Name : Amruta Pede

Reg. no. :2020Bit070

Practical 1

Write c/ c++ code to implement concept of :

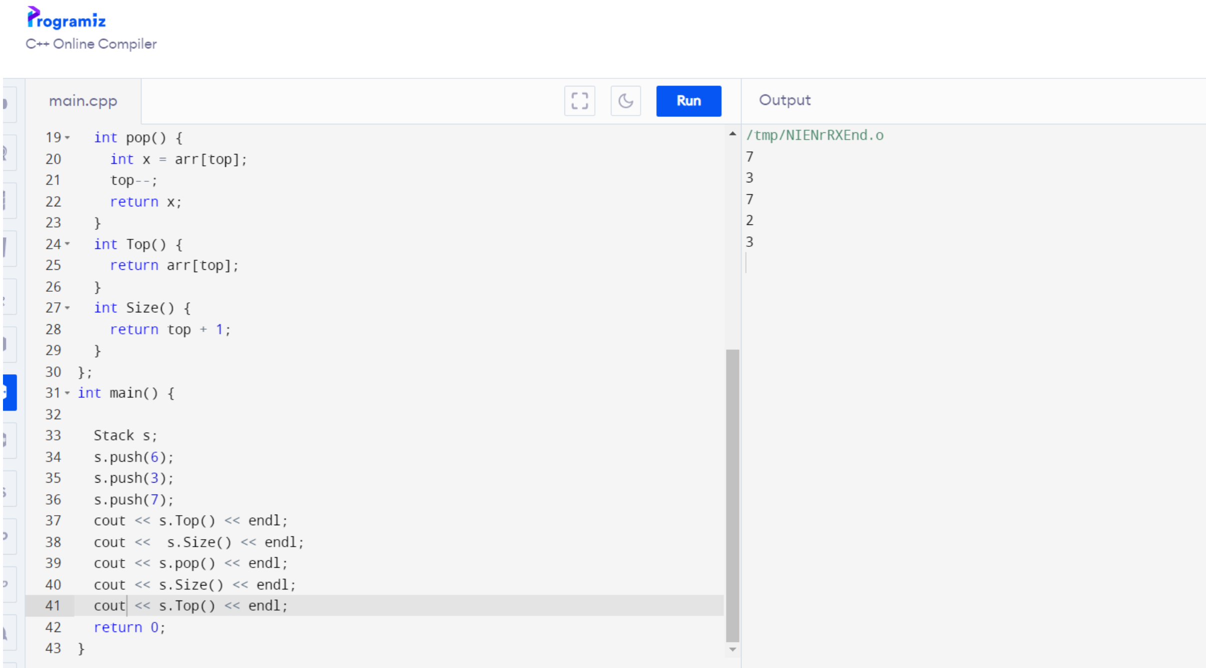
1) Stack
2) Queue
3) Linkedlist
4) Tree
5) Graph

1)Stack

```
#include<bits/stdc++.h>
using namespace std;
class Stack {
 int size;
 int * arr;
 int top;
 public:
  Stack() {
   top = -1;
   size = 1000;
   arr = new int[size];
  }
 void push(int x) {
  top++;
  arr[top] = x;
 }
```

```
  int pop() {
    int x = arr[top];
    top--;
    return x;
  }
  int Top() {
    return arr[top];
  }
  int Size() {
    return top + 1;
  }
};
int main() {

  Stack s;
  s.push(6);
  s.push(3);
  s.push(7);
  cout << "Top of stack is before deleting any element " << s.Top() << endl;
  cout << "Size of stack before deleting any element " << s.Size() << endl;
  cout << "The element deleted is " << s.pop() << endl;
  cout << "Size of stack after deleting an element " << s.Size() << endl;
  cout << "Top of stack after deleting an element " << s.Top() << endl;
  return 0;
}
```

Output:



2)Queue

Question :

Implement Queue Data Structure using Array with all functions like pop, push, top, size, etc. Implement Queue Data Structure using Array with all functions like pop, push, top, size, etc.

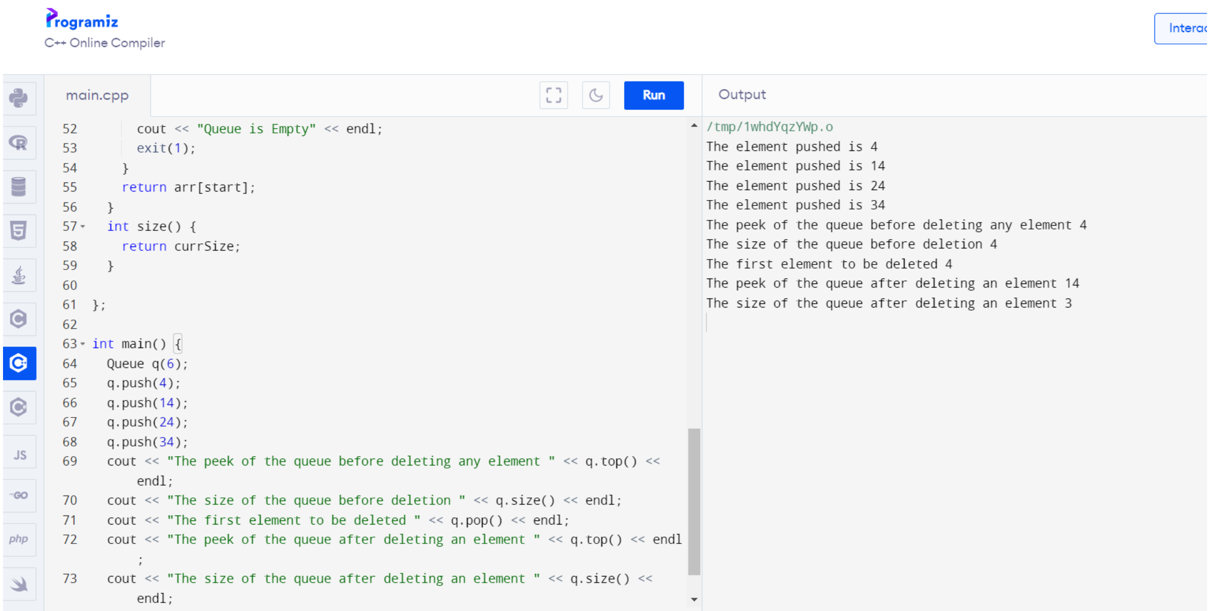
```
#include<bits/stdc++.h>

using namespace std;
class Queue {
  int * arr;
  int start, end, currSize, maxSize;
  public:
    Queue() {
      arr = new int[16];
```

```
    start = -1;
    end = -1;
    currSize = 0;
  }

Queue(int maxSize) {
  ( * this).maxSize = maxSize;
  arr = new int[maxSize];
  start = -1;
  end = -1;
  currSize = 0;
}
void push(int newElement) {
  if (currSize == maxSize) {
    cout << "Queue is full\nExiting..." << endl;
    exit(1);
  }
  if (end == -1) {
    start = 0;
    end = 0;
  } else
    end = (end + 1) % maxSize;
  arr[end] = newElement;
  cout << "The element pushed is " << newElement << endl;
  currSize++;
}
```

```
  int pop() {
    if (start == -1) {
      cout << "Queue Empty\nExiting..." << endl;
    }
    int popped = arr[start];
    if (currSize == 1) {
      start = -1;
      end = -1;
    } else
      start = (start + 1) % maxSize;
    currSize--;
    return popped;
  }
  int top() {
    if (start == -1) {
      cout << "Queue is Empty" << endl;
      exit(1);
    }
    return arr[start];
  }
  int size() {
    return currSize;
  }

};
```

```
int main() {
  Queue q(6);
  q.push(4);
  q.push(14);
  q.push(24);
  q.push(34);
  cout << "The peek of the queue before deleting any element " << q.top() << endl;
  cout << "The size of the queue before deletion " << q.size() << endl;
  cout << "The first element to be deleted " << q.pop() << endl;
  cout << "The peek of the queue after deleting an element " << q.top() << endl;
  cout << "The size of the queue after deleting an element " << q.size() << endl;

  return 0;
}
```

Output:



3)linkedlist :

```
// A complete working C++ program to
// demonstrate all insertion methods
// on Linked List
#include <bits/stdc++.h>
using namespace std;

// A linked list node
class Node
{
    public:
    int data;
    Node *next;
};

// Given a reference (pointer to pointer)
// to the head of a list and an int, inserts
// a new node on the front of the list.
void push(Node** head_ref, int new_data)
{

    // 1. allocate node
    Node* new_node = new Node();

    // 2. put in the data
    new_node->data = new_data;

    // 3. Make next of new node as head
    new_node->next = (*head_ref);
```

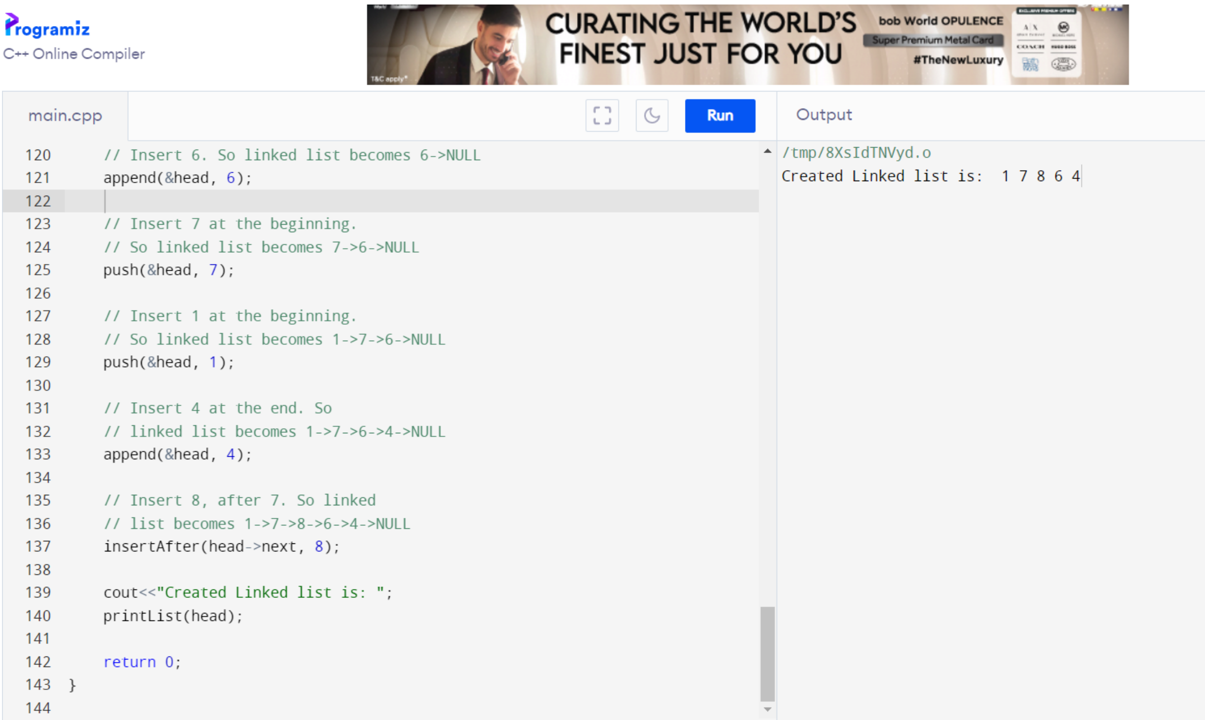
```
    // 4. move the head to point
    // to the new node
    (*head_ref) = new_node;
}

// Given a node prev_node, insert a new
// node after the given prev_node
void insertAfter(Node* prev_node, int new_data)
{
    // 1. check if the given prev_node
    // is NULL
    if (prev_node == NULL)
    {
        cout<<"The given previous node cannot be NULL";
        return;
    }

    // 2. allocate new node
    Node* new_node = new Node();

    // 3. put in the data
    new_node->data = new_data;

    // 4. Make next of new node
    // as next of prev_node
    new_node->next = prev_node->next;

    // 5. move the next of prev_node
    // as new_node
    prev_node->next = new_node;
}
```

```
// Given a reference (pointer to pointer)
// to the head of a list and an int,
// appends a new node at the end
void append(Node** head_ref, int new_data)
{

    // 1. allocate node
    Node* new_node = new Node();

    //used in step 5
    Node *last = *head_ref;

    // 2. put in the data
    new_node->data = new_data;

    /* 3. This new node is going to be
    the last node, so make next of
    it as NULL*/
    new_node->next = NULL;

    /* 4. If the Linked List is empty,
    then make the new node as head */
    if (*head_ref == NULL)
    {
        *head_ref = new_node;
        return;
    }

    /* 5. Else traverse till the last node */
    while (last->next != NULL)
```

```
    {
        last = last->next;
    }

    /* 6. Change the next of last node */
    last->next = new_node;
    return;
}

// This function prints contents of
// linked list starting from head
void printList(Node *node)
{
    while (node != NULL)
    {
        cout<<" "<<node->data;
        node = node->next;
    }
}
// Driver code
int main()
{ // Start with the empty list
    Node* head = NULL;

    // Insert 6. So linked list becomes 6->NULL
    append(&head, 6);

    // Insert 7 at the beginning.
    // So linked list becomes 7->6->NULL
    push(&head, 7);
```

```
// Insert 1 at the beginning.  So linked list becomes 1->7->6->NULL
push(&head, 1);

// Insert 4 at the end. So linked list becomes 1->7->6->4->NULL
append(&head, 4);

// Insert 8, after 7. So linked list becomes 1->7->8->6->4->NULL
insertAfter(head->next, 8);

cout<<"Created Linked list is: ";
printList(head);

return 0;
}
```

Output :

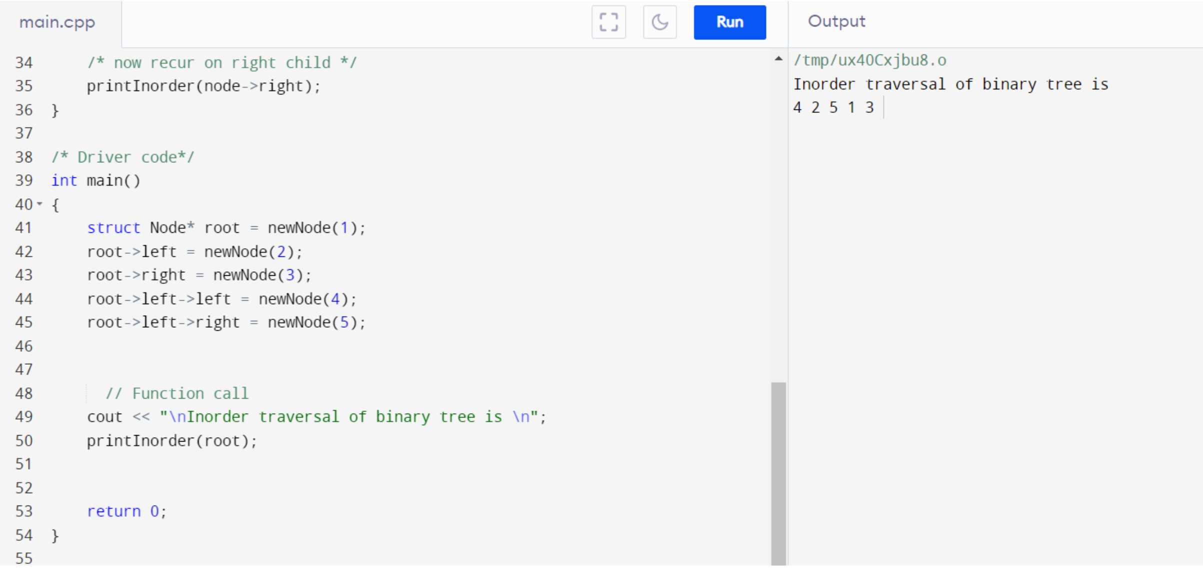


4) Tree :

```
// C++ program of Inorder tree traversals
#include <bits/stdc++.h>
using namespace std;

/* A binary tree node has data, pointer to left child
and a pointer to right child */
struct Node {
   int data;
   struct Node *left, *right;
};

// Utility function to create a new tree node
Node* newNode(int data)
{
   Node* temp = new Node;
   temp->data = data;
   temp->left = temp->right = NULL;
   return temp;
}

/* Given a binary tree, print its nodes in inorder*/
void printInorder(struct Node* node)
{
   if (node == NULL)
      return;

   /* first recur on left child */
```

```
    printInorder(node->left);

    /* then print the data of node */
    cout << node->data << " ";

    /* now recur on right child */
    printInorder(node->right);
}

/* Driver code*/
int main()
{
    struct Node* root = newNode(1);
    root->left = newNode(2);
    root->right = newNode(3);
    root->left->left = newNode(4);
    root->left->right = newNode(5);


      // Function call
    cout << "\nInorder traversal of binary tree is \n";
    printInorder(root);


    return 0;
}
```

Output :



5)Graph :

```
// A simple representation of graph using STL

#include <bits/stdc++.h>

using namespace std;


// A utility function to add an edge in an

// undirected graph.

void addEdge(vector<int> adj[], int u, int v)

{

    adj[u].push_back(v);

    adj[v].push_back(u);

}


// A utility function to print the adjacency list

// representation of graph

void printGraph(vector<int> adj[], int V)

{

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

        cout << "\n Adjacency list of vertex " << v

            << "\n head ";
```

```
        for (auto x : adj[v])
            cout << "-> " << x;
        printf("\n");
    }
}

// Driver code
int main()
{
    int V = 5;
    vector<int> adj[V];
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 4);
    addEdge(adj, 1, 2);
    addEdge(adj, 1, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 3);
    addEdge(adj, 3, 4);
    printGraph(adj, V);
    return 0;
}
```

Output :

main.cpp

Run

```cpp
        cout << "\n Adjacency list of vertex " << v
             << "\n head ";
        for (auto x : adj[v])
            cout << "-> " << x;
        printf("\n");
    }
}

// Driver code
int main()
{
    int V = 5;
    vector<int> adj[V];
    addEdge(adj, 0, 1);
    addEdge(adj, 0, 4);
    addEdge(adj, 1, 2);
    addEdge(adj, 1, 3);
    addEdge(adj, 1, 4);
    addEdge(adj, 2, 3);
    addEdge(adj, 3, 4);
    printGraph(adj, V);
    return 0;
}
```

Output

```
/tmp/giTyFQJ26O.o
Adjacency list of vertex 0
 head -> 1-> 4

 Adjacency list of vertex 1
 head -> 0-> 2-> 3-> 4
Adjacency list of vertex 2
 head -> 1-> 3

 Adjacency list of vertex 3
 head -> 1-> 2-> 4

 Adjacency list of vertex 4
 head -> 0-> 1-> 3
```