**TASK-1.1 Name: Tanaya Balaji Dahatonde**

1. **Write code for balance bracket**

#include<stdio.h>

int main()

{

char Exp[50];

int x=0, i=0;

printf("Enter an Expression");

scanf("%s", Exp);  .

while(Exp [i]!= '\0')

{

if(Exp [i]=='(')

{

x++;

}

else if(Exp [i]==')')

{

x--;

if(x<0)

break;

}

i++;

}

if(x==0)

{

printf("Expression is balanced");

}

else

{

printf("Expression is unbalanced");

}

return 0;

}

**2) Given “n” ropes of different lengths, connect them into a single rope with minimum cost. Assume that the cost to connect two ropes is the same as the sum of their lengths. (Hint: Use Priority Queue)**

#include <iostream>

#include <queue>

#include <vector>

using namespace std;

int findMinCost(vector<int> const &prices)

{

priority\_queue<int, vector<int>, greater<int>> pq(prices.begin(), prices.end());

int cost = 0;

while (pq.size() > 1)

{

int x = pq.top();

pq.pop();

int y = pq.top();

pq.pop();

int sum = x + y;

pq.push(sum);

cost += sum;

}

return cost;

}

int main()

{

vector<int> prices = { 5, 4, 2, 8 };

cout << "The minimum cost is " << findMinCost(prices);

return 0;

}

**3) Implement binary search tree**

#include <stdio.h>

#include <stdlib.h>

struct node {

    int key;

    struct node \*left, \*right;

};

// A utility function to create a new BST node

struct node\* newNode(int item)

{

    struct node\* temp

        = (struct node\*)malloc(sizeof(struct node));

    temp->key = item;

    temp->left = temp->right = NULL;

    return temp;

}

// A utility function to do inorder traversal of BST

void inorder(struct node\* root)

{

    if (root != NULL) {

        inorder(root->left);

        printf("%d \n", root->key);

        inorder(root->right);

    }

}

/\* A utility function to insert

   a new node with given key in

 \* BST \*/

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

{

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

    if (node == NULL)

        return newNode(key);

    /\* Otherwise, recur down the tree \*/

    if (key < node->key)

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

    else if (key > node->key)

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

    /\* return the (unchanged) node pointer \*/

    return node;

}

// Driver Code

int main()

{

    struct node\* root = NULL;

    root = insert(root, 50);

    insert(root, 30);

    insert(root, 20);

    insert(root, 40);

    insert(root, 70);

    insert(root, 60);

    insert(root, 80);

    // print inoder traversal of the BST

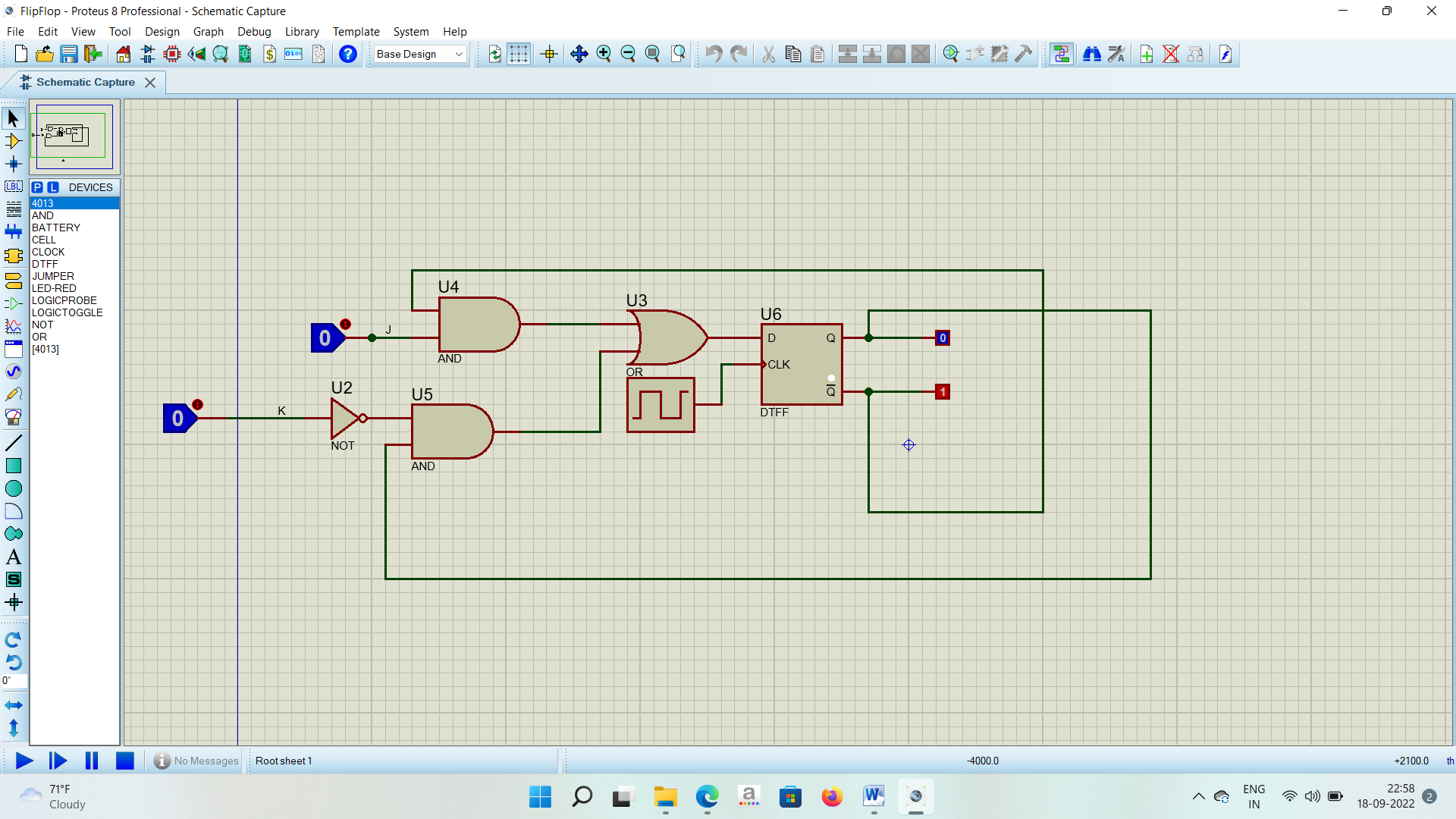
    inorder(root);

    return 0;

}

**TASK-1.2**

**3) Construct JK flip flop using D flip flop - @member3**

****