**Q1**

**CODE**:

// Binary Search Tree operations in C++

#include <iostream>

using namespace std;

class node {

public:

  int key;

  node \*left, \*right;

};

// Create a node

node \*newNode(int item) {

  node\* temp = new node();

  temp->key = item;

  temp->left = temp->right = NULL;

  return temp;

}

void inorder(node \*root) {

  if (root != NULL) {

    inorder(root->left);

    cout << root->key << " -> ";

    inorder(root->right);

  }

}

// Insert a node

 node \*insert( node \*node, int key) {

  if (node == NULL)

    return newNode(key);

  if (key < node->key)

    node->left = insert(node->left, key);

  else

    node->right = insert(node->right, key);

  return node;

}

node\* insert\_array(node\* root, int x[],int z)

{

    int j = z-1;

    for(int i=0;i<j;i++)

    {

        root = insert(root,x[i]);

    }

    return root;

}

int main() {

  node \*root = NULL;

  int x[] = {1,2,3,4,5,6,7};

  int z = sizeof(x)/sizeof(x[0]);

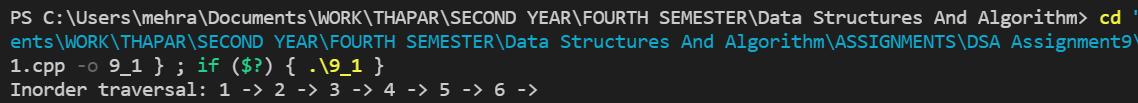
  root = insert\_array(root,x,z);

  cout << "Inorder traversal: ";

  inorder(root);

}

**OUTPUT:**



**Q2**

**CODE**:

// Binary Search Tree operations in C++

#include <iostream>

using namespace std;

class node {

public:

  int key;

  node \*left, \*right;

};

// Create a node

node \*newNode(int item) {

  node\* temp = new node();

  temp->key = item;

  temp->left = temp->right = NULL;

  return temp;

}

void inorder(node \*root) {

  if (root != NULL) {

    inorder(root->left);

    cout << root->key << " -> ";

    inorder(root->right);

  }

}

// Insert a node

 node \*insert( node \*node, int key) {

  if (node == NULL)

    return newNode(key);

  if (key < node->key)

    node->left = insert(node->left, key);

  else

    node->right = insert(node->right, key);

  return node;

}

int main() {

  node \*root = NULL;

  root = insert(root, 8);

  root = insert(root, 3);

  root = insert(root, 1);

  root = insert(root, 6);

  root = insert(root, 7);

  root = insert(root, 10);

  root = insert(root, 14);

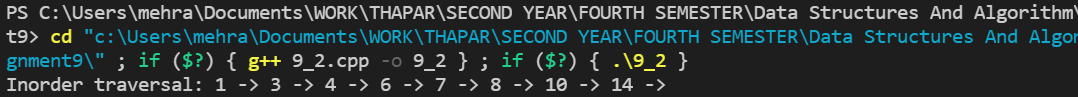
  root = insert(root, 4);

  cout << "Inorder traversal: ";

  inorder(root);

}

**OUTPUT:**



**Q3**

**CODE**:

// Binary Search Tree operations in C++

#include <iostream>

#include <string>

using namespace std;

class node {

public:

  int key;

  node \*left, \*right;

};

// Create a node

node \*newNode(int item) {

  node\* temp = new node();

  temp->key = item;

  temp->left = temp->right = NULL;

  return temp;

}

void inorder(node \*root) {

  if (root != NULL) {

    inorder(root->left);

    cout << root->key << " -> ";

    inorder(root->right);

  }

}

// Insert a node

 node \*insert( node \*node, int key) {

  if (node == NULL)

    return newNode(key);

  if (key < node->key)

    node->left = insert(node->left, key);

  else

    node->right = insert(node->right, key);

  return node;

}

string search(node\* root,int number)

{

    if(root==NULL)

        return "NOT PRESENT";

    if(number==root->key)

        return "PRESENT";

    if(number<root->key)

        return search(root->left,number);

    if(number>root->key)

        return search(root->right,number);

}

int main() {

  node \*root = NULL;

  root = insert(root, 8);

  root = insert(root, 3);

  root = insert(root, 1);

  root = insert(root, 6);

  root = insert(root, 7);

  root = insert(root, 10);

  root = insert(root, 14);

  root = insert(root, 4);

  cout << "Inorder traversal: ";

  inorder(root);

  cout<<"\nWhich element do you want to search\n";

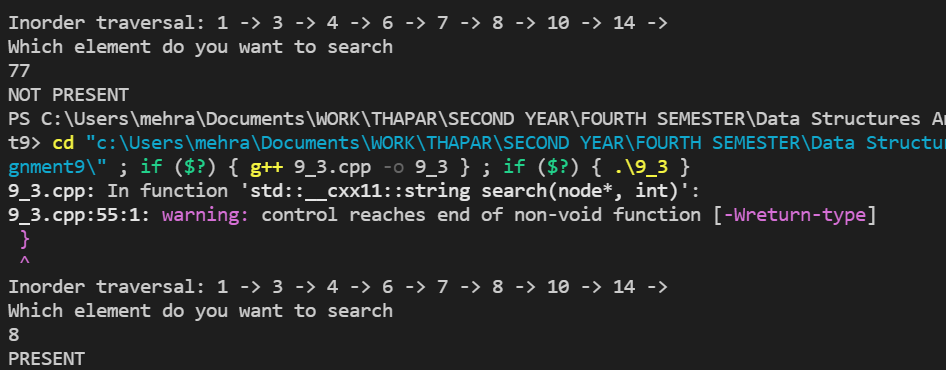
  int x;

  cin>>x;

  cout<<search(root,x)<<endl;

}

**OUTPUT:**



**Q4**

**CODE:**

// Binary Search Tree operations in C++

#include <iostream>

using namespace std;

class node {

public:

  int key;

  node \*left, \*right;

};

// Create a node

node \*newNode(int item) {

  node\* temp = new node();

  temp->key = item;

  temp->left = temp->right = NULL;

  return temp;

}

void inorder(node \*root) {

  if (root != NULL) {

    inorder(root->left);

    cout << root->key << " -> ";

    inorder(root->right);

  }

}

// Insert a node

 node \*insert( node \*node, int key) {

  if (node == NULL)

    return newNode(key);

  if (key < node->key)

    node->left = insert(node->left, key);

  else

    node->right = insert(node->right, key);

  return node;

}

int main() {

  node \*root = NULL;

  root = insert(root, 8);

  root = insert(root, 3);

  root = insert(root, 1);

  root = insert(root, 6);

  root = insert(root, 7);

  root = insert(root, 10);

  root = insert(root, 14);

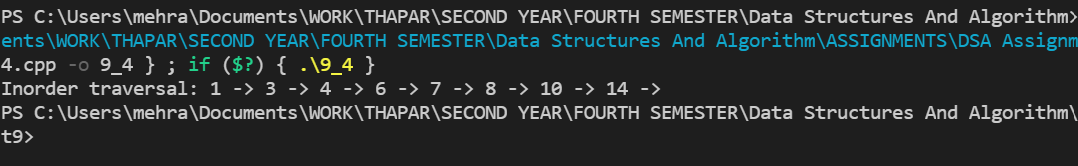
  root = insert(root, 4);

  cout << "Inorder traversal: ";

  inorder(root);

}

**OUTPUT:**



**Q5**

**CODE**:

// Binary Search Tree operations in C++

#include <iostream>

using namespace std;

class node {

public:

  int key;

  node \*left, \*right;

};

// Create a node

node \*newNode(int item) {

  node\* temp = new node();

  temp->key = item;

  temp->left = temp->right = NULL;

  return temp;

}

void inorder(node \*root) {

  if (root != NULL) {

    inorder(root->left);

    cout << root->key << " -> ";

    inorder(root->right);

  }

}

// Insert a node

 node \*insert( node \*node, int key) {

  if (node == NULL)

    return newNode(key);

  if (key < node->key)

    node->left = insert(node->left, key);

  else

    node->right = insert(node->right, key);

  return node;

}

node \*minValueNode(node \*n) {

   node \*current = n;

  // Find the leftmost leaf

  while (current && current->left != NULL)

    current = current->left;

  return current;

}

// Deleting a node

node \*deleteNode(node \*root, int key) {

  // Return if the tree is empty

  if (root == NULL) return root;

  // Find the node to be deleted

  if (key < root->key)

    root->left = deleteNode(root->left, key);

  else if (key > root->key)

    root->right = deleteNode(root->right, key);

  else {

    // If the node is with only one child or no child

    if (root->left == NULL) {

       node \*temp = root->right;

      free(root);

      return temp;

    } else if (root->right == NULL) {

       node \*temp = root->left;

      free(root);

      return temp;

    }

    // If the node has two children

     node \*temp = minValueNode(root->right);

    // Place the inorder successor in position of the node to be deleted

    root->key = temp->key;

    // Delete the inorder successor

    root->right = deleteNode(root->right, temp->key);

  }

  return root;

}

int main() {

  node \*root = NULL;

  root = insert(root, 8);

  root = insert(root, 3);

  root = insert(root, 1);

  root = insert(root, 6);

  root = insert(root, 7);

  root = insert(root, 10);

  root = insert(root, 14);

  root = insert(root, 4);

  cout << "Inorder traversal: ";

  inorder(root);

  cout << "\nAfter deleting 10\n";

  root = deleteNode(root, 10);

  cout << "Inorder traversal: ";

  inorder(root);

}

**OUTPUT:**

