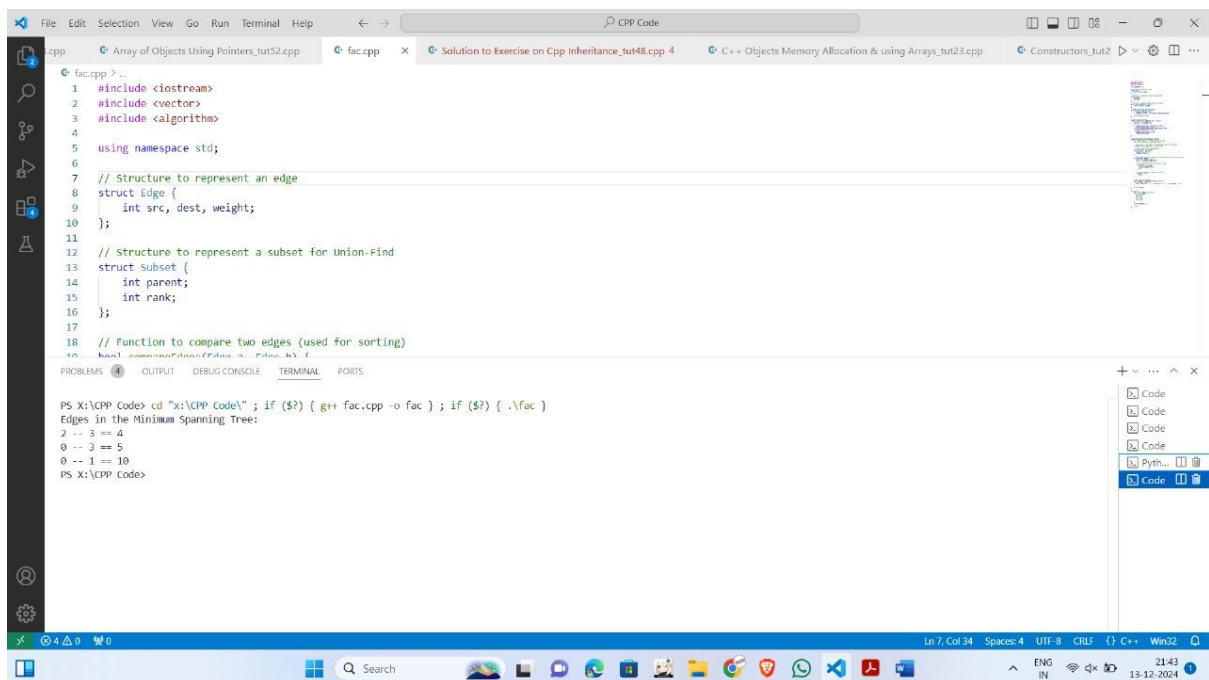
Practical No.: 4

Program Description: Implementation of prim Algorithm.

Solution:

```cpp
#include <iostream>

#include <vector>

#include <climits>

using namespace std;

int minKey(const vector<int>& key, const vector<bool>& mstSet, int V) {

  int min = INT_MAX, min_index;

  for (int v = 0; v < V; v++) {

    if (!mstSet[v] && key[v] < min) {

      min = key[v];

      min_index = v;

    }

  }

  return min_index;

}

void primMST(const vector<vector<int>>& graph, int V) {

  vector<int> parent(V);

  vector<int> key(V, INT_MAX);

  vector<bool> mstSet(V, false);

  key[0] = 0;

  parent[0] = -1; T

  for (int count = 0; count < V - 1; count++) {

    int u = minKey(key, mstSet, V);

    mstSet[u] = true;

    for (int v = 0; v < V; v++) {

      if (graph[u][v] && !mstSet[v] && graph[u][v] < key[v]) {

        parent[v] = u;

        key[v] = graph[u][v];

      }  }
```

```cpp
    cout << "Edge \tWeight\n";

    for (int i = 1; i < V; i++) {

        cout << parent[i] << " -- " << i << "\t" << graph[i][parent[i]] << endl;

    }

}

int main() {

    int V = 5;

    vector<vector<int>> graph = {

        {0, 2, 0, 6, 0},

        {2, 0, 3, 8, 5},

        {0, 3, 0, 0, 7},

        {6, 8, 0, 0, 9},

        {0, 5, 7, 9, 0}

    };

    primMST(graph, V);

    return 0;

}
```

OUTPUT:

File Submitted by: *Amisha Goswami (0902CS231012)*

Practical No.: 5

Program Description: Implementation of Dijkstra algorithm.

Solution:

```cpp
#include <iostream>

using namespace std;

#include <limits.h>

#define V 9

int minDistance(int dist[], bool sptSet[])

{

    int min = INT_MAX, min_index;

    for (int v = 0; v < V; v++)

        if (sptSet[v] == false && dist[v] <= min)

            min = dist[v], min_index = v;

    return min_index;

}

void printSolution(int dist[])

{

    cout << "Vertex \t Distance from Source" << endl;

    for (int i = 0; i < V; i++)

        cout << i << " \t\t\t" << dist[i] << endl;

}

void dijkstra(int graph[V][V], int src)

{

    int dist[V];

    bool sptSet[V];

    for (int i = 0; i < V; i++)

        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {

        int u = minDistance(dist, sptSet);

        sptSet[u] = true;
```

```cpp
        for (int v = 0; v < V; v++)

            if (!sptSet[v] && graph[u][v]

                && dist[u] != INT_MAX

                && dist[u] + graph[u][v] < dist[v])

                dist[v] = dist[u] + graph[u][v];

    }

    printSolution(dist);

}

int main()

{

    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },

                { 4, 0, 8, 0, 0, 0, 0, 11, 0 },

                { 0, 8, 0, 7, 0, 4, 0, 0, 2 },

                { 0, 0, 7, 0, 9, 14, 0, 0, 0 },

                { 0, 0, 0, 9, 0, 10, 0, 0, 0 },

                { 0, 0, 4, 14, 10, 0, 2, 0, 0 },

                { 0, 0, 0, 0, 0, 2, 0, 1, 6 },

                { 8, 11, 0, 0, 0, 0, 1, 0, 7 },

                { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);

    return 0;

}

path

#include <iostream>

using namespace std;

#include <limits.h>

#define V 9

int minDistance(int dist[], bool sptSet[])

{

    int min = INT_MAX, min_index;
```

```cpp
    for (int v = 0; v < V; v++)

        if (sptSet[v] == false && dist[v] <= min)

            min = dist[v], min_index = v;

    return min_index;

}

void printSolution(int dist[])

{

    cout << "Vertex \t Distance from Source" << endl;

    for (int i = 0; i < V; i++)

        cout << i << " \t\t\t" << dist[i] << endl;

}

void dijkstra(int graph[V][V], int src)

{

    int dist[V

    for (int i = 0; i < V; i++)

        dist[i] = INT_MAX, sptSet[i] = false;

    dist[src] = 0;

    for (int count = 0; count < V - 1; count++) {.

        int u = minDistance(dist, sptSet);

        sptSet[u] = true;.

        for (int v = 0; v < V; v++)

            if (!sptSet[v] && graph[u][v]

                && dist[u] != INT_MAX

                && dist[u] + graph[u][v] < dist[v])

                dist[v] = dist[u] + graph[u][v];

    }

    printSolution(dist);

}

int main()

{

    int graph[V][V] = { { 0, 4, 0, 0, 0, 0, 0, 8, 0 },
```

```
            { 4, 0, 8, 0, 0, 0, 0, 11, 0 },

            { 0, 8, 0, 7, 0, 4, 0, 0, 2 },

            { 0, 0, 7, 0, 9, 14, 0, 0, 0 },

            { 0, 0, 0, 9, 0, 10, 0, 0, 0 },

            { 0, 0, 4, 14, 10, 0, 2, 0, 0 },

            { 0, 0, 0, 0, 0, 2, 0, 1, 6 },

            { 8, 11, 0, 0, 0, 0, 1, 0, 7 },

            { 0, 0, 2, 0, 0, 0, 6, 7, 0 } };

    dijkstra(graph, 0);


    return 0;
}
```

OUTPUT:

Section-E (Sorting & Searching)

Practical No.: 1

Program Description: Implementation of sorting.

(A).Bubble

(B).Selection

(C).Insertion

(D).Quick

(E).Merge

Solution:

(A).Bubble

```cpp
#include <bits/stdc++.h>

using namespace std;

void bubbleSort(vector<int>& arr) {

  int n = arr.size();

  bool swapped;


  for (int i = 0; i < n - 1; i++) {

    swapped = false;

    for (int j = 0; j < n - i - 1; j++) {

      if (arr[j] > arr[j + 1]) {

        swap(arr[j], arr[j + 1]);

        swapped = true;

      }

    }

    if (!swapped)

      break;

  }

}

void printVector(const vector<int>& arr) {

  for (int num : arr)
```

```cpp
    cout << " " << num;

}

int main() {

    vector<int> arr = { 64, 34, 25, 12, 22, 11, 90 };

    bubbleSort(arr);

    cout << "Sorted array: \n";

    printVector(arr);

    return 0;

}
```

OUTPUT:

(B). Selection
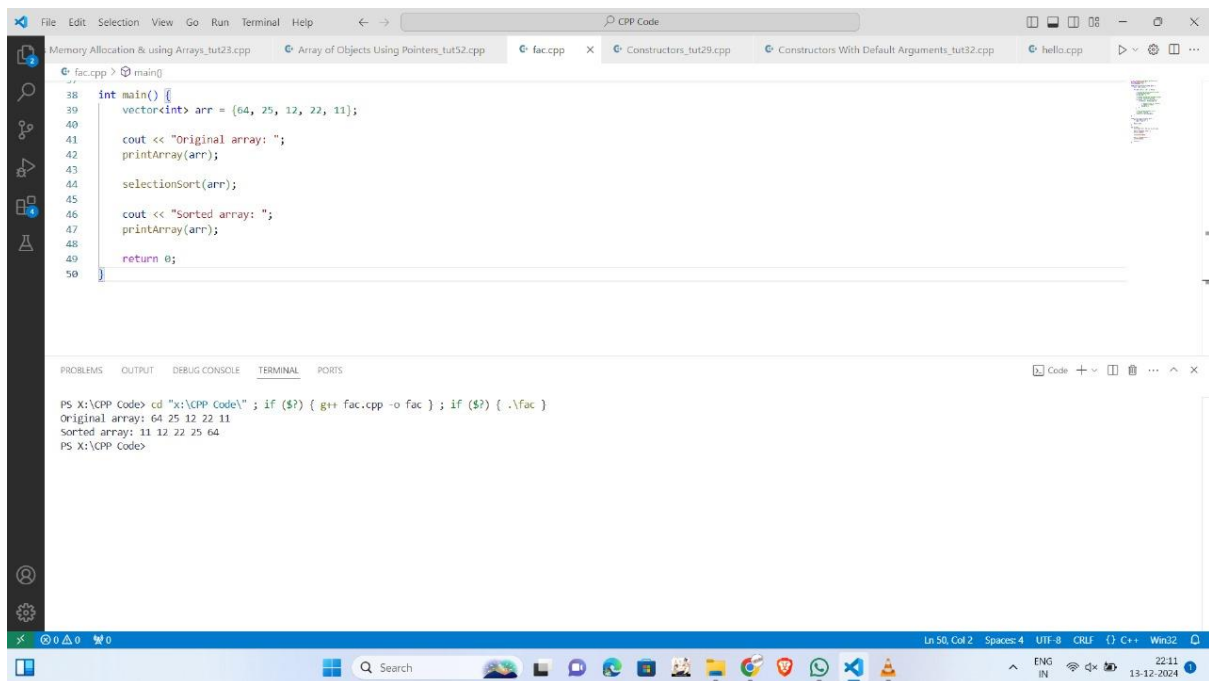
```cpp
#include <bits/stdc++.h>
using namespace std;

void selectionSort(vector<int> &arr) {
    int n = arr.size();
    for (int i = 0; i < n - 1; ++i) {
        int min_idx = i;
        for (int j = i + 1; j < n; ++j) {
            if (arr[j] < arr[min_idx]) {

                // Update min_idx if a smaller
                // element is found
                min_idx = j;        }
        }
        swap(arr[i], arr[min_idx]);  }
}
void printArray(vector<int> &arr) {
    for (int &val : arr) {
        cout << val << " ";
    }
    cout << endl;
}
int main() {
    vector<int> arr = {64, 25, 12, 22, 11};
    cout << "Original array: ";
    printArray(arr);
    selectionSort(arr);
    cout << "Sorted array: ";
    printArray(arr);
    return 0; }
```

(C). Insertion

```cpp
#include <iostream>
using namespace std;
void insertionSort(int arr[], int n)
{
    for (int i = 1; i < n; ++i) {
        int key = arr[i];
        int j = i - 1;
        while (j >= 0 && arr[j] > key) {
            arr[j + 1] = arr[j];
            j = j - 1;
        }
        arr[j + 1] = key;
    }
}
void printArray(int arr[], int n)
{
    for (int i = 0; i < n; ++i)
        cout << arr[i] << " ";
    cout << endl;
}
int main()
{
    int arr[] = { 12, 11, 13, 5, 6 };
    int n = sizeof(arr) / sizeof(arr[0]);
    insertionSort(arr, n);
    printArray(arr, n);
    return 0;
}
```

(D).Quick

```cpp
#include <bits/stdc++.h>
using namespace std;
int partition(vector<int>& arr, int low, int high) {
    int pivot = arr[high];

    int i = low - 1;
    for (int j = low; j <= high - 1; j++) {
        if (arr[j] < pivot) {
            i++;
            swap(arr[i], arr[j]);
        }
    }
    swap(arr[i + 1], arr[high]);
    return i + 1;
}
void quickSort(vector<int>& arr, int low, int high) {
    if (low < high) {
        int pi = partition(arr, low, high);
        quickSort(arr, low, pi - 1);
        quickSort(arr, pi + 1, high);  }}
int main() {
    vector<int> arr = {10, 7, 8, 9, 1, 5};
    int n = arr.size();
    quickSort(arr, 0, n - 1);

    for (int i = 0; i < n; i++) {
        cout << arr[i] << " ";   }
 return 0; }
```
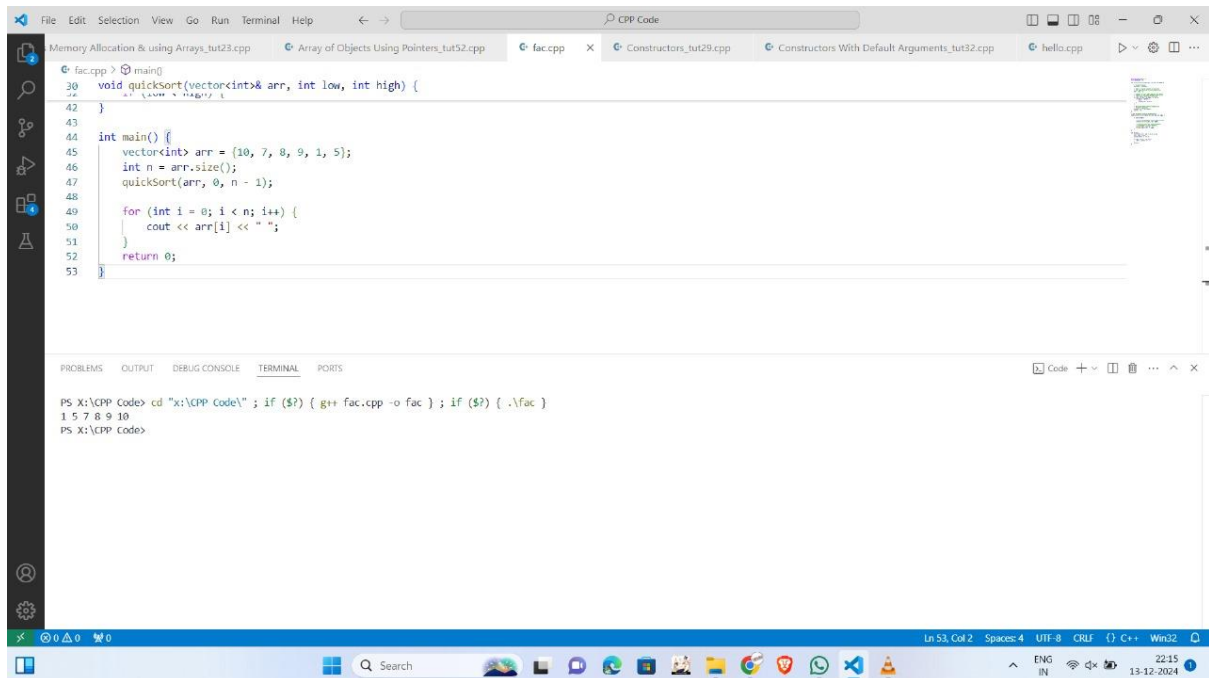
(E).Merge

```cpp
#include <bits/stdc++.h>
using namespace std;
void merge(vector<int>& arr, int left,
           int mid, int right) {
  int n1 = mid - left + 1;
  int n2 = right - mid;
  vector<int> L(n1), R(n2);
  for (int i = 0; i < n1; i++)
    L[i] = arr[left + i];
  for (int j = 0; j < n2; j++)
    R[j] = arr[mid + 1 + j];
  int i = 0, j = 0;
  int k = left;
  while (i < n1 && j < n2) {
    if (L[i] <= R[j]) {
      arr[k] = L[i];
      i++;
    }     else {
      arr[k] = R[j];
      j++; }
    k++;   }
  while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;   }
  while (j < n2) {
  arr[k] = R[j];
    j++;
    k++;
  }
```

File Submitted by: *Amisha Goswami (0902CS231012)*

```cpp
}
void mergeSort(vector<int>& arr, int left, int right)
{
    if (left >= right)
        return;
    int mid = left + (right - left) / 2;
    mergeSort(arr, left, mid);
    mergeSort(arr, mid + 1, right);
    merge(arr, left, mid, right);
}
void printVector(vector<int>& arr)
{
    for (int i = 0; i < arr.size(); i++)
        cout << arr[i] << " ";
    cout << endl;
}
int main()
{
    vector<int> arr = { 12, 11, 13, 5, 6, 7 };
    int n = arr.size();
    cout << "Given vector is \n";
    printVector(arr);
    mergeSort(arr, 0, n - 1);
    cout << "\nSorted vector is \n";
    printVector(arr);
    return 0;
}
```

Practical No.: 2

Program Description: Implementation of Binary Search on a list of numbers stored in an Array.

Solution:

```cpp
#include <iostream>
using namespace std;
#include <iostream>
using namespace std;
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2
        if (arr[mid] == target) {
            return mid;
        }
        else if (arr[mid] < target) {
            left = mid + 1;
        }
        else {
            right = mid - 1;
        }
    }
    return -1;
}
void printArray(int arr[], int size) {
    for (int i = 0; i < size; i++) {
        cout << arr[i] << " ";
    }
    cout << endl;
}
```

```cpp
int main() {
    int arr[] = {2, 3, 4, 10, 40};
    int size = sizeof(arr) / sizeof(arr[0]);
    int target = 10;
    cout << "Array: ";
    printArray(arr, size);
    int result = binarySearch(arr, size, target);
    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found in the array." << endl;  }
    return 0; }
int binarySearch(int arr[], int size, int target) {
    int left = 0;
    int right = size - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == target) {
            return mid;
        }
        else if (arr[mid] < target) {
            left = mid + 1;      }
            else {
            right = mid − 1 }
    return -1; }
void printArray(int arr[], int size) {
    for (int I = 0; I < size; I++) {
        cout << arr[I] << " "; }
    cout << endl; }
int main() {
    int arr[] = {2, 3, 4, 10, 40;
```
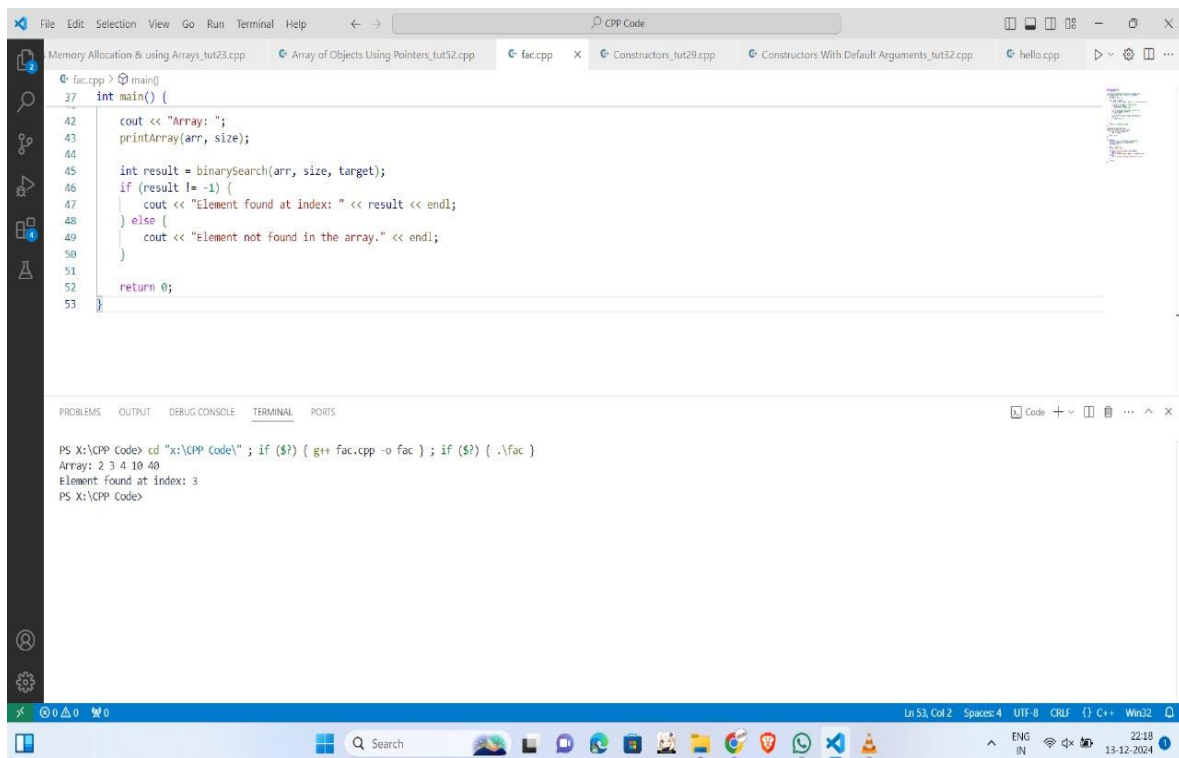
```cpp
    int size = sizeof(arr) / sizeof(arr[0]);

    int target = 10;

    cout << "Array: ";

    printArray(arr, size);

    int result = binarySearch(arr, size, target);

    if (result != -1) {

        cout << "Element found at index: " << result << endl;

    } else {

        cout << "Element not found in the array." << endl;

    }    return 0; }
```

OUTPUT:

Program Description: Implementation of Binary Search on a list of strings

stored in an Array

Solution:

```cpp
#include <iostream>

using namespace std;

#include <iostream>

using namespace std;

int binarySearch(int arr[], int size, int target) {

    int left = 0;

    int right = size - 1;

    while (left <= right) {

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

        if (arr[mid] == target) {

            return mid;

        }

        else if (arr[mid] < target) {

            left = mid + 1;

        }

        else {

            right = mid - 1;

        }

    }

    return -1;

}

void printArray(int arr[], int size) {

    for (int i = 0; i < size; i++) {

        cout << arr[i] << " ";

    }

    cout << endl; }

int main() {
```

```cpp
    int arr[] = {2, 3, 4, 10, 40;

    int size = sizeof(arr) / sizeof(arr[0]);

    int target = 10;

    cout << "Array: ";

    printArray(arr, size);

    int result = binarySearch(arr, size, target);

    if (result != -1) {

        cout << "Element found at index: " << result << endl;

    } else {

        cout << "Element not found in the array." << endl;

    }

    return 0;

}

int binarySearch(int arr[], int size, int target) {

    int left = 0;

    int right = size - 1;

    while (left <= right) {

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

        if (arr[mid] == target) {

            return mid;

        }

        else if (arr[mid] < target) {

            left = mid + 1;

        }

        else {

            right = mid - 1;      } }

    return -1; }

void printArray(int arr[], int size) {

    for (int I = 0; I < size; I++) {

        cout << arr[I] << " ";

    }
```

```cpp
    cout << endl;

}

int main() {

    int arr[] = {2, 3, 4, 10, 40};

    int size = sizeof(arr) / sizeof(arr[0]);

    int target = 10;

    cout << "Array: ";

    printArray(arr, size);

    int result = binarySearch(arr, size, target);

    if (result != -1) {

        cout << "Element found at index: " << result << endl;

    } else {

        cout << "Element not found in the array." << endl;

    }

    return 0; }
```
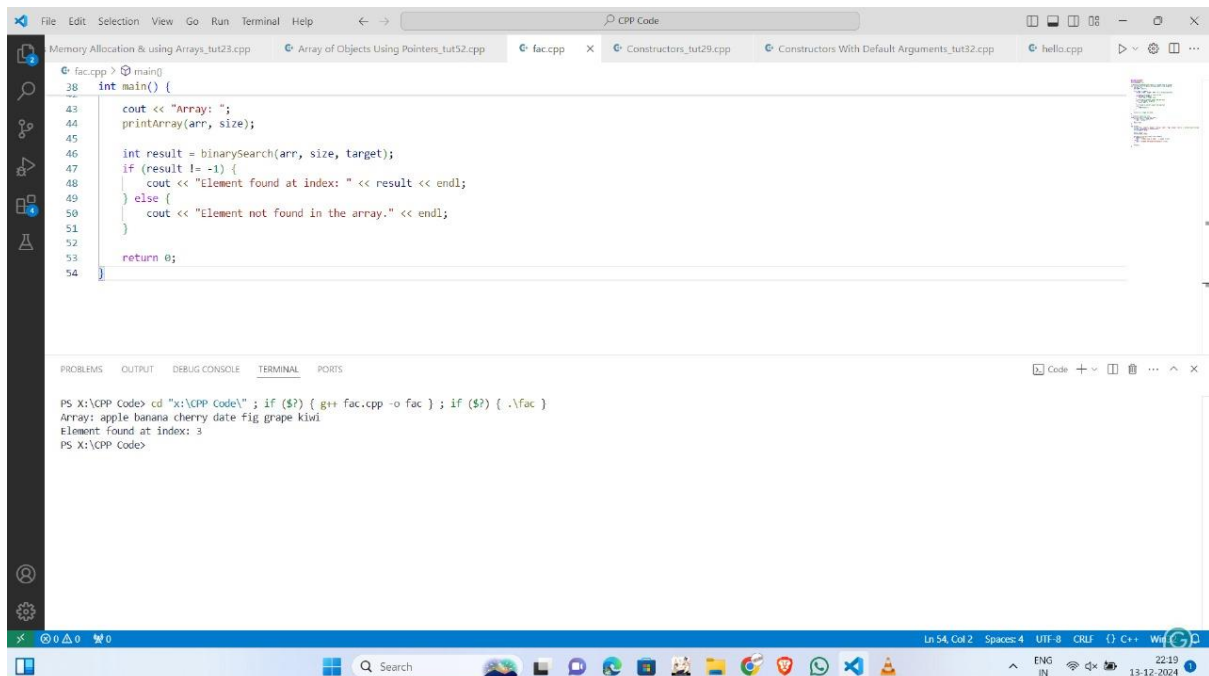
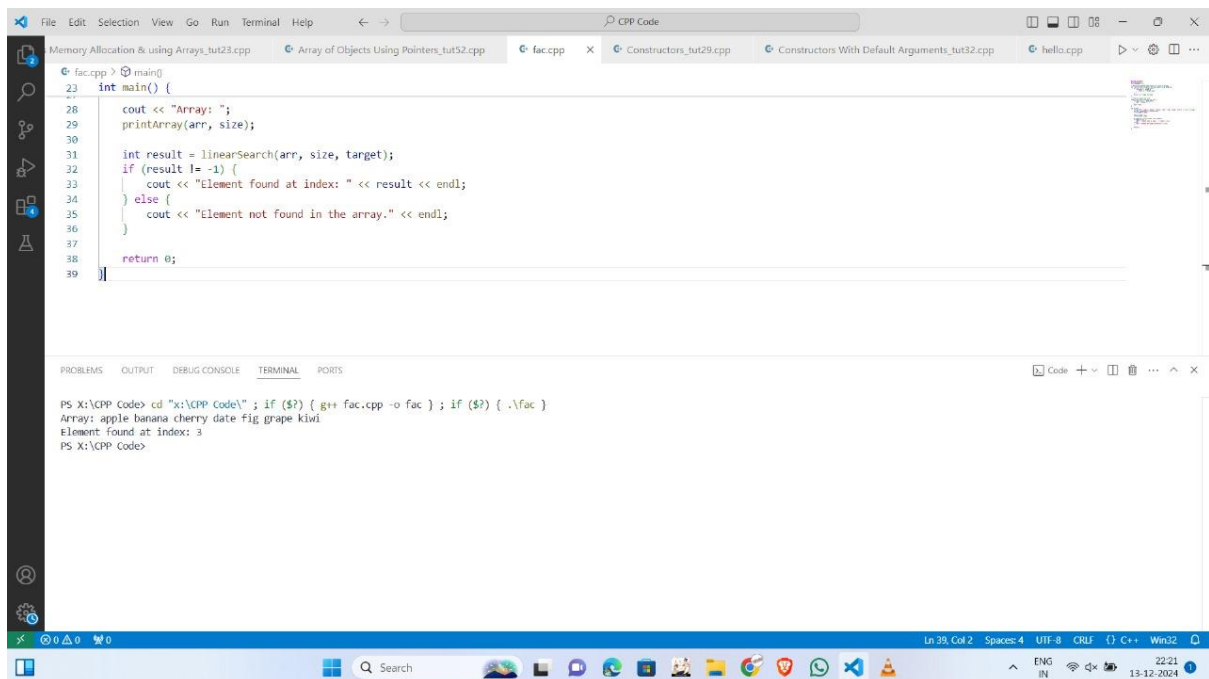OUTPUT:

Practical No.: 4

Program Description: Implementation of Linear Search on a list of strings

stored in an Array.

Solution:

```cpp
#include <iostream>

#include <string>

using namespace std;

int linearSearch(string arr[], int size, const string& target) {

    for (int I = 0; I < size; I++) {

        if (arr[I] == target) {

            return I;      }

    }

    return -1;

void printArray(string arr[], int size) {

    for (int I = 0; I < size; I++) {

        cout << arr[I] << " ";   }

    cout << endl;

}

int main() {

    string arr[] = {"apple", "banana", "cherry", "date", "fig", "grape", "kiwi"}; // Array of strings

    int size = sizeof(arr) / sizeof(arr[0]);

    string target = "date";

    cout << "Array: ";

    printArray(arr, size);

    int result = linearSearch(arr, size, target);

    if (result != -1) {

        cout << "Element found at index: " << result << endl;

    } else {

        cout << "Element not found in the array." << endl;   }

    return 0;

}
```

OUTPUT:


```cpp
int main() {

    cout << "Array: ";
    printArray(arr, size);

    int result = linearSearch(arr, size, target);
    if (result != -1) {
        cout << "Element found at index: " << result << endl;
    } else {
        cout << "Element not found in the array." << endl;
    }

    return 0;
}
```

```
PS X:\CPP Code> cd "x:\CPP Code\" ; if ($?) { g++ fac.cpp -o fac } ; if ($?) { .\fac }
Array: apple banana cherry date fig grape kiwi
Element found at index: 3
PS X:\CPP Code>
```

File Submitted by: *Amisha Goswami (0902CS231012)*