**Task 1: Inserting a node in a binary tree [used level order]**

#include <iostream>

#include <queue>

using namespace std;

class node {

public:

node\* left;

node\* right;

int data;

node(int i) {

data = i;

left = NULL;

right = NULL;

}

};

node\* insert\_in\_binary\_tree(node\* root, int i) {

if (root == NULL) {

node\* newnode = new node(i);

root = newnode;

return root;

}

else {

node\* newnode = new node(i);

queue<node\*> q;

q.push(root);

while (!q.empty()) {

node\* temp = q.front();

q.pop();

if (temp->left != NULL) {

q.push(temp->left);

}

else {

temp->left = newnode;

return root;

}

if (temp->right != NULL) {

q.push(temp->right);

}

else {

temp->right = newnode;

return root;

}

}

}

}

void inorder(node\* temp)

{

if (temp == NULL)

return;

inorder(temp->left);

cout << temp->data << ' ';

inorder(temp->right);

}

void display(node\* current\_node) {

if (current\_node != NULL) {

cout << current\_node->data << endl;

display(current\_node->left);

display(current\_node->right);

}

}

int main() {

node\* root = new node(10);

node\* n11 = new node(11);

node\* n9 = new node(9);

node\* n7 = new node(7);

node\* n15 = new node(15);

node\* n8 = new node(8);

root->left = n11;

root->right = n9;

n11->left = n7;

n11->right = n15;

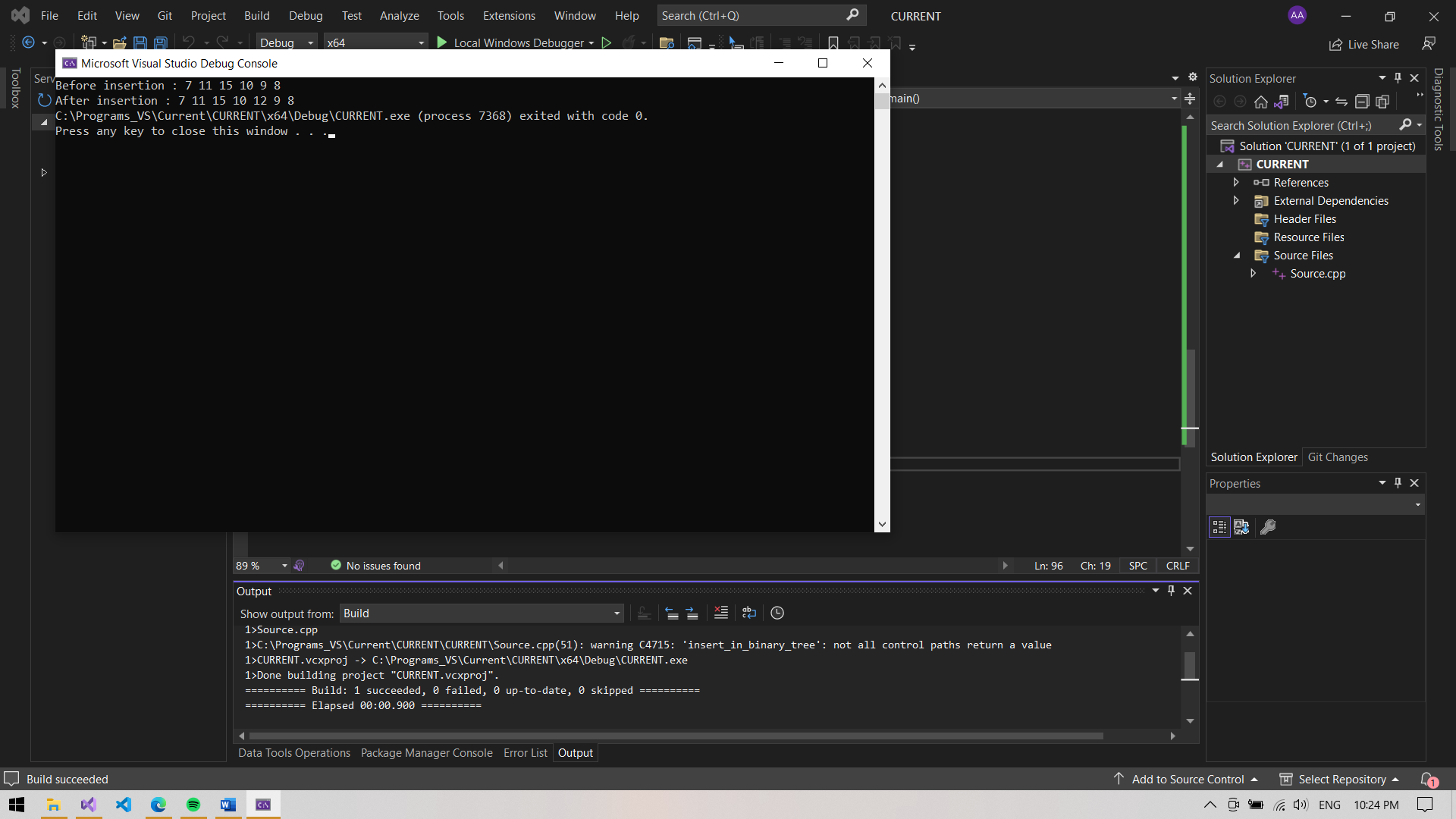
//n9->left = n15; tested it to confirm if it works

n9->right = n8;

root = insert\_in\_binary\_tree(root, 12);

inorder(root);

}



**Task 2: Searching a data in a binary tree**

#include <iostream>

#include <queue>

using namespace std;

class node {

public:

node\* left;

node\* right;

int data;

node(int i) {

data = i;

left = NULL;

right = NULL;

}

};

node\* search\_in\_binary\_tree(node\* root, int i) {

if (root == NULL) {

cout << "The tree is empty" << endl;

}

else {

queue<node\*> q;

q.push(root);

node\* temp = q.front();

while (!q.empty()) {

temp = q.front();

q.pop();

if (temp->data == i) {

cout << "found the data" << endl;

return temp;

}

if (temp->left != NULL) {

q.push(temp->left);

}

if (temp->right != NULL) {

q.push(temp->right);

}

}

if (temp->data != i) {

cout << "The data is not found" << endl;

}

}

}

void inorder(node\* temp)

{

if (temp == NULL)

return;

inorder(temp->left);

cout << temp->data << ' ';

inorder(temp->right);

}

void display(node\* current\_node) {

if (current\_node != NULL) {

cout << current\_node->data << endl;

display(current\_node->left);

display(current\_node->right);

}

}

int main() {

node\* root = new node(10);

node\* n11 = new node(11);

node\* n9 = new node(9);

node\* n7 = new node(7);

node\* n15 = new node(15);

node\* n8 = new node(8);

root->left = n11;

root->right = n9;

n11->left = n7;

n9->left = n15;

n9->right = n8;

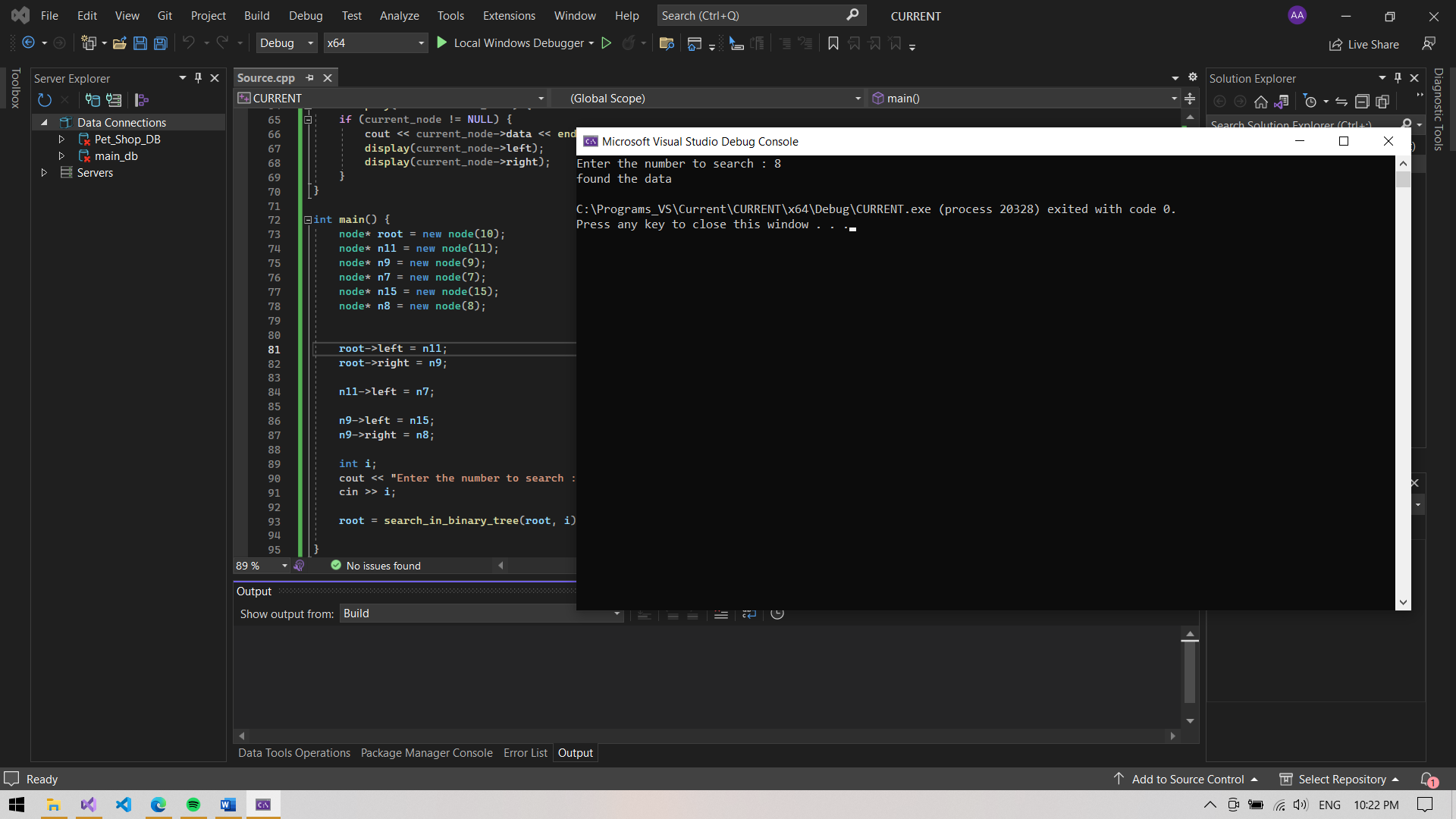
int i;

cout << "Enter the number to search : ";

cin >> i;

root = search\_in\_binary\_tree(root, i);

}



**Task 2: Searching minimum data in a binary tree**

#include <iostream>

#include <queue>

using namespace std;

class node {

public:

node\* left;

node\* right;

int data;

node(int i) {

data = i;

left = NULL;

right = NULL;

}

};

void minimum\_in\_binary\_tree(node\* root) {

if (root == NULL) {

cout << "The tree is empty" << endl;

}

else {

queue<node\*> q;

q.push(root);

int minimum = root->data;

node\* temp = q.front();

while (!q.empty()) {

temp = q.front();

q.pop();

if (temp->data < minimum) {

minimum = temp->data;

}

if (temp->left != NULL) {

q.push(temp->left);

}

if (temp->right != NULL) {

q.push(temp->right);

}

}

cout << "Minimum data : " << minimum << endl;

}

}

void inorder(node\* temp)

{

if (temp == NULL)

return;

inorder(temp->left);

cout << temp->data << ' ';

inorder(temp->right);

}

void display(node\* current\_node) {

if (current\_node != NULL) {

cout << current\_node->data << endl;

display(current\_node->left);

display(current\_node->right);

}

}

int main() {

node\* root = new node(10);

node\* n11 = new node(11);

node\* n9 = new node(9);

node\* n7 = new node(7);

node\* n15 = new node(15);

node\* n8 = new node(8);

root->left = n11;

root->right = n9;

n11->left = n7;

n9->left = n15;

n9->right = n8;

minimum\_in\_binary\_tree(root);

}

