**TRF LEVEL 2 WORKSHOP**

**TASK 1.1**

c. Balance Bracket

#include <stdio.h>

#include <string.h>

#include <stdlib.h>

struct Node

{

    char data;

    struct Node \*next;

}\*top=NULL;

void push(char x)

{

    struct Node \*t;

    t = (struct Node\*)malloc(sizeof(struct Node));

    if (t==NULL)

    {

        printf("stack is full");

    }

    else

    {

        t->data = x;

        t->next = top;

        top = t;

    }

}

char pop()

{

    struct Node \*t;

    char x=-1;

    if (top==NULL)

    {

        printf("Stack is empty");

    }

    else

    {

        t = top;

        top = t->next;

        x=t->data;

        free(t);

    }

    return x;

}

void display()

{

    struct Node \*p;

    while (top!=NULL)

    {

        printf("%d ",p->data);

        top = p->next;

    }

    printf("\n");

}

int isBalanced(char \*e)

{

    int i;

    for(i=0;e[i]!='\0';i++)

    {

        if (e[i]=='(')

        {

            push(e[i]);

        }

        else if(e[i]==')')

        {

            if (top==NULL)

            {

                return 0;

            }

            pop();

        }

    }

    if (top==NULL)

    {

        return 1;

    }

    else

    {

        return 0;

    }

}

int main()

{

    int x;

    char \*e, exp[20];

    printf("Enter brackets: ");

    scanf("%s",&exp);

    e=exp;

    x = isBalanced(e);

    printf("%d",x);

    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;

};

struct node \*newNode(int item) {

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

  temp->key = item;

  temp->left = temp->right = NULL;

  return temp;

}

void inorder(struct node \*root) {

  if (root != NULL) {

    inorder(root->left);

    printf("%d -> ", root->key);

    inorder(root->right);

  }

}

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

  if (node == NULL) return newNode(key);

  if (key < node->key)

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

  else

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

  return node;

}

struct node \*minValueNode(struct node \*node) {

  struct node \*current = node;

  while (current && current->left != NULL)

    current = current->left;

  return current;

}

struct node \*deleteNode(struct node \*root, int key) {

  if (root == NULL) return root;

  if (key < root->key)

    root->left = deleteNode(root->left, key);

  else if (key > root->key)

    root->right = deleteNode(root->right, key);

  else {

    if (root->left == NULL) {

      struct node \*temp = root->right;

      free(root);

      return temp;

    } else if (root->right == NULL) {

      struct node \*temp = root->left;

      free(root);

      return temp;

    }

    struct node \*temp = minValueNode(root->right);

    root->key = temp->key;

    root->right = deleteNode(root->right, temp->key);

  }

  return root;

}

int main() {

  struct node \*root = NULL;

  root = insert(root, 8);

  root = insert(root, 3);

  root = insert(root, 1);

  root = insert(root, 6);

  root = insert(root, 7);

  root = insert(root, 10);

  root = insert(root, 14);

  root = insert(root, 4);

  printf("Inorder traversal: ");

  inorder(root);

  printf("\nAfter deleting 10\n");

  root = deleteNode(root, 10);

  printf("Inorder traversal: ");

  inorder(root);

}