*Stack using linked List:*

#include <bits/stdc++.h>

using namespace std;

struct Node { int data;

struct Node\* link;

};

struct Node\* top;

void push(int data)

{ struct Node\* temp;

temp = new Node();

if (!temp) {

cout << "\nHeap Overflow";

exit(1); }

temp->data = data;

temp->link = top;

top = temp; }

int isEmpty()

{ return top == NULL; }

int peek()

{ if (!isEmpty())

return top->data;

else

exit(1); }

void pop()

{ struct Node\* temp;

if (top == NULL) {

cout << "\nStack Underflow" << endl;

exit(1); }

else { temp = top;

top = top->link;

temp->link = NULL;

free(temp); } }

void display()

{ struct Node\* temp;

if (top == NULL) {

cout << "\nStack Underflow";

exit(1); }

else {

temp = top;

while (temp != NULL) {

cout << temp->data << " ";

temp = temp->link; }}}

int main()

{ push(11); push(22); push(33); push(44);

display();

cout << "\nTop element is %d\n" << peek();

pop(); pop();

display();

cout << "\nTop element is %d\n" << peek();

return 0;}

*Queue using linked list:*

#include <stdio.h>

#include <stdlib.h>

struct QNode {

    int key;

    struct QNode\* next;

};

struct Queue {

    struct QNode \*front, \*rear;

};

struct QNode\* newNode(int k)

{

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

    temp->key = k;

    temp->next = NULL;

    return temp;

}

struct Queue\* createQueue()

{

    struct Queue\* q = (struct Queue\*)malloc(sizeof(struct Queue));

    q->front = q->rear = NULL;

    return q;

}

void enQueue(struct Queue\* q, int k)

{

    // Create a new LL node

    struct QNode\* temp = newNode(k);

    if (q->rear == NULL) {

        q->front = q->rear = temp;

        return;

    }

    q->rear->next = temp;

    q->rear = temp;

}

void deQueue(struct Queue\* q)

{

    // If queue is empty, return NULL.

    if (q->front == NULL)

        return;

    struct QNode\* temp = q->front;

    q->front = q->front->next;

    // If front becomes NULL, then change rear also as NULL

    if (q->front == NULL)

        q->rear = NULL;

    free(temp);

}

int main()

{

    struct Queue\* q = createQueue();

    enQueue(q, 10);

    enQueue(q, 20);

    deQueue(q);

    deQueue(q);

    enQueue(q, 30);

    enQueue(q, 40);

    enQueue(q, 50);

    deQueue(q);

    printf("Queue Front : %d \n", q->front->key);

    printf("Queue Rear : %d", q->rear->key);

    return 0;

}

*Binary Tree:*

1.Height function

#include<bits/stdc++.h>

using namespace std;

struct Node

{

int data;

struct Node\* left;

struct Node\* right;

Node (int data)

{

this->data = data;

left = right = NULL;

}

};

void inOrder(struct Node \*root)

{

stack<Node \*> s;

Node \*curr = root;

while (curr != NULL || s.empty() == false)

{

while (curr != NULL)

{

s.push(curr);

curr = curr->left;

}

curr = s.top();

s.pop();

cout << curr->data << " ";

curr = curr->right;

}

}

int height(Node\* root)

{

if (root == NULL)

return 0;

return 1 + max(height(root->left), height(root->right));

}

int main()

{

struct Node \*root = new Node(1);

root->left = new Node(2);

root->right = new Node(3);

root->left->left = new Node(4);

root->left->right = new Node(5);

inOrder(root);

cout<<"\nThe height of the tree is :"<<height(root)<<endl;

return 0;

}

2.List of Leaf Nodes function:

void printLeafNodes(Node \*root)

{

    if (!root)

        return;

    if (!root->left && !root->right)

    {

        cout << root->data << " ";

        return;

    }

    if (root->left)

       printLeafNodes(root->left);

    if (root->right)

       printLeafNodes(root->right);

}

3. Degree of each node:

int getLevelUtil(struct node \*node, int data, int level)

{

    if (node == NULL)

        return 0;

    if (node->data == data)

        return level;

    int downlevel = getLevelUtil(node->left, data, level+1);

    if (downlevel != 0)

        return downlevel;

    downlevel = getLevelUtil(node->right, data, level+1);

    return downlevel;

}

int getLevel(struct node \*node, int data)

{

    return getLevelUtil(node,data,1);

}

4.Ancestors to a given node:

bool printAncestors(struct node \*root, int target)

{

  /\* base cases \*/

  if (root == NULL)

     return false;

  if (root->data == target)

     return true;

  if ( printAncestors(root->left, target) ||

       printAncestors(root->right, target) )

  {

    cout << root->data << " ";

    return true;

  }

  return false;

}

5.Count full nodes in binary tree:

unsigned int getfullCount(struct Node\* root)

{

if (root == NULL)

return 0;

int res = 0;

if (root->left && root->right)

res++;

res += (getfullCount(root->left) +

getfullCount(root->right));

return res;

}