

```

#include<iostream>

#include<stdlib.h>

#include<queue>

#include<omp.h> // Include OpenMP header

using namespace std;

class node
{
    public:
        node *left, *right;
        int data;
};

class Breadthfs
{
    public:
        node *insert(node *, int);
        void bfs(node *);
};

node *insert(node *root, int data)
// inserts a node in tree
{
    if(!root)
    {
        root = new node;
        root->left = NULL;
        root->right = NULL;
        root->data = data;
        return root;
    }

    queue<node *> q;
    q.push(root);
    while(!q.empty())

```

```

{
    node *temp = q.front();
    q.pop();
    if(temp->left == NULL)
    {
        temp->left = new node;
        temp->left->left = NULL;
        temp->left->right = NULL;
        temp->left->data = data;
        return root;
    }
    else
    {
        q.push(temp->left);
    }
    if(temp->right == NULL)
    {
        temp->right = new node;
        temp->right->left = NULL;
        temp->right->right = NULL;
        temp->right->data = data;
        return root;
    }
    else
    {
        q.push(temp->right);
    }
}
}

void bfs(node *head)
{

```

```

queue<node*> q;

q.push(head);

int qSize;

while (!q.empty())
{
    qSize = q.size();

    // Get the number of threads OpenMP will use

    #pragma omp parallel
    {
        // Print the number of threads in the parallel region

        if (omp_get_thread_num() == 0) // Print only once by thread 0
            cout << "\nNumber of threads: " << omp_get_num_threads() << endl;

        // Parallel for processing nodes in the current level

        #pragma omp for
        for (int i = 0; i < qSize; i++)
        {
            node* currNode;

            #pragma omp critical
            {
                currNode = q.front();

                q.pop();

                cout << "\t" << currNode->data;
            }

            #pragma omp critical
            {
                if(currNode->left) // Push parent's left node in queue
                    q.push(currNode->left);

                if(currNode->right) // Push parent's right node in queue
                    q.push(currNode->right);
            }
        }
    }
}

```

```

    }
}
}
int main()
{
    node *root = NULL;

    int data;

    char ans;

    do
    {
        cout << "\nEnter data: ";

        cin >> data;

        root = insert(root, data);

        cout << "Do you want to insert one more node? (y/n): ";

        cin >> ans;

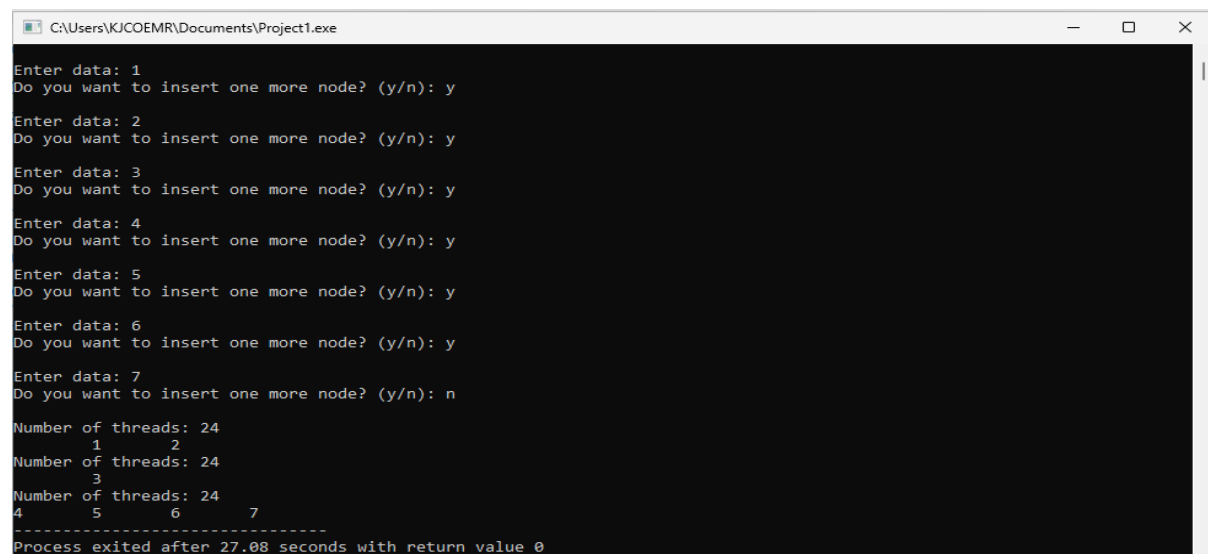
    } while(ans == 'y' || ans == 'Y');

    bfs(root);

    return 0;
}

```

Output:



```

C:\Users\KJ\COEMR\Documents\Project1.exe
Enter data: 1
Do you want to insert one more node? (y/n): y
Enter data: 2
Do you want to insert one more node? (y/n): y
Enter data: 3
Do you want to insert one more node? (y/n): y
Enter data: 4
Do you want to insert one more node? (y/n): y
Enter data: 5
Do you want to insert one more node? (y/n): y
Enter data: 6
Do you want to insert one more node? (y/n): y
Enter data: 7
Do you want to insert one more node? (y/n): n
Number of threads: 24
1      2
Number of threads: 24
3
Number of threads: 24
4      5      6      7
-----
Process exited after 27.08 seconds with return value 0

```