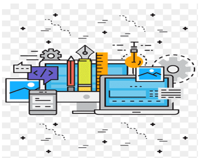


DATA STRUCTURE PRACTICAL NO : 01

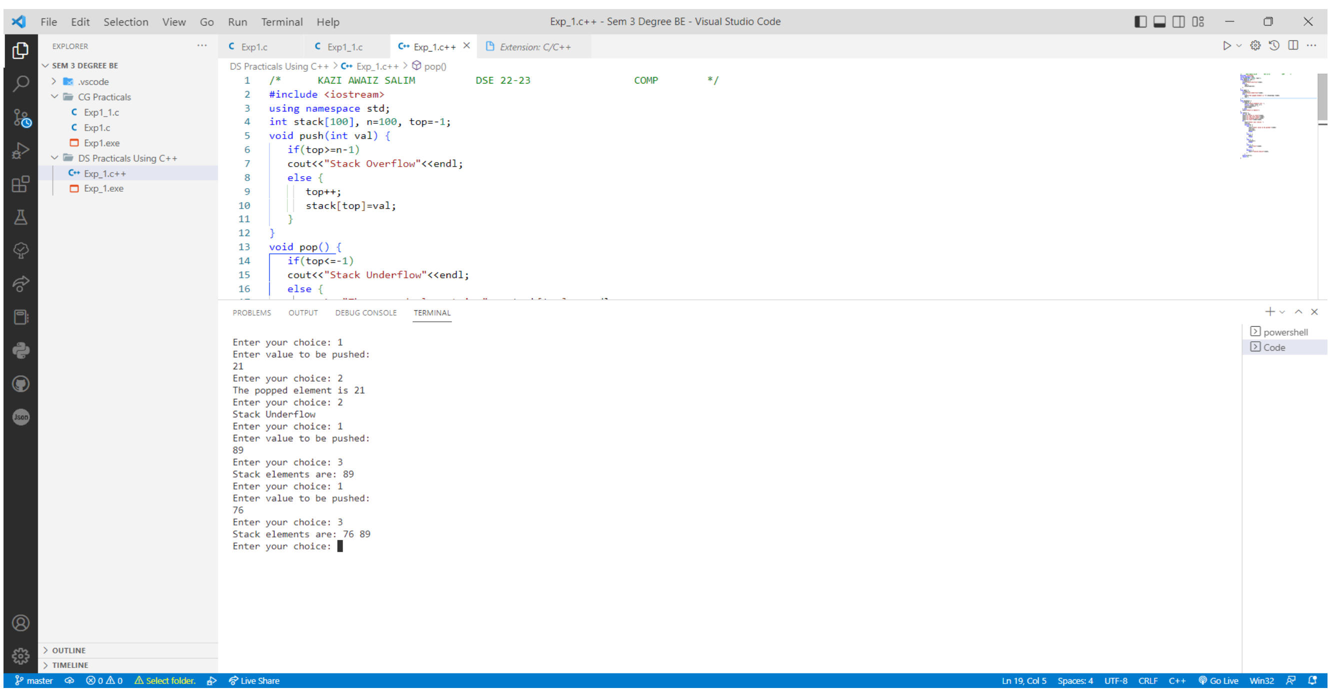
Practical No. 1: Implement Stack ADT using array.

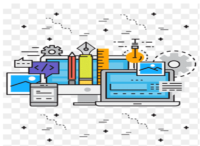
PROGRAM

```
/*      KAZI AWAIZ SALIM           DSE 22-23                  COMP        */
#include <iostream>
using namespace std;
int stack[100], n=100, top=-1;
void push(int val) {
   if(top>=n-1)
   cout<<"Stack Overflow"<<endl;
   else {
      top++;
      stack[top]=val;
   }
}
void pop() {
   if(top<=-1)
   cout<<"Stack Underflow"<<endl;
   else {
      cout<<"The popped element is "<< stack[top] <<endl;
      top--;
   }
}
void display() {
   if(top>=0) {
      cout<<"Stack elements are: ";
      for(int i=top; i>=0; i--)
      cout<<stack[i]<<" ";
      cout<<endl;
   } else
   cout<<"Stack is empty\n";
}
int main() {
   int ch, val;
   cout<<"1) Push in stack"<<endl;
   cout<<"2) Pop from stack"<<endl;
   cout<<"3) Display stack"<<endl;
   cout<<"4) Exit"<<endl<<endl;
   do {
      cout<<"Enter your choice: ";
      cin>>ch;
      switch(ch) {
         case 1: {
            cout<<"Enter value to be pushed:"<<endl;
            cin>>val;
            push(val);
            break;
         }
         case 2: {
            pop();
            break;
```

```
            }
            case 3: {
                display();
                break;
            }
            case 4: {
                cout<<"Exit"<<endl;
                break;
            }
            default: {
                cout<<"Invalid Choice"<<endl;
            }
        }
    }while(ch!=4);
    return 0;
}
```

OUTPUT OF PR 1





DATA STRUCTURE PRACTICAL NO : 02

Practical No. 2 : Convert an Infix expression to Postfix expression using stack ADT.

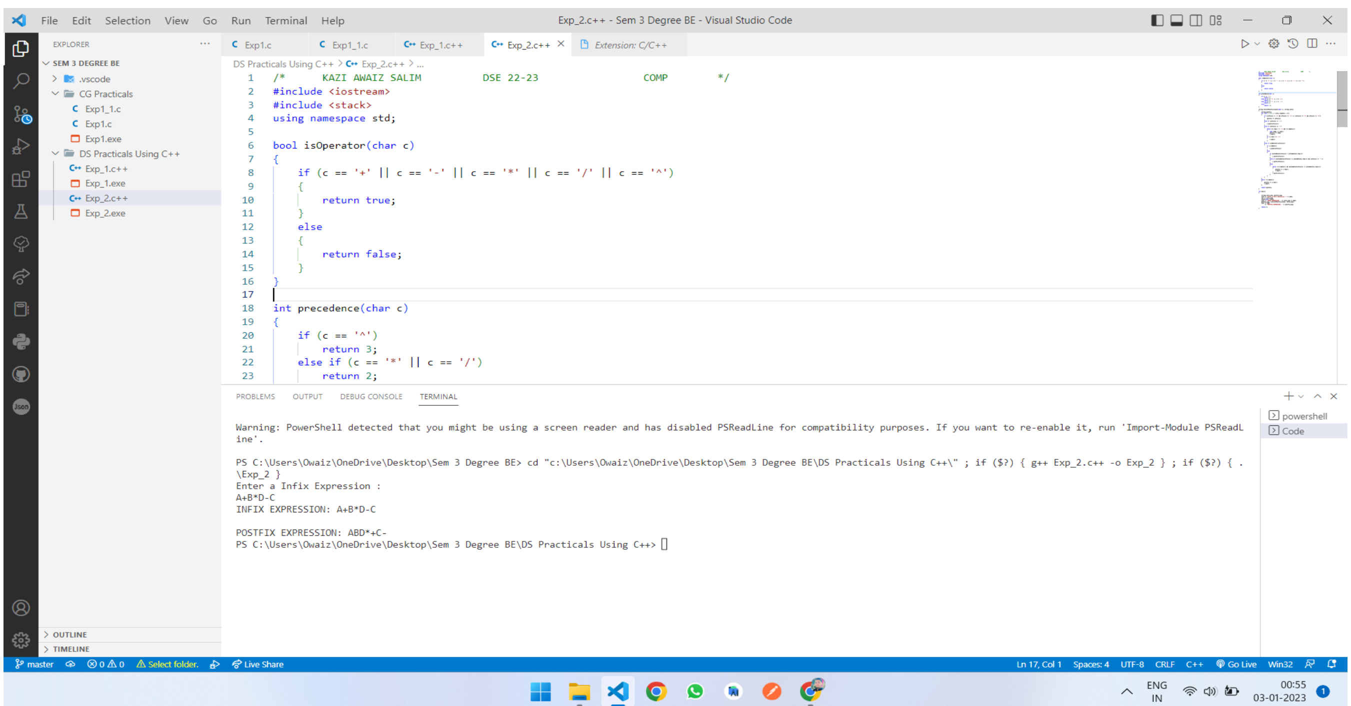
PROGRAM

```
/*      KAZI AWAIZ SALIM          DSE 22-23                 COMP        */
#include <iostream>
#include <stack>
using namespace std;

bool isOperator(char c)
{
    if (c == '+' || c == '-' || c == '*' || c == '/' || c == '^')
    {
        return true;
    }
    else
    {
        return false;
    }
}

int precedence(char c)
{
    if (c == '^')
        return 3;
    else if (c == '*' || c == '/')
        return 2;
    else if (c == '+' || c == '-')
        return 1;
    else
        return -1;
}

string InfixToPostfix(stack<char> s, string infix)
{
    string postfix;
    for (int i = 0; i < infix.length(); i++)
    {
        if ((infix[i] >= 'a' && infix[i] <= 'z') || (infix[i] >= 'A' && infix[i] <= 'Z'))
        {
            postfix += infix[i];
        }
        else if (infix[i] == '(')
        {
            s.push(infix[i]);
        }
        else if (infix[i] == ')')
```

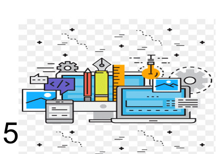
```
        {
            while ((s.top() != '(') && (!s.empty()))
            {
                char temp = s.top();
                postfix += temp;
                s.pop();
            }
            if (s.top() == '(')
            {
                s.pop();
            }
        }
        else if (isOperator(infix[i]))
        {
            if (s.empty())
            {
                s.push(infix[i]);
            }
            else
            {
                if (precedence(infix[i]) > precedence(s.top()))
                {
                    s.push(infix[i]);
                }
                else if ((precedence(infix[i]) == precedence(s.top())) && (infix[i] == '^'))
                {
                    s.push(infix[i]);
                }
                else
                {
                    while ((!s.empty()) && (precedence(infix[i]) <= precedence(s.top())))
                    {
                        postfix += s.top();
                        s.pop();
                    }
                    s.push(infix[i]);
                }
            }
        }
    }
    while (!s.empty())
    {
        postfix += s.top();
        s.pop();
    }

    return postfix;
}

int main()
{
```

```
    string infix_exp, postfix_exp;
    cout << "Enter a Infix Expression :" << endl;
    cin >> infix_exp;
    stack<char> stack;
    cout << "INFIX EXPRESSION: " << infix_exp << endl;
    postfix_exp = InfixToPostfix(stack, infix_exp);
    cout << endl
         << "POSTFIX EXPRESSION: " << postfix_exp;

    return 0;
}
```

OUTPUT OF PR 2



DATA STRUCTURE PRACTICAL NO : 03



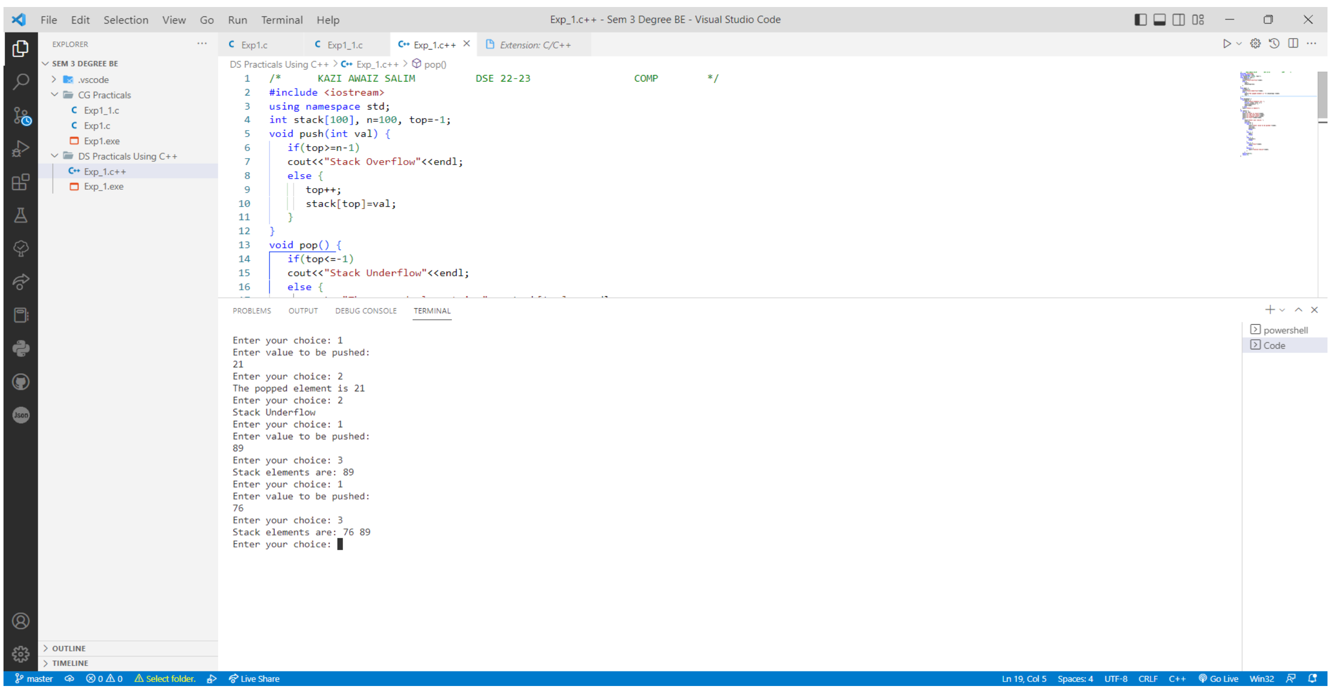
Practical No. 3 : Applications of Stack ADT.

PROGRAM

```
/*      KAZI AWAIZ SALIM          DSE 22-23                 COMP        */
#include <iostream>
using namespace std;
int stack[100], n=100, top=-1;
void push(int val) {
   if(top>=n-1)
   cout<<"Stack Overflow"<<endl;
   else {
      top++;
      stack[top]=val;
   }
}
void pop() {
   if(top<=-1)
   cout<<"Stack Underflow"<<endl;
   else {
      cout<<"The popped element is "<< stack[top] <<endl;
      top--;
   }
}
void display() {
   if(top>=0) {
      cout<<"Stack elements are: ";
      for(int i=top; i>=0; i--)
      cout<<stack[i]<<" ";
      cout<<endl;
   } else
   cout<<"Stack is empty\n";
}
int main() {
   int ch, val;
   cout<<"1) Push in stack"<<endl;
   cout<<"2) Pop from stack"<<endl;
   cout<<"3) Display stack"<<endl;
   cout<<"4) Exit"<<endl<<endl;
   do {
      cout<<"Enter your choice: ";
      cin>>ch;
      switch(ch) {
         case 1: {
            cout<<"Enter value to be pushed:"<<endl;
            cin>>val;
            push(val);
            break;
         }
         case 2: {
            pop();
```

```
            break;
        }
        case 3: {
            display();
            break;
        }
        case 4: {
            cout<<"Exit"<<endl;
            break;
        }
        default: {
            cout<<"Invalid Choice"<<endl;
        }
    }
  }while(ch!=4);
  return 0;
}
```

OUTPUT OF PR NO 3



Practical No. 4 : Implement Priority Queue ADT using array.

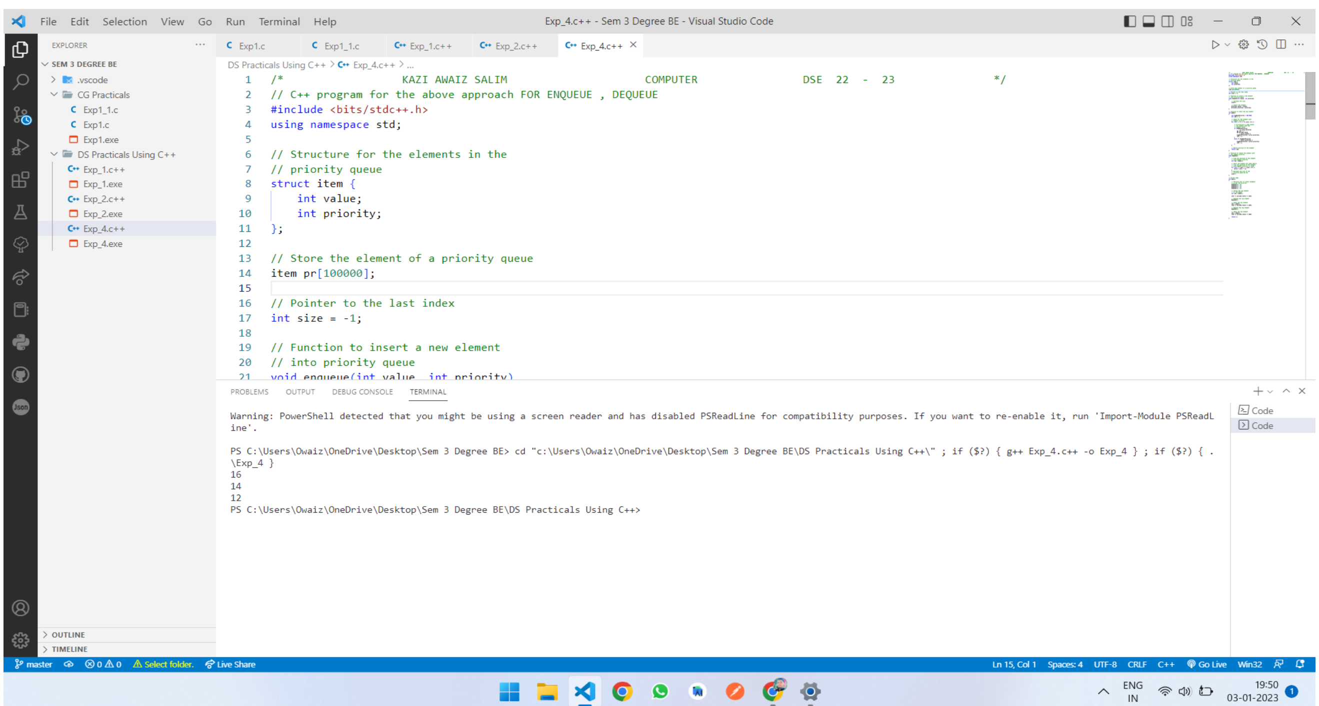
PROGRAM

```
/*    KAZI AWAIZ SALIM                     COMPUTER                 DSE  22  -  23    */
// C++ program for the above approach FOR ENQUEUE , DEQUEUE
#include <bits/stdc++.h>
using namespace std;

// Structure for the elements in the
// priority queue
struct item {
    int value;
    int priority;
};

// Store the element of a priority queue
item pr[100000];

// Pointer to the last index
int size = -1;

// Function to insert a new element
// into priority queue
void enqueue(int value, int priority)
{
    // Increase the size
    size++;

    // Insert the element
    pr[size].value = value;
    pr[size].priority = priority;
}

// Function to check the top element
int peek()
{
    int highestPriority = INT_MIN;
    int ind = -1;

    // Check for the element with
    // highest priority
    for (int i = 0; i <= size; i++) {

        // If priority is same choose
        // the element with the
        // highest value
        if (highestPriority
                == pr[i].priority
            && ind > -1
```

```
            && pr[ind].value
                < pr[i].value) {
            highestPriority = pr[i].priority;
            ind = i;
        }
        else if (highestPriority
                < pr[i].priority) {
            highestPriority = pr[i].priority;
            ind = i;
        }
    }

    // Return position of the element
    return ind;
}

// Function to remove the element with
// the highest priority
void dequeue()
{
    // Find the position of the element
    // with highest priority
    int ind = peek();

    // Shift the element one index before
    // from the position of the element
    // with highest priority is found
    for (int i = ind; i < size; i++) {
        pr[i] = pr[i + 1];
    }
    // Decrease the size of the
    // priority queue by one
    size--;
}

// Driver Code
int main()
{
    // Function Call to insert elements
    // as per the priority
    enqueue(10, 2);
    enqueue(14, 4);
    enqueue(16, 4);
    enqueue(12, 3);

    // Stores the top element
    // at the moment
    int ind = peek();

    cout << pr[ind].value << endl;
```

```
    // Dequeue the top element
    dequeue();

    // Check the top element
    ind = peek();
    cout << pr[ind].value << endl;

    // Dequeue the top element
    dequeue();

    // Check the top element
    ind = peek();
    cout << pr[ind].value << endl;

    return 0;
}
```

OUTPUT OF PR NO 4



DATA STRUCTURE PRACTICAL NO : 05

Practical No. 5 : Implement Singly Linked List ADT..

PROGRAM

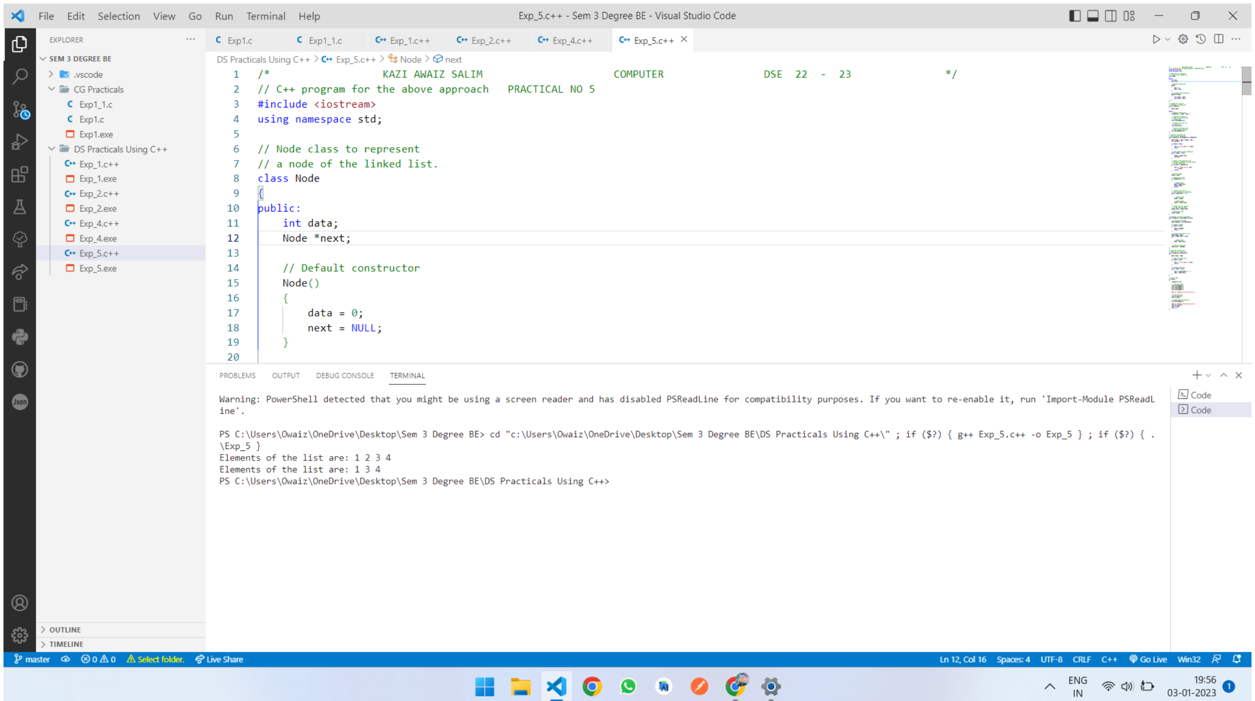
```
/*                  KAZI AWAIZ SALIM                     COMPUTER                 DSE  22  -
23                 */
// C++ program for the above approach   PRACTICAL NO 5
#include <iostream>
using namespace std;

// Node class to represent
// a node of the linked list.
class Node
{
public:
    int data;
    Node *next;

    // Default constructor
    Node()
    {
        data = 0;
        next = NULL;
    }

    // Parameterised Constructor
    Node(int data)
    {
        this->data = data;
        this->next = NULL;
    }
};

// Linked list class to
// implement a linked list.
class Linkedlist
{
    Node *head;

public:
    // Default constructor
    Linkedlist() { head = NULL; }

    // Function to insert a
    // node at the end of the
    // linked list.
    void insertNode(int);

    // Function to print the
    // linked list.
    void printList();
```

```
    // Function to delete the
    // node at given position
    void deleteNode(int);
};

// Function to delete the
// node at given position
void Linkedlist::deleteNode(int nodeOffset)
{
    Node *temp1 = head, *temp2 = NULL;
    int ListLen = 0;

    if (head == NULL)
    {
        cout << "List empty." << endl;
        return;
    }

    // Find length of the linked-list.
    while (temp1 != NULL)
    {
        temp1 = temp1->next;
        ListLen++;
    }

    // Check if the position to be
    // deleted is greater than the length
    // of the linked list.
    if (ListLen < nodeOffset)
    {
        cout << "Index out of range"
             << endl;
        return;
    }

    // Declare temp1
    temp1 = head;

    // Deleting the head.
    if (nodeOffset == 1)
    {

        // Update head
        head = head->next;
        delete temp1;
        return;
    }

    // Traverse the list to
    // find the node to be deleted.
    while (nodeOffset-- > 1)
```

```
    {

        // Update temp2
        temp2 = temp1;

        // Update temp1
        temp1 = temp1->next;
    }

    // Change the next pointer
    // of the previous node.
    temp2->next = temp1->next;

    // Delete the node
    delete temp1;
}

// Function to insert a new node.
void Linkedlist::insertNode(int data)
{
    // Create the new Node.
    Node *newNode = new Node(data);

    // Assign to head
    if (head == NULL)
    {
        head = newNode;
        return;
    }

    // Traverse till end of list
    Node *temp = head;
    while (temp->next != NULL)
    {

        // Update temp
        temp = temp->next;
    }

    // Insert at the last.
    temp->next = newNode;
}

// Function to print the
// nodes of the linked list.
void Linkedlist::printList()
{
    Node *temp = head;

    // Check for empty list.
    if (head == NULL)
```

```
    {
        cout << "List empty" << endl;
        return;
    }

    // Traverse the list.
    while (temp != NULL)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
}

// Driver Code
int main()
{
    Linkedlist list;

    // Inserting nodes
    list.insertNode(1);
    list.insertNode(2);
    list.insertNode(3);
    list.insertNode(4);

    cout << "Elements of the list are: ";

    // Print the list
    list.printList();
    cout << endl;

    // Delete node at position 2.
    list.deleteNode(2);

    cout << "Elements of the list are: ";
    list.printList();
    cout << endl;
    return 0;
}
```

OUTPUT of PR No : 5


Exp_5.c++ - Sem 3 Degree BE - Visual Studio Code
DS Practicals Using C++ > Exp_5.c++ > Node > next
/* KAZI AWAIZ SALIM COMPUTER DSE 22 - 23 */
// C++ program for the above approach PRACTICAL NO 5
#include <iostream>
using namespace std;
// Node class to represent
// a node of the linked list.
class Node
{
public:
int data;
Node *next;
// Default constructor
Node()
{
data = 0;
next = NULL;
}
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL
Warning: PowerShell detected that you might be using a screen reader and has disabled PSReadLine for compatibility purposes. If you want to re-enable it, run 'Import-Module PSReadLine'.
PS C:\Users\Owaiz\OneDrive\Desktop\Sem 3 Degree BE> cd "c:\Users\Owaiz\OneDrive\Desktop\Sem 3 Degree BE\DS Practicals Using C++\" ; if ($?) { g++ Exp_5.c++ -o Exp_5 } ; if ($?) { .\Exp_5 }
Elements of the list are: 1 2 3 4
Elements of the list are: 1 3 4
PS C:\Users\Owaiz\OneDrive\Desktop\Sem 3 Degree BE\DS Practicals Using C++>

DATA STRUCTURE PRACTICAL NO : 06

Practical No. 6 : Implement Binary Search Tree ADT using Linked List.

PROGRAM

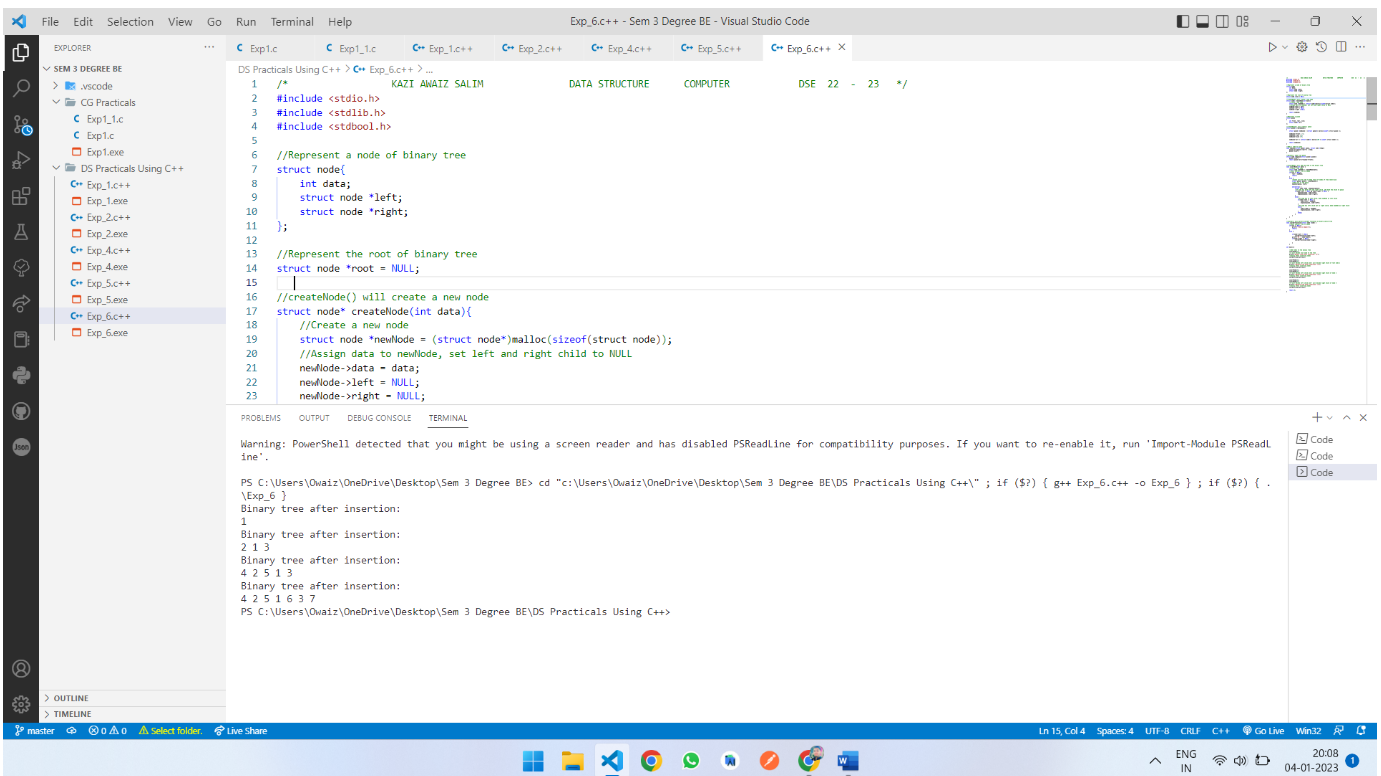
```
/*  KAZI AWAIZ SALIM               DATA STRUCTURE      COMPUTER        DSE  22  -  23   */
#include <stdio.h>
#include <stdlib.h>
#include <stdbool.h>

//Represent a node of binary tree
struct node{
    int data;
    struct node *left;
    struct node *right;
};

//Represent the root of binary tree
struct node *root = NULL;

//createNode() will create a new node
struct node* createNode(int data){
    //Create a new node
    struct node *newNode = (struct node*)malloc(sizeof(struct node));
    //Assign data to newNode, set left and right child to NULL
    newNode->data = data;
    newNode->left = NULL;
    newNode->right = NULL;

    return newNode;
}

//Represent a queue
struct queue
{
    int front, rear, size;
    struct node* *arr;
};

//createQueue() will create a queue
struct queue* createQueue()
{
    struct queue* newQueue = (struct queue*) malloc(sizeof( struct queue ));

    newQueue->front = -1;
    newQueue->rear = 0;
    newQueue->size = 0;
```

```
    newQueue->arr = (struct node**) malloc(100 * sizeof( struct node* ));

    return newQueue;
}

//Adds a node to queue
void enqueue(struct queue* queue, struct node *temp){
    queue->arr[queue->rear++] = temp;
    queue->size++;
}

//Deletes a node from queue
struct node *dequeue(struct queue* queue){
    queue->size--;
    return queue->arr[++queue->front];
}


//insertNode() will add new node to the binary tree
void insertNode(int data) {
    //Create a new node
    struct node *newNode = createNode(data);
    //Check whether tree is empty
    if(root == NULL){
        root = newNode;
        return;
    }
    else {
        //Queue will be used to keep track of nodes of tree level-wise
        struct queue* queue = createQueue();
        //Add root to the queue
        enqueue(queue, root);

        while(true) {
            struct node *node = dequeue(queue);
            //If node has both left and right child, add both the child to queue
            if(node->left != NULL && node->right != NULL) {
                enqueue(queue, node->left);
                enqueue(queue, node->right);
            }
            else {
                //If node has no left child, make newNode as left child
                if(node->left == NULL) {
                    node->left = newNode;
                    enqueue(queue, node->left);
                }
                //If node has left child but no right child, make newNode as right child
                else {
```

```
                    node->right = newNode;
                    enqueue(queue, node->right);
                }
                break;
            }
        }
    }
}

//inorder() will perform inorder traversal on binary search tree
void inorderTraversal(struct node *node) {
    //Check whether tree is empty
    if(root == NULL){
        printf("Tree is empty\n");
        return;
    }
    else {

        if(node->left != NULL)
            inorderTraversal(node->left);
        printf("%d ", node->data);
        if(node->right != NULL)
            inorderTraversal(node->right);

        }
    }

int main(){

    //Add nodes to the binary tree
    insertNode(1);
    //1 will become root node of the tree
    printf("Binary tree after insertion: \n");
    //Binary after inserting nodes
    inorderTraversal(root);

    insertNode(2);
    insertNode(3);
    //2 will become left child and 3 will become right child of root node 1
    printf("\nBinary tree after insertion: \n");
    //Binary after inserting nodes
    inorderTraversal(root);

    insertNode(4);
    insertNode(5);
    //4 will become left child and 5 will become right child of node 2
    printf("\nBinary tree after insertion: \n");
    //Binary after inserting nodes
    inorderTraversal(root);
```

```
    insertNode(6);
    insertNode(7);
    //6 will become left child and 7 will become right child of node 3
    printf("\nBinary tree after insertion: \n");
    //Binary after inserting nodes
    inorderTraversal(root);

    return 0;
}
```

OUTPUT OF PR 6



DATA STRUCTURE PRACTICAL NO : 07

Practical No. 7 : Implement Graph Traversal techniques: a) Breadth First Search b) Depth First Search

PROGRAM

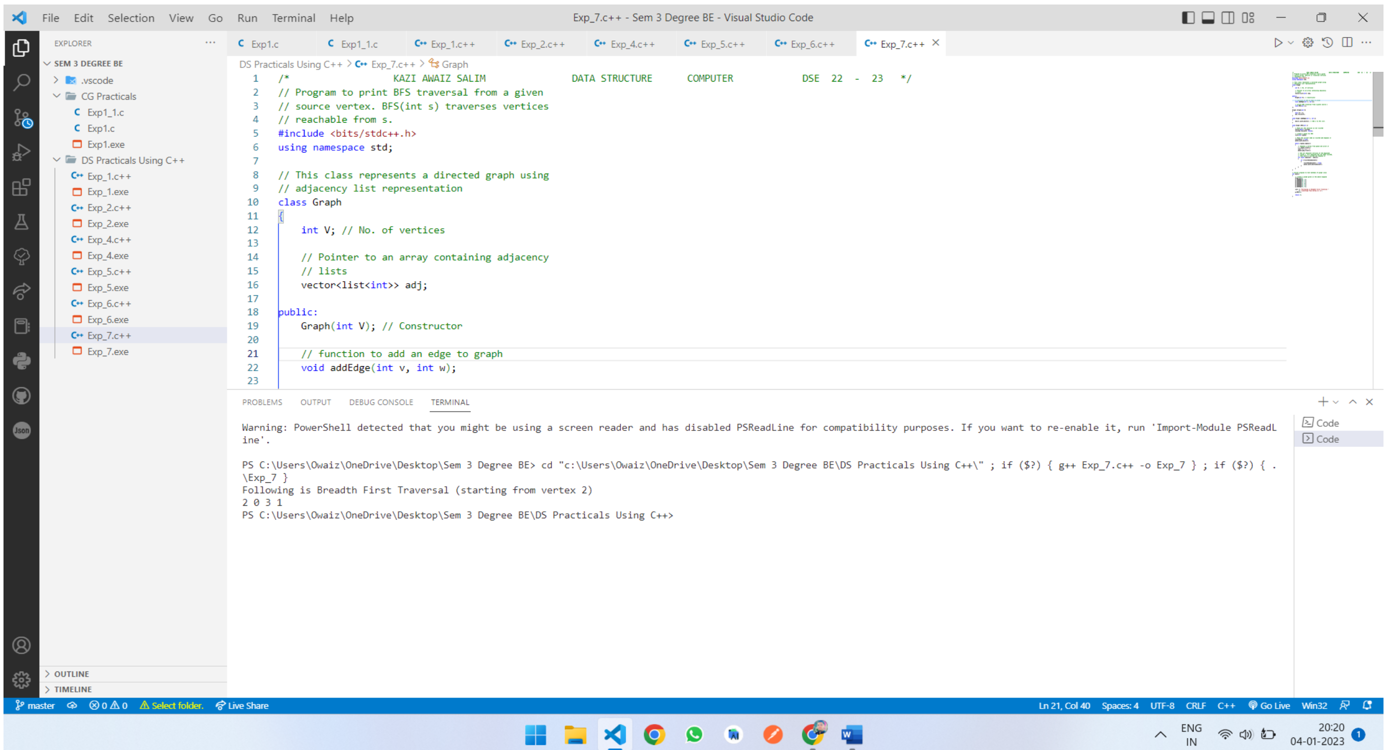
a) Breadth First Search

```
/*  KAZI AWAIZ SALIM      DATA STRUCTURE      COMPUTER      DSE  22  -  23   */
// Program to print BFS traversal from a given
// source vertex. BFS(int s) traverses vertices
// reachable from s.
#include <bits/stdc++.h>
using namespace std;

// This class represents a directed graph using
// adjacency list representation
class Graph
{
    int V; // No. of vertices

    // Pointer to an array containing adjacency
    // lists
    vector<list<int>> adj;

public:
    Graph(int V); // Constructor

    // function to add an edge to graph
    void addEdge(int v, int w);

    // prints BFS traversal from a given source s
    void BFS(int s);
};

Graph::Graph(int V)
{
    this->V = V;
    adj.resize(V);
}

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v’s list.
}

void Graph::BFS(int s)
{
    // Mark all the vertices as not visited
```

```
    vector<bool> visited;
    visited.resize(V, false);

    // Create a queue for BFS
    list<int> queue;

    // Mark the current node as visited and enqueue it
    visited[s] = true;
    queue.push_back(s);

    while (!queue.empty())
    {
        // Dequeue a vertex from queue and print it
        s = queue.front();
        cout << s << " ";
        queue.pop_front();

        // Get all adjacent vertices of the dequeued
        // vertex s. If a adjacent has not been visited,
        // then mark it visited and enqueue it
        for (auto adjecent : adj[s])
        {
            if (!visited[adjecent])
            {
                visited[adjecent] = true;
                queue.push_back(adjecent);
            }
        }
    }
}

// Driver program to test methods of graph class
int main()
{
    // Create a graph given in the above diagram
    Graph g(4);
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Breadth First Traversal "
         << "(starting from vertex 2) \n";
    g.BFS(2);

    return 0;
}
```

OUTPUT OF PR 7

A)



B) Depth First Search

```
/*   KAZI AWAIZ SALIM     DATA STRUCTURE     COMPUTER        DSE  22  -  23   */
// C++ program to print DFS traversal from
// a given vertex in a given graph
#include <bits/stdc++.h>
using namespace std;

// Graph class represents a directed graph
// using adjacency list representation
class Graph
{
public:
    map<int, bool> visited;
    map<int, list<int>> adj;

    // function to add an edge to graph
    void addEdge(int v, int w);

    // DFS traversal of the vertices
```

```
    // reachable from v
    void DFS(int v);
};

void Graph::addEdge(int v, int w)
{
    adj[v].push_back(w); // Add w to v’s list.
}

void Graph::DFS(int v)
{
    // Mark the current node as visited and
    // print it
    visited[v] = true;
    cout << v << " ";

    // Recur for all the vertices adjacent
    // to this vertex
    list<int>::iterator i;
    for (i = adj[v].begin(); i != adj[v].end(); ++i)
        if (!visited[*i])
            DFS(*i);
}

// Driver's code
int main()
{
    // Create a graph given in the above diagram
    Graph g;
    g.addEdge(0, 1);
    g.addEdge(0, 2);
    g.addEdge(1, 2);
    g.addEdge(2, 0);
    g.addEdge(2, 3);
    g.addEdge(3, 3);

    cout << "Following is Depth First Traversal"
            " (starting from vertex 2) \n";

    // Function call
    g.DFS(2);

    return 0;
}
```

OUTPUT OF PR 7 B)