1. Write a program to create a singly linked list of n nodes and display it in reverse order.

Code:

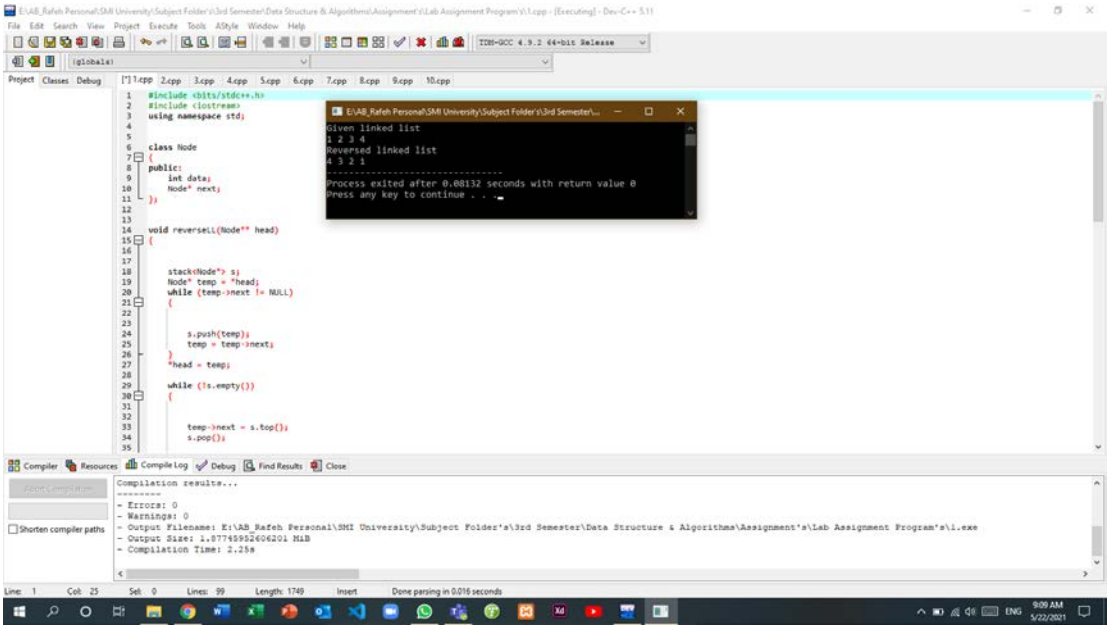
```
#include <bits/stdc++.h>
#include <iostream>
using namespace std;
class Node
{
public:
    int data;
    Node* next;
};
// reverse the linked list
void reverseLL(Node** head)
{
    stack<Node*> s;
    Node* temp = *head;
    while (temp->next != NULL)
    {
       s.push(temp);
        temp = temp->next;
    }
    *head = temp;
```

```
    while (!s.empty())
    {
        temp->next = s.top();
        s.pop();
        temp = temp->next;
    }
    temp->next = NULL;
}
void printlist(Node* temp)
{
    while (temp != NULL)
    {
        cout << temp->data << " ";
        temp = temp->next;
    }
}
void insert_back(Node** head, int value)
{
Node* temp = new Node();
    temp->data = value;
    temp->next = NULL;
    if (*head == NULL)
    {
      *head = temp;
      return;
```

```
    }
    else
    {
      Node* last_node = *head;
      while (last_node->next != NULL)
      {
        last_node = last_node->next;
      }
      last_node->next = temp;
      return;
    }
}
 int main()
{
    Node* head = NULL;
    insert_back(&head, 1);
    insert_back(&head, 2);
    insert_back(&head, 3);
    insert_back(&head, 4);
    cout << "Given linked list\n";
    printlist(head);
    reverseLL(&head);
    cout << "\nReversed linked list\n";
    printlist(head);
    return 0;
```

}

Output:



2. Write a program to create a singly linked list of n nodes and count the number of nodes.

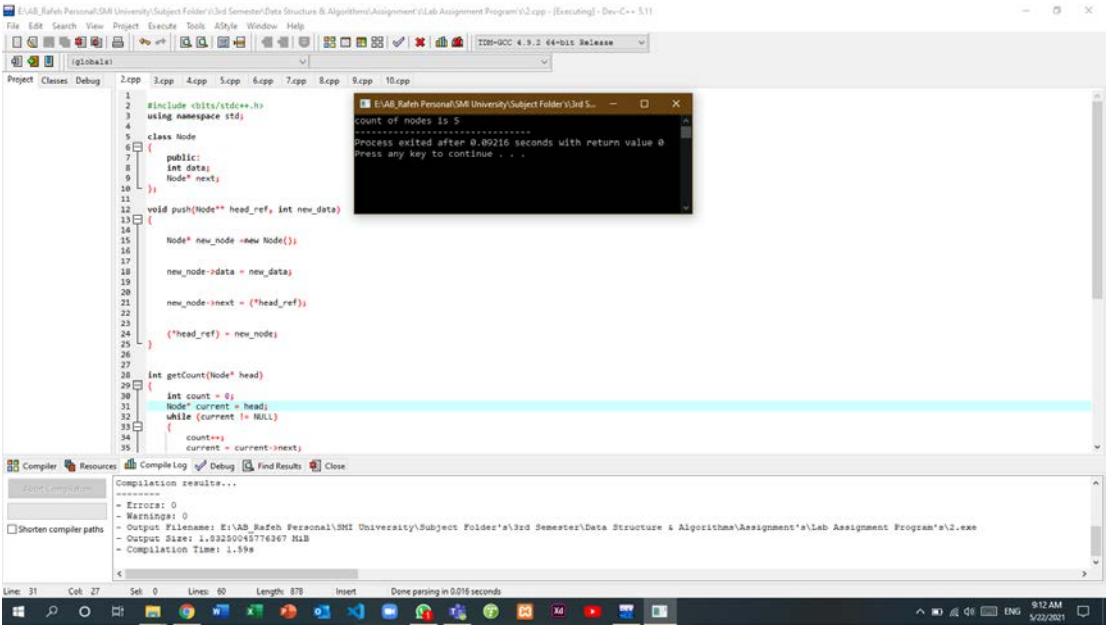
Code:

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

using namespace std;


class Node

{

    public:

    int data;

    Node* next;

};
```

```
    void push(Node** head_ref, int new_data)
    {
        Node* new_node =new Node();
        new_node->data = new_data;
        new_node->next = (*head_ref);
        (*head_ref) = new_node;
    }

    /* Counts no. of nodes in linked list */
    int getCount(Node* head)
    {
        int count = 0; // Initialize count
        Node* current = head; // Initialize current
        while (current != NULL)
        {
                count++;
                current = current->next;
        }
        return count;
    }
int main()
    {
        Node* head = NULL;
        push(&head, 1);
        push(&head, 3);
```

```
    push(&head, 1);
    push(&head, 2);
    push(&head, 1);

    cout<<"count of nodes is "<< getCount(head);
    return 0;
}
```

Output:



3. Write a program to find the maximum value from a doubly linked list.

Code:

```
#include <iostream>

using namespace std;

struct Node

{
```

```
    int data;
    struct Node* next;
    struct Node* prev;
};
 void push(struct Node** head_ref, int new_data)
{
    struct Node* new_node =
    (struct Node*)malloc(sizeof(struct Node));
    new_node->data = new_data;
    beginning, prev is always NULL */
    new_node->prev = NULL;
    new_node->next = (*head_ref);
    if ((*head_ref) != NULL)
        (*head_ref)->prev = new_node;
    (*head_ref) = new_node;
}
int LargestInDLL(struct Node** head_ref)
{
    struct Node *max, *temp;
    temp = max = *head_ref;

    // traverse the whole doubly linked list
    while (temp != NULL)
    {
        if (temp->data > max->data)
```

```
            max = temp;
        temp = temp->next;
    }
    return max->data;
}
int main()
{
    struct Node* head = NULL;
    push(&head, 20);
    push(&head, 14);
    push(&head, 181);
    push(&head, 100);

    cout << LargestInDLL(&head);
    return 0;
}
```

Output: code run from online free codesite

Output:

```
181
```

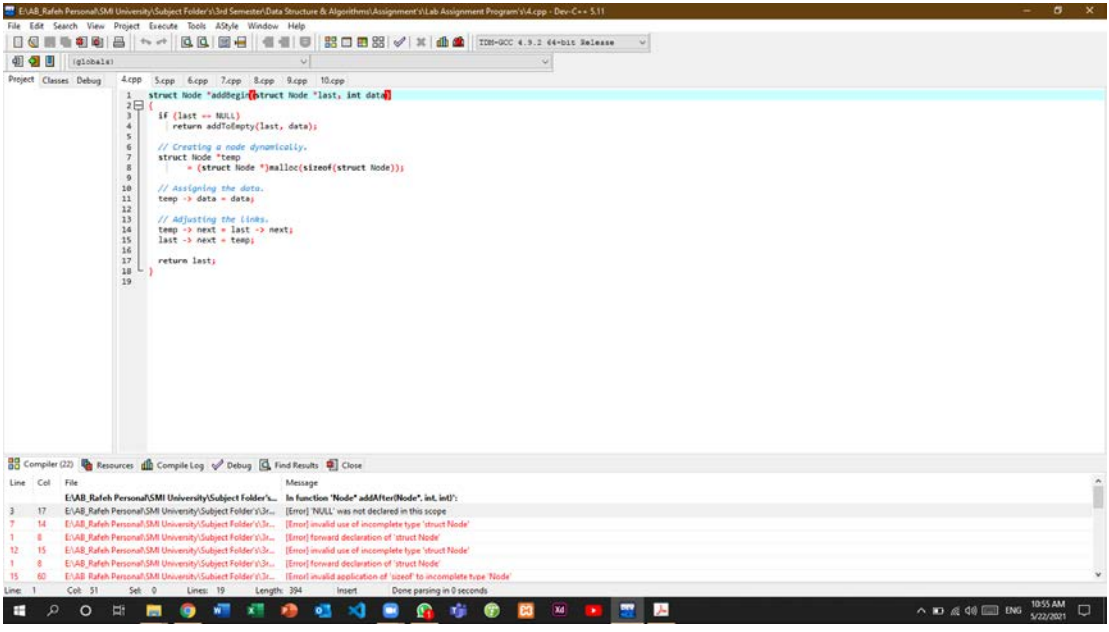
4. Write a program to insert a node at the beginning of a circular linked list.

Code:

```
struct Node *addBegin(struct Node *last, int data)
{
  if (last == NULL)
```

```
    return addToEmpty(last, data);


  struct Node *temp
        = (struct Node *)malloc(sizeof(struct Node));


  // Assigning the data.
  temp -> data = data;
  temp -> next = last -> next;
  last -> next = temp;
  return last;
}
```

Output: Found error unable to resolve



5. Write a program to insert a node at the middle of a circular linked list.

Code:

```
struct Node *addAfter(struct Node *last, int data, int item)
{
    if (last == NULL)
        return NULL;

    struct Node *temp, *p;
    p = last -> next;

    do
    {
        if (p ->data == item)
        {
            temp = (struct Node *)malloc(sizeof(struct Node));

            // Assigning the data.
            temp -> data = data;
            temp -> next = p -> next;
            p -> next = temp
            if (p == last)
                last = temp;
            return last;
        }
```

```
        p = p -> next;
    } while (p != last -> next);

    cout << item << " not present in the list." << endl;
    return last;
}
```

Output: Code run from online codingsite

Output:

```
Linked list before insertion: 1 2 4 5
Linked list after insertion: 1 2 3 4 5
```

6. Write a program to remove duplicate nodes in an unsorted linked list.

Code:

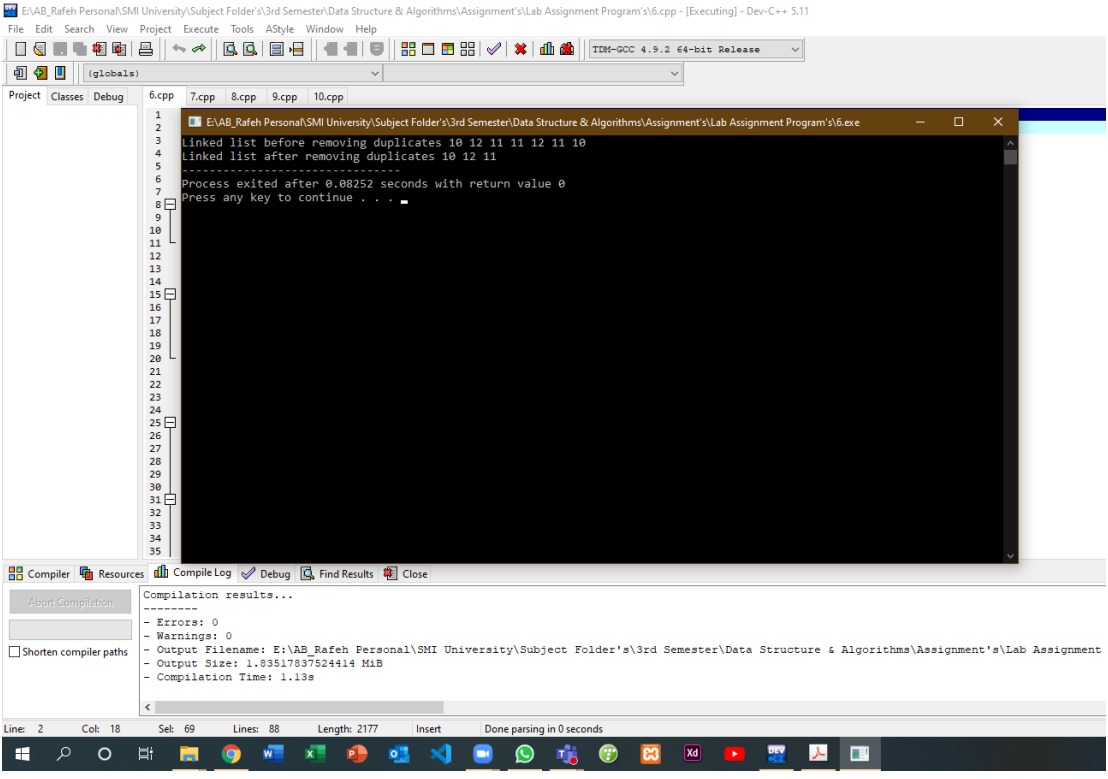
```
#include<bits/stdc++.h>
using namespace std;
struct Node
{
    int data;
    struct Node *next;
};
struct Node *newNode(int data)
{
   Node *temp = new Node;
   temp->data = data;
   temp->next = NULL;
```

```
   return temp;
}
void removeDuplicates(struct Node *start)
{
    struct Node *ptr1, *ptr2, *dup;
    ptr1 = start;
    while (ptr1 != NULL && ptr1->next != NULL)
    {
        ptr2 = ptr1;

        while (ptr2->next != NULL)
        {
            if (ptr1->data == ptr2->next->data)
            {
                dup = ptr2->next;
                ptr2->next = ptr2->next->next;
                delete(dup);
            }
            else
                ptr2 = ptr2->next;
        }
        ptr1 = ptr1->next;
    }
}
void printList(struct Node *node)
```

```
{
    while (node != NULL)
    {
        printf("%d ", node->data);
        node = node->next;
    }
}
int main()
{
    struct Node *start = newNode(10);
    start->next = newNode(12);
    start->next->next = newNode(11);
    start->next->next->next = newNode(11);
    start->next->next->next->next = newNode(12);
    start->next->next->next->next->next =
                                newNode(11);
    start->next->next->next->next->next->next =
                                newNode(10);

    printf("Linked list before removing duplicates ");
    printList(start);
    removeDuplicates(start);
    printf("\nLinked list after removing duplicates ");
    printList(start);
    return 0;
```

}

Output:



7. Merge two sorted linked lists and return it as a new sorted list. The new list should be made by splicing together the nodes of the first two lists.

Code:

```
#include <iostream>

using namespace std;

struct Node {

    int key;

    struct Node* next;

};
```

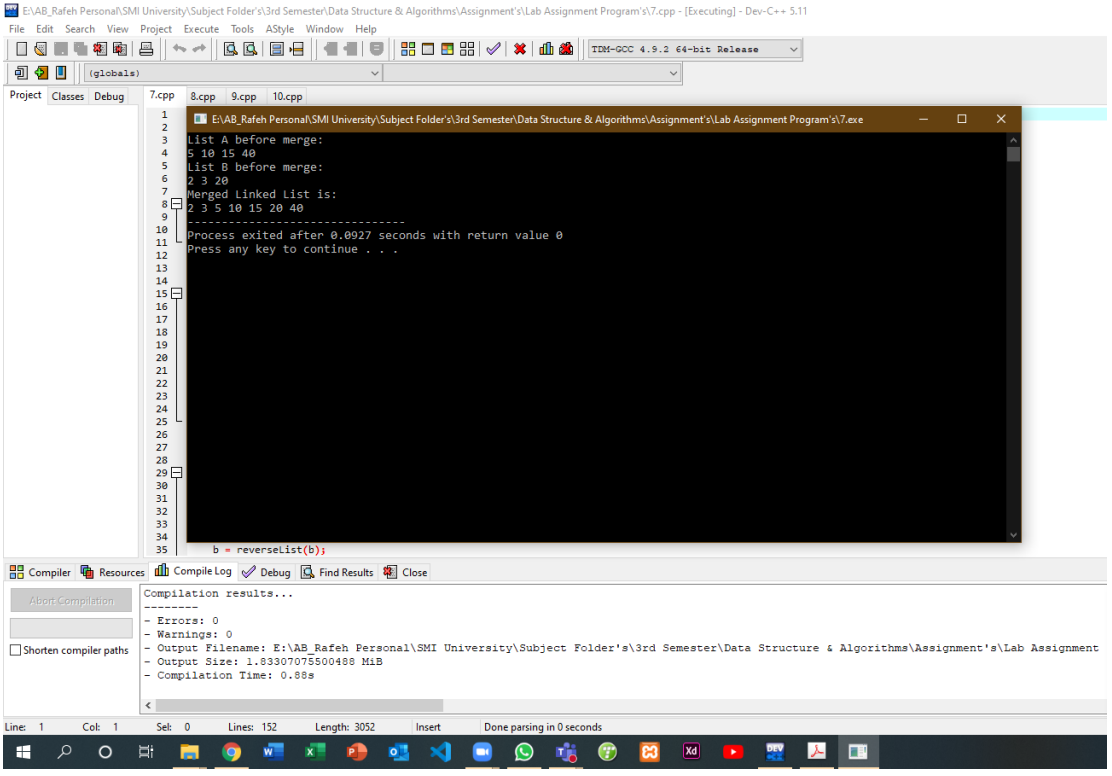
```
Node* reverseList(Node* head)
{
    if (head->next == NULL)
            return head;
    Node* rest = reverseList(head->next);
    head->next->next = head;
    head->next = NULL;
    return rest;
}
Node* sortedMerge(Node* a, Node* b)
{
    a = reverseList(a);
    b = reverseList(b);
    Node* head = NULL;

    Node* temp;
    while (a != NULL && b != NULL) {
            if (a->key >= b->key) {
                    temp = a->next;
                    a->next = head;
                    head = a;
                    a = temp;
            }

            // If b's value is greater. Below steps are similar
```

```
            // to above (Only 'a' is replaced with 'b')
            else {
                temp = b->next;
                b->next = head;
                head = b;
                b = temp;
            }
    }
    while (a != NULL) {
            temp = a->next;
            a->next = head;
            head = a;
            a = temp;
    }
    while (b != NULL) {
            temp = b->next;
            b->next = head;
            head = b;
            b = temp;
    }

    // Return the head of the result list
    return head;
}
void printList(struct Node* Node)
```

```
{
    while (Node != NULL) {
            cout << Node->key << " ";
            Node = Node->next;
    }
}
Node* newNode(int key)
{
    Node* temp = new Node;
    temp->key = key;
    temp->next = NULL;
    return temp;
}
int main()
{
    struct Node* res = NULL;
    Node* a = newNode(5);
    a->next = newNode(10);
    a->next->next = newNode(15);
    a->next->next->next = newNode(40);

    Node* b = newNode(2);
    b->next = newNode(3);
    b->next->next = newNode(20);
    cout << "List A before merge: \n";
```

```
    printList(a);
    cout << "\nList B before merge: \n";
    printList(b);
    res = sortedMerge(a, b);
    cout << "\nMerged Linked List is: \n";
    printList(res);
    return 0;
}
```

Output:



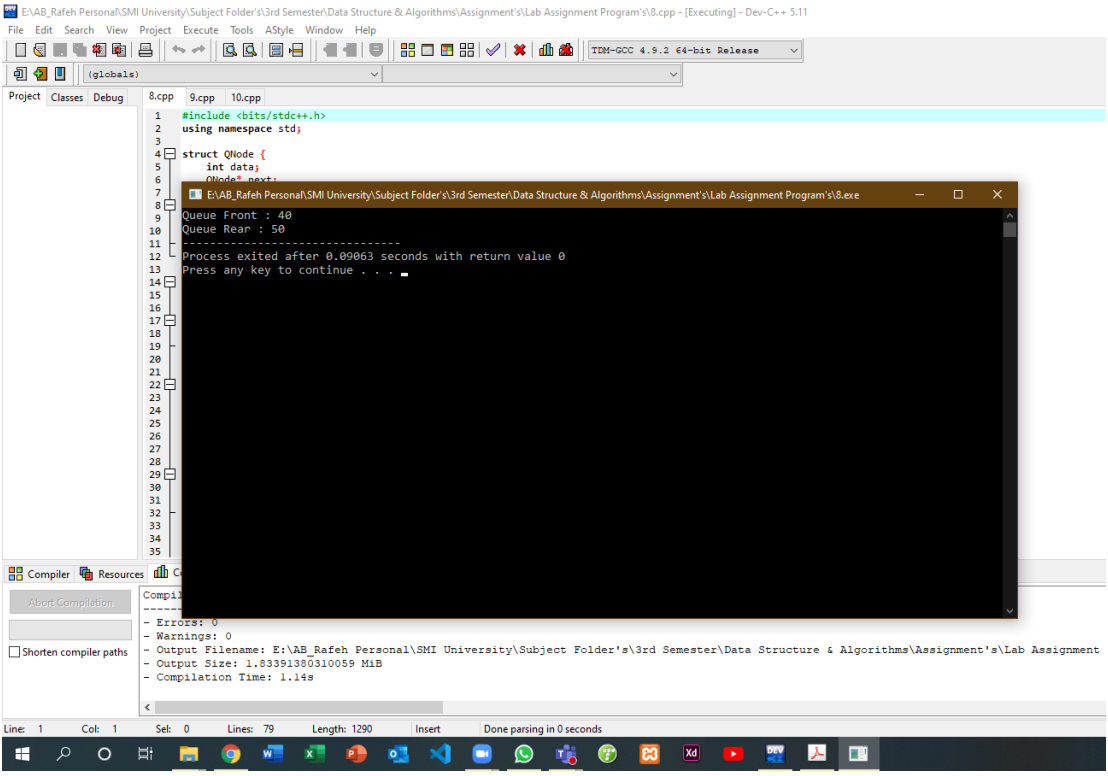
8. Write a program to Implement Queue Data Structure using Linked List.

Code:

```
#include <bits/stdc++.h>
using namespace std;

struct QNode {
    int data;
    QNode* next;
    QNode(int d)
    {
        data = d;
        next = NULL;
    }
};

struct Queue {
    QNode *front, *rear;
    Queue()
    {
        front = rear = NULL;
    }

    void enQueue(int x)
    {

        // Create a new LL node
        QNode* temp = new QNode(x);


        if (rear == NULL) {
            front = rear = temp;
            return;
        }

        rear->next = temp;
        rear = temp;
    }

    void deQueue()
    {
```

```
        if (front == NULL)

                return;
        QNode* temp = front;
        front = front->next;

        if (front == NULL)
                rear = NULL;

        delete (temp);
    }
};

int main()
{

    Queue q;
    q.enQueue(10);
    q.enQueue(20);

    q.deQueue();
    q.deQueue();

    q.enQueue(30);
    q.enQueue(40);
    q.enQueue(50);

    q.deQueue();

    cout << "Queue Front : " << (q.front)->data << endl;

    cout << "Queue Rear : " << (q.rear)->data;

}
```

Output:



9. Write a program to Implement a Stack using Linked List.

Code:

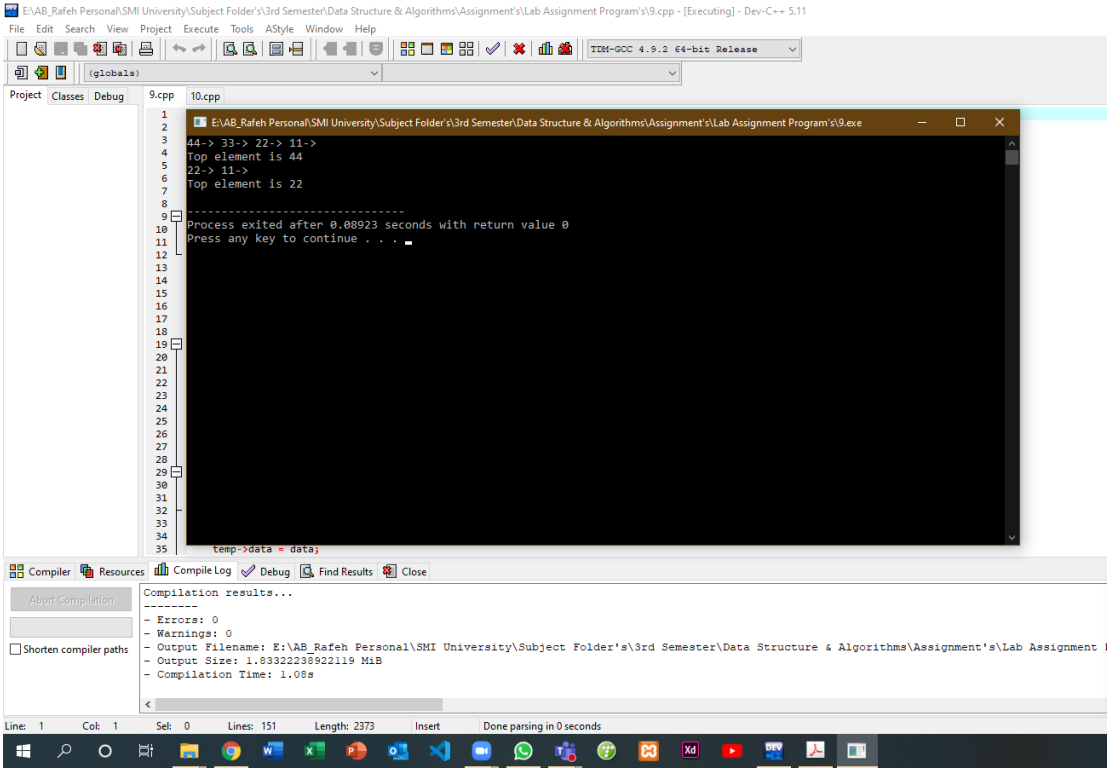
```
#include <bits/stdc++.h>
using namespace std;

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

struct Node* top;

void push(int data)
{
```

```
    struct Node* temp;
    temp = new Node();


    if (!temp)
    {
            cout << "\nHeap Overflow";
            exit(1);
    }


    temp->data = data;


    temp->link = top;


    top = temp;
}
  int isEmpty()
{
    return top == NULL;
}

int peek()
{

    if (!isEmpty())
            return top->data;
    else
            exit(1);
}

void pop()
{
    struct Node* temp;

    if (top == NULL)
    {
            cout << "\nStack Underflow" << endl;
            exit(1);
```

```
    }
    else
    {

            temp = top;
            top = top->link;

            temp->link = NULL;

            free(temp);
    }
}

void display()
{
    struct Node* temp;

    if (top == NULL)
    {
            cout << "\nStack Underflow";
            exit(1);
    }
    else
    {
            temp = top;
            while (temp != NULL)
            {
                    cout << temp->data << "-> ";
                    temp = temp->link;
            }
    }
}
int main()
{
    push(11);
    push(22);
    push(33);
    push(44);

    display();

    cout << "\nTop element is "
```

```
            << peek() << endl;
    pop();
    pop();

    display();

    cout << "\nTop element is "
            << peek() << endl;

    return 0;
}
```

Output:



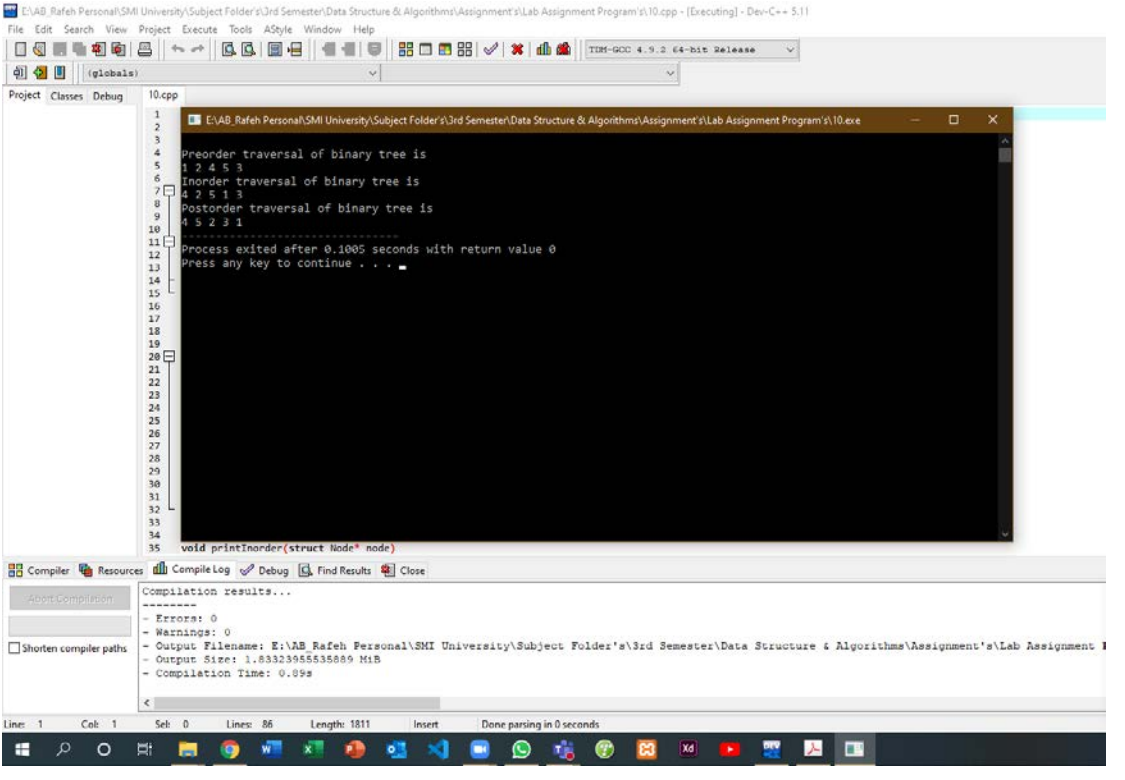
10. Write a program to implement Binary Tree Traversals - Preorder, Inorder, Postorder.

Code:

```
#include <iostream>
using namespace std;

struct Node {
    int data;
    struct Node *left, *right;
    Node(int data)
    {
            this->data = data;
            left = right = NULL;
    }
};

void printPostorder(struct Node* node)
{
    if (node == NULL)
            return;

    printPostorder(node->left);

    printPostorder(node->right);

    cout << node->data << " ";
}

void printInorder(struct Node* node)
{
    if (node == NULL)
            return;

    printInorder(node->left);

    cout << node->data << " ";

    printInorder(node->right);
}
```

```
void printPreorder(struct Node* node)
{
    if (node == NULL)
            return;

    cout << node->data << " ";

    printPreorder(node->left);

    printPreorder(node->right);
}

int main()

{
    struct Node* root = new Node(1);
    root->left = new Node(2);
    root->right = new Node(3);
    root->left->left = new Node(4);
    root->left->right = new Node(5);

    cout << "\nPreorder traversal of binary tree is \n";

    printPreorder(root);

    cout << "\nInorder traversal of binary tree is \n";

    printInorder(root);

    cout << "\nPostorder traversal of binary tree is \n";

    printPostorder(root);

    return 0;
}
```

Output:


E:\AB_Rafeh Personal\SMI University\Subject Folder's\3rd Semester\Data Structure & Algorithms\Assignment's\Lab Assignment Program's\10.exe
Preorder traversal of binary tree is
1 2 4 5 3
Inorder traversal of binary tree is
4 2 5 1 3
Postorder traversal of binary tree is
4 5 2 3 1
Process exited after 0.1005 seconds with return value 0
Press any key to continue . . .
void printInorder(struct Node* node)
Compilation results...
- Errors: 0
- Warnings: 0
- Output Filename: E:\AB_Rafeh Personal\SMI University\Subject Folder's\3rd Semester\Data Structure & Algorithms\Assignment's\Lab Assignment
- Output Size: 1.83323955535889 MiB
- Compilation Time: 0.09s