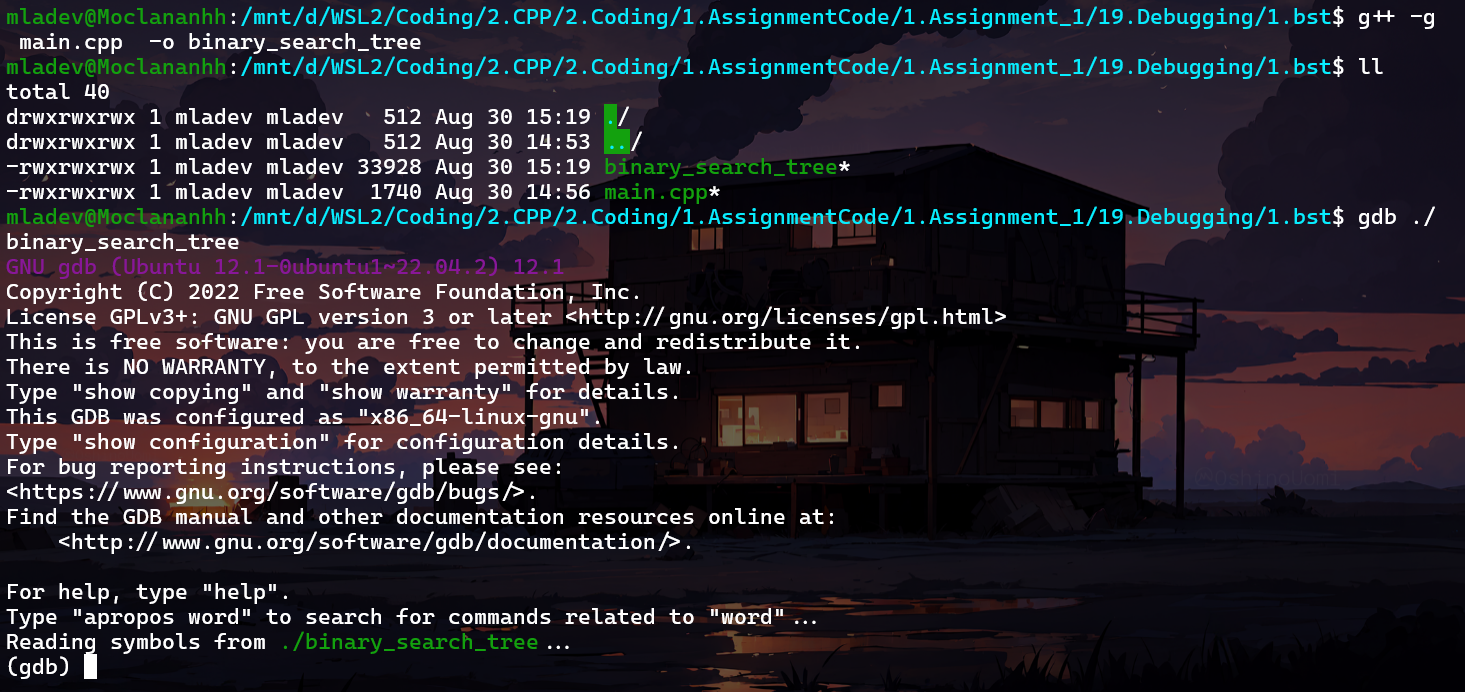
## Assignment\_30-08-2024

1 Binary search tree

To practice gdb options and fix issues with gdb commands, let's first compile the given C++ code with the **-g** flag to include debugging symbols.



Since the code seems to be incomplete, let's identify the issue. We notice that the insert function is missing the logic to handle the case when the key to be inserted is smaller than the node's key.

This is updated version:

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

{

    // If the tree is empty, return a new node

    if (node == NULL)

        return new Node(key);

    // If the key is already present in the tree,

    // return the node

    if (node->key == key)

        return node;

    // Otherwise, recur down the tree/ If the key

    // to be inserted is greater than the node's key,

    // insert it in the right subtree

    if (node->key < key)

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

    // If the key to be inserted is smaller than

    // the node's key,insert it in the left subtree

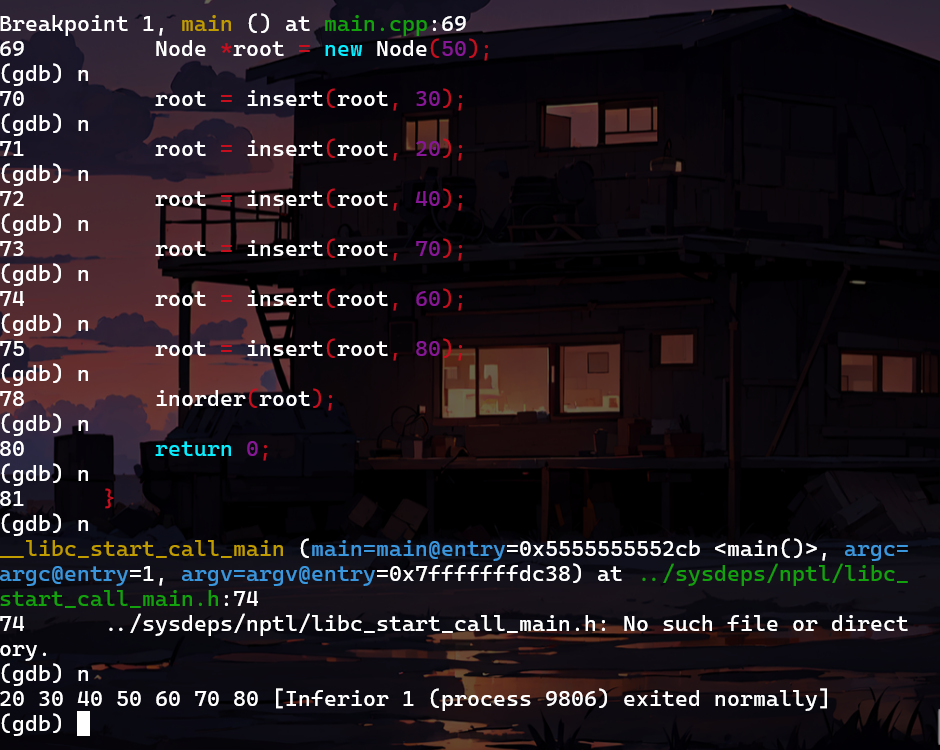
    else

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

    // Return the (unchanged) node pointer

    return node;

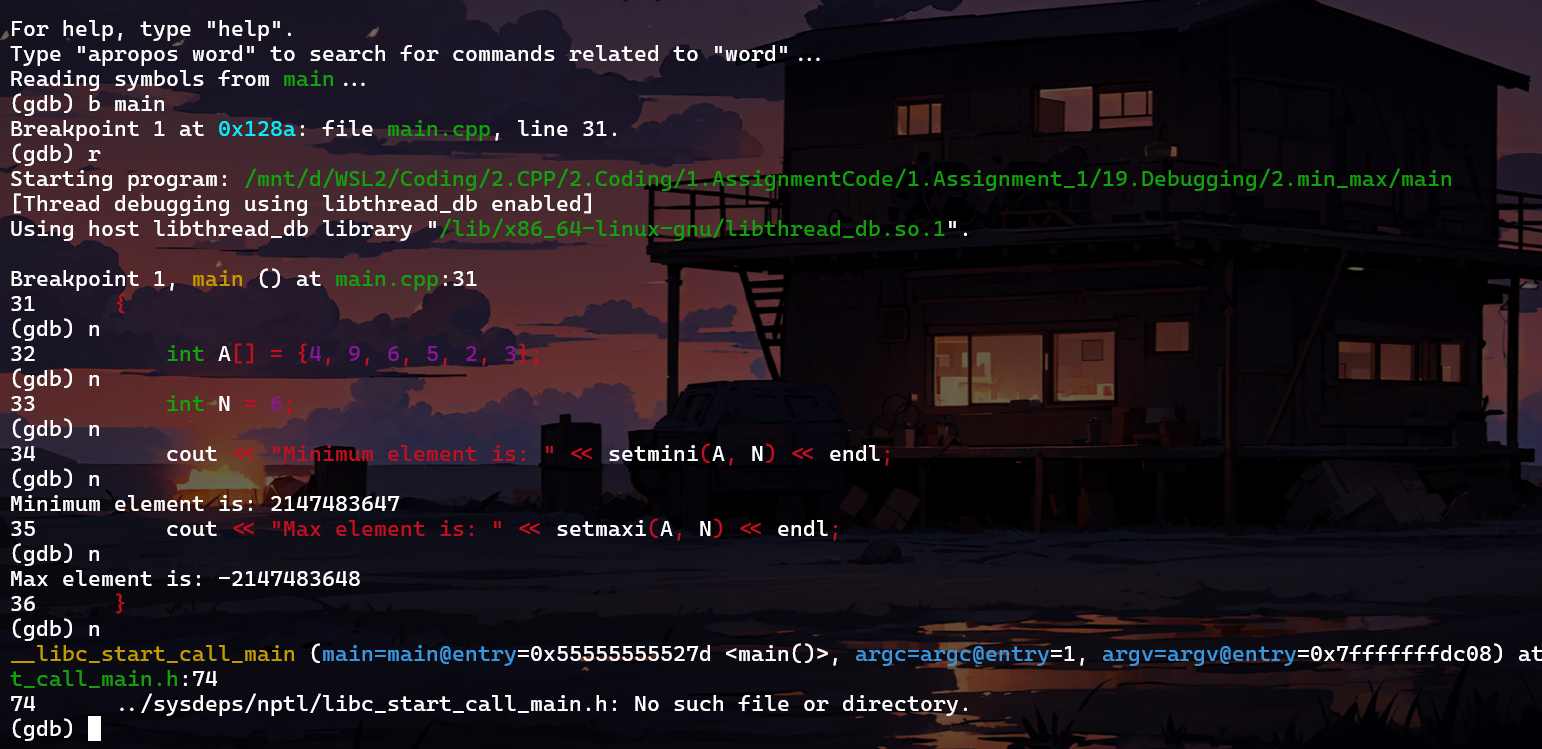
}



2 Min max

Compile the program and go to debug mode to check code





Issues:

- In the setmini function, the code is missing the assignment of the minimum value to mini.

- In the setmaxi function, the loop condition is incorrect. It should be i < N instead of i > N.

Fixed code:

int setmini(int A[], int N)

{

    int mini = INT\_MAX;

    for (int i = 0; i < N; i++)

    {

        if (A[i] < mini)

        {

            mini = A[i]; // assign the minimum value to mini

        }

    }

    return mini;

}

int setmaxi(int A[], int N)

{

    int maxi = INT\_MIN;

    for (int i = 0; i < N; i++) // fix the loop condition

    {

        if (A[i] > maxi)

        {

            maxi = A[i];

        }

    }

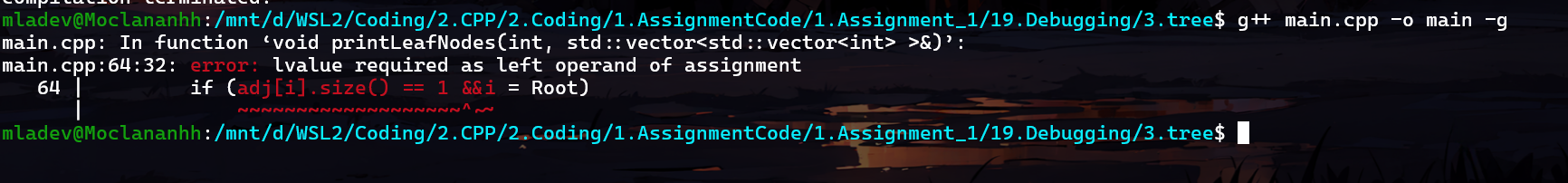
    return maxi;

}



3 tree

Run this code to check



Problem here:

### ****1. Logical Error in**** printLeafNodes ****Function****

* The condition if (adj[i].size() == 1 && i = Root) is incorrect. It should be if (adj[i].size() == 1 && i != Root) to correctly identify leaf nodes. Additionally, = is an assignment operator and should be != for comparison.

### ****2. Logical Error in**** printDegrees ****Function****

* Similarly, the condition if (i = Root) should be if (i == Root).

### ****3. Array Initialization Issue in**** printChildren ****Function****

* The initialization of the vis array using {0} is incorrect for dynamic arrays. Instead, we should use fill\_n or vector<int> to handle this dynamically.

### ****4. Root Node Condition in**** printParents ****Function****

* The condition if (parent == 0) should check if the parent is the root, so parent == -1 is a better choice for indicating the root node.

Code fixed:

#include <bits/stdc++.h>

using namespace std;

// Function to add an edge between vertices x and y

void addEdge(int x, int y, vector<vector<int>> &adj)

{

    adj[x].push\_back(y);

    adj[y].push\_back(x);

}

// Function to print the parent of each node

void printParents(int node, vector<vector<int>> &adj, int parent)

{

    // current node is Root, thus, has no parent

    if (parent == -1)

        cout << node << "->Root" << endl;

    else

        cout << node << "->" << parent << endl;

    // Using DFS

    for (auto cur : adj[node])

        if (cur != parent)

            printParents(cur, adj, node);

}

// Function to print the children of each node

void printChildren(int Root, vector<vector<int>> &adj)

{

    // Queue for the BFS

    queue<int> q;

    // pushing the root

    q.push(Root);

    // visit vector to keep track of nodes that have been visited

    vector<int> vis(adj.size(), 0);

    // BFS

    while (!q.empty())

    {

        int node = q.front();

        q.pop();

        vis[node] = 1;

        cout << node << "-> ";

        for (auto cur : adj[node])

        {

            if (vis[cur] == 0)

            {

                cout << cur << " ";

                q.push(cur);

            }

        }

        cout << endl;

    }

}

// Function to print the leaf nodes

void printLeafNodes(int Root, vector<vector<int>> &adj)

{

    // Leaf nodes have only one edge and are not the root

    for (int i = 1; i < adj.size(); i++)

        if (adj[i].size() == 1 && i != Root)

            cout << i << " ";

    cout << endl;

}

// Function to print the degrees of each node

void printDegrees(int Root, vector<vector<int>> &adj)

{

    for (int i = 1; i < adj.size(); i++)

    {

        cout << i << ": ";

        // Root has no parent, thus, its degree is equal to the edges it is connected to

        if (i == Root)

            cout << adj[i].size() << endl;

        else

            cout << adj[i].size() - 1 << endl;

    }

}

// Driver code

int main()

{

    // Number of nodes

    int N = 7, Root = 1;

    // Adjacency list to store the tree

    vector<vector<int>> adj(N + 1, vector<int>());

    // Creating the tree

    addEdge(1, 2, adj);

    addEdge(1, 3, adj);

    addEdge(1, 4, adj);

    addEdge(2, 5, adj);

    addEdge(2, 6, adj);

    addEdge(4, 7, adj);

    // Printing the parents of each node

    cout << "The parents of each node are:" << endl;

    printParents(Root, adj, -1); // -1 indicates that the root has no parent

    // Printing the children of each node

    cout << "The children of each node are:" << endl;

    printChildren(Root, adj);

    // Printing the leaf nodes in the tree

    cout << "The leaf nodes of the tree are:" << endl;

    printLeafNodes(Root, adj);

    // Printing the degrees of each node

    cout << "The degrees of each node are:" << endl;

    printDegrees(Root, adj);

}



