```
#include <iostream> // --> Used for standard input and output
#include <queue> // --> Used for queue data structure in BFS
#include <omp.h> // --> Required for OpenMP parallel processing
using namespace std; // --> Avoids prefixing std:: with standard library names

class Node // --> Defines the structure of a tree node
{
public:
  Node *left, *right; // --> Pointers to left and right child nodes
  int data; // --> Data value of the node
};

class BreadthFS // --> Class to perform insert and BFS operations
{
public:
  Node *insert(Node *root, int data); // --> Function to insert a node into tree
  void bfs(Node *root); // --> Function to perform parallel BFS traversal
};

// Insert a new node using level-order insertion
Node *BreadthFS::insert(Node *root, int data) // --> Inserts node into tree in level order
{
  if (!root) // --> If tree is empty, create root
  {
    root = new Node; // --> Allocate memory for root
    root->left = nullptr; // --> Initialize left child to null
    root->right = nullptr; // --> Initialize right child to null
    root->data = data; // --> Set data in root
    return root; // --> Return the root node
  }
```

```
std::queue<Node *> q; // --> Queue for level-order traversal
q.push(root); // --> Push root node into the queue

while (!q.empty()) // --> Loop until queue is empty
{
  Node *current = q.front(); // --> Get the front node in queue
  q.pop(); // --> Remove the front node from queue

  if (!current->left) // --> If left child is null, insert here
  {
    current->left = new Node; // --> Create new left child
    current->left->left = nullptr; // --> Initialize left child of new node
    current->left->right = nullptr; // --> Initialize right child of new node
    current->left->data = data; // --> Set data for new node
    return root; // --> Return root after insertion
  }
  else
  {
    q.push(current->left); // --> Push left child into queue for further check
  }

  if (!current->right) // --> If right child is null, insert here
  {
    current->right = new Node; // --> Create new right child
    current->right->left = nullptr; // --> Initialize left child of new node
    current->right->right = nullptr; // --> Initialize right child of new node
    current->right->data = data; // --> Set data for new node
    return root; // --> Return root after insertion
  }
  else
  {
        q.push(current->right); // --> Push right child into queue for further check
      }
    }
    return root; // --> Return root if insertion completes
}

// Parallel BFS using OpenMP
void BreadthFS::bfs(Node *root) // --> Performs BFS using parallel processing
{
    if (!root) // --> If tree is empty, return
      return;

    queue<Node *> q; // --> Queue for BFS traversal
    q.push(root); // --> Push root node into the queue

    while (!q.empty()) // --> Loop until queue becomes empty
    {
      int level_size = q.size(); // --> Get number of nodes at current level

#pragma omp parallel for // --> Parallelize loop for each node at this level
      for (int i = 0; i < level_size; i++) // --> Iterate over nodes in current level
      {
        Node *current = nullptr; // --> Declare current node pointer

#pragma omp critical // --> Ensure only one thread accesses queue at a time
        {
          current = q.front(); // --> Get front node from queue
          q.pop(); // --> Remove node from queue
          cout << current->data << "\t"; // --> Print node data
        }
```

```
#pragma omp critical // --> Ensure safe queue insertion from multiple threads
        {
            if (current->left) // --> If left child exists
                q.push(current->left); // --> Add left child to queue
            if (current->right) // --> If right child exists
                q.push(current->right); // --> Add right child to queue
        }
      }
   }
}

int main() // --> Main function
{
   BreadthFS bfs; // --> Create object of BreadthFS
   Node *root = nullptr; // --> Initialize root to null
   int data; // --> Variable to store input data
   char choice; // --> Variable to store user choice

   cout << "\n\nName: Shriharsh Deshmukh\nRoll No.62 \t Div.A\n\n"; // --> Display student
details

   do // --> Loop to insert multiple nodes
   {
      cout << "Enter data: "; // --> Prompt user for data
      cin >> data; // --> Read input data
      root = bfs.insert(root, data); // --> Insert data into tree
      cout << "Insert another node? (y/n): "; // --> Ask to continue
      cin >> choice; // --> Read user choice
   } while (choice == 'y' || choice == 'Y'); // --> Continue loop if user enters 'y' or 'Y'

   cout << "BFS Traversal:\n"; // --> Display BFS heading
```

```
    bfs.bfs(root); // --> Call BFS function


    return 0; // --> Return 0 indicating successful execution
}


// Run Commands:
// g++ -fopenmp -o parallel_bfs 1_Breadth_First_Search.cpp // --> Compile the code with OpenMP
// .\parallel_bfs // --> Run the executable
```