Queue:

Using linked list:

#include<iostream>

using namespace std;

struct node

{

int data;

node\* next;

};

struct node\* rear = NULL;

struct node\* front = NULL;

int values;

bool isfull()

{

if (values == 5)

{

return true;

}

else

{

return false;

}

}

void Enqueue(int data1)

{

node\* newnode = new node;

newnode->data = data1;

if (isfull())

{

cout << "Queue is full" << endl;

}

else

{

if (rear == NULL)

{

rear = newnode;

front = newnode;

values++;

}

else

{

rear->next = newnode;

rear = newnode;

values++;

}

}

}

bool isempty()

{

if (rear == NULL)

{

return true;

}

else

{

return false;

}

}

void Dequeue()

{

node\* temp = front;

if (isempty())

{

cout << "Queue is empty" << endl;

}

else

{

if (front == rear)

{

front = NULL;

rear = NULL;

delete temp;

temp = NULL;

values--;

}

else

{

front = front->next;

delete temp;

temp = NULL;

values--;

}

}

}

void display()

{

node\* temp = front;

if (isempty())

{

cout << "Queue is empty" << endl;

}

else

{

while (temp != rear->next)

{

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

temp = temp->next;

}

cout << endl;

}

}

int main()

{

int choice;

int val;

cout << "1. push" << endl;

cout << "2. pop" << endl;

cout << "3. display" << endl;

cout << "4. Exit" << endl << endl;

do

{

cout << "Enter choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert : ";

cin >> val;

Enqueue(val);

break;

case 2:

Dequeue();

break;

case 3:

display();

break;

case 4:

break;

default:

cout << "Error!" << endl;

break;

}

cout << endl;

} while (choice != 4);

}

Text

Description automatically generated Text

Description automatically generated

Using array:

#include<iostream>

using namespace std;

int N = 5;

int queue[5];

int rear = -1;

int front = -1;

bool isfull()

{

if (rear == N- 1)

{

return true;

}

else

{

return false;

}

}

void Enqueue(int data)

{

if (isfull())

{

cout << "Queue is full" << endl;

}

else

{

if (rear == -1)

{

rear++;

front++;

queue[rear] = data;

}

else

{

rear++;

queue[rear] = data;

}

}

}

bool isempty()

{

if (rear == -1)

{

return true;

}

else

{

return false;

}

}

void Dequeue()

{

if (isempty())

{

cout << "Queue is empty" << endl;

}

else

{

if (front == rear)

{

front = -1;

rear = -1;

}

else

{

front++;

}

}

}

void display()

{

if (isempty())

{

cout << "Queue is empty" << endl;

}

else

{

for (int i = front; i <= rear; i++)

{

cout << queue[i] << " ";

}

cout << endl;

}

}

int main()

{

int choice;

int val;

cout << "1. push" << endl;

cout << "2. pop" << endl;

cout << "3. display" << endl;

cout << "4. Exit" << endl << endl;

do

{

cout << "Enter choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert : ";

cin >> val;

Enqueue(val);

break;

case 2:

Dequeue();

break;

case 3:

display();

break;

case 4:

break;

default:

cout << "Error!" << endl;

break;

}

cout << endl;

} while (choice != 4);

}

Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated

Circular Queue:

Using Linked list:

#include<iostream>

using namespace std;

struct node

{

int data;

node\* next;

};

struct node\* front = NULL;

struct node\* rear = NULL;

int values;

int s = 5;

bool isfull()

{

if (values == s)

{

return true;

}

else

{

return false;

}

}

void Enqueue(int data1)

{

node\* newnode = new node;

newnode->data = data1;

newnode->next = NULL;

if (isfull())

{

cout << "Queue is full" << endl;

}

else

{

if (rear == NULL)

{

rear = newnode;

front = newnode;

values++;

}

else

{

rear->next = newnode;

rear = newnode;

values++;

}

rear->next = front;

}

}

bool isempty()

{

if (rear == NULL)

{

return true;

}

else

{

return false;

}

}

void Dequeue()

{

node\* temp = front;

if (isempty())

{

cout << "Queue is empty" << endl;

}

else

{

if (rear == front)

{

rear = NULL;

front = NULL;

delete temp;

temp = NULL;

values--;

}

else

{

front = front->next;

rear->next = front;

delete temp;

temp = NULL;

values--;

}

}

}

void display()

{

node\* temp = front;

do

{

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

temp = temp->next;

} while (temp != rear->next);

if(temp == front)

{

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

}

cout << endl;

}

int main()

{

int choice;

int val;

cout << "1. push" << endl;

cout << "2. pop" << endl;

cout << "3. display" << endl;

cout << "4. Exit" << endl << endl;

do

{

cout << "Enter choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert : ";

cin >> val;

Enqueue(val);

break;

case 2:

Dequeue();

break;

case 3:

display();

break;

case 4:

break;

default:

cout << "Error!" << endl;

break;

}

cout << endl;

} while (choice != 4);

}

Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated

Using array:

#include<iostream>

using namespace std;

int S = 5;

int queue[5];

int rear = -1;

int front = -1;

int values;

bool isfull()

{

if (values == S)

{

return true;

}

else

{

return false;

}

}

void Enqueue(int data1)

{

if (isfull())

{

cout << "Queue is full" << endl;

}

else

{

if (rear == -1)

{

front = 0;

rear = 0;

queue[rear] = data1;

values++;

}

else

{

rear = (rear + 1) % S;

queue[rear] = data1;

values++;

}

}

}

bool isempty()

{

if (rear == -1)

{

return true;

}

else

{

return false;

}

}

void Dequeue()

{

if (isempty())

{

cout << "Queue is empty" << endl;

}

else

{

if (rear == front)

{

rear = -1;

front = -1;

values--;

}

else

{

front = (front + 1) % S;

values--;

}

}

}

void display()

{

if (isempty())

{

cout << "Queue is empty" << endl;

}

else

{

int i = front;

while (i != rear)

{

cout << queue[i] << " ";

i = (i + 1) % S;

}

cout << queue[rear] << " ";

cout << endl;

}

}

int main()

{

int choice;

int val;

cout << "1. push" << endl;

cout << "2. pop" << endl;

cout << "3. display" << endl;

cout << "4. Exit" << endl << endl;

do

{

cout << "Enter choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert : ";

cin >> val;

Enqueue(val);

break;

case 2:

Dequeue();

break;

case 3:

display();

break;

case 4:

break;

default:

cout << "Error!" << endl;

break;

}

cout << endl;

} while (choice != 4);

}

Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated

Trees:

Diagram, shape

Description automatically generated

1. In these tree A is parent of B & H.
2. The children of H, are I & M.
3. The siblings of J, are K & L.
4. The node who has no child are known as lead nodes like M, L, K, J etc.
5. And the other nodes are internal nodes.
6. The level of tree is start from 0/1. The level of E is 2 & level of G is 3.
7. The height of tree is start from 0. The height of these tree is 3.
8. The ancestors of the node E are B & A.
9. The descendant of the node H are I, M, J, K, & L.

Binary tree:

Chart, line chart

Description automatically generated

1. The tree whose children are 0,1 or 2 are known as binary tree.
2. The node who carries 2 children are known as Full node like Black one.
3. The node who have 1 child are known as neither like blue one.
4. The node who have 0 child are known as leaf node like green one.

Full binary tree:

Chart, line chart

Description automatically generated

1. The tree whose children must be 0 or 2 are known as Full binary tree as shown in above figure.

Perfect binary tree:

Shape

Description automatically generated

1. The tree in which every node have 2 children except leaf node & all leaf node should be placed in same level or same height are known as perfect binary tree as shown in above figure.

Formulas:

1. Number of nodes in trees = 2h+1 -1

Example: (h is height)

Shape

Description automatically generated = 23+1-1 = 16-1 = 15

1. Number of leaf nodes = 2h, Example: 23 = 8
2. Number of internal nodes = 2h-1, Example: 23-1 = 8-1 = 7
3. A perfect binary tree with n nodes has height log2(n + 1) – 1

Complete Binary tree:

1. The tree in which every node has 2 children except leaf node. The height of each leaf node should be same as h-1. Example: if a tree of height 3. If there is a leaf node is present at height of 3 and 2 then First condition is true.
2. If there is a node present on left side than it left node has al least one child there is a node it must attach on its right side of node.

A picture containing scissors, chain

Description automatically generated



1. The tree on left side, is a complete binary tree because leaf nodes are present on h or h-1. And it also full fill the second condition.
2. The tree on right side is not a complete binary tree because it can full fill the first condition but. The part which was shaded have a left side node. And the shaded node have children but as condition ask the children must be present on left side node. So, it does not full fill the second condition.

Shape

Description automatically generated

Insertion, searching, Height, Number of nodes & leaf nodes:

#include<iostream>

using namespace std;

struct node

{

int data;

node\* left;

node\* right;

};

struct node\* root = NULL;

int numberofnodes = 1;

void insert(int data1)

{

node\* newnode = new node;

newnode->data = data1;

newnode->left = NULL;

newnode->right = NULL;

node\* temp = root;

if(root == NULL)

{

root = newnode;

}

else

{

while (temp != NULL)

{

if (temp->data < data1)

{

if (temp->right == NULL)

{

temp->right = newnode;

break;

}

temp = temp->right;

}

else if (temp->data > data1)

{

if (temp->left == NULL)

{

temp->left = newnode;

break;

}

temp = temp->left;

}

else if (temp->data == data1)

{

cout << "Duplicate value is not allowed" << endl;

}

}

}

}

int search(node\* temp, int data1)

{

if (temp == NULL)

{

cout << "Value not found." << endl;

}

else if (temp->data == data1)

{

cout << "Value is found." << endl;

}

else if (temp->data > data1)

{

return search(temp->left, data1);

}

else if (temp->data < data1)

{

return search(temp->right, data1);

}

}

int height(node\* temp)

{

if (temp == NULL)

{

return -1;

}

else

{

int leftdepth=height(temp->left);

int rightdepth = height(temp->right);

if (leftdepth > rightdepth)

{

return leftdepth + 1;

}

else

{

return rightdepth + 1;

}

}

}

int numberofnode(node\* temp)

{

if(temp == NULL)

{

return 0;

}

if (temp->left != NULL)

{

numberofnodes = numberofnodes + 1;

numberofnode(temp->left);

}

if(temp->right != NULL)

{

numberofnodes = numberofnodes + 1;

numberofnode(temp->right);

}

return numberofnodes;

}

void display(node\* temp)

{

if (temp != NULL)

{

display(temp->left);

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

display(temp->right);

}

}

int leafnodes(node\* temp)

{

int i = 0;

if (temp == NULL)

{

return 0;

}

else if (temp->left == NULL && temp->right == NULL)

{

return 1;

}

else

{

return leafnodes(temp->left) + leafnodes(temp->right);

}

}

int main()

{

int choice, val1, val2;

cout << "1. Insert nodes." << endl;

cout << "2. Search nodes." << endl;

cout << "3. Display." << endl;

cout << "4. Height of tree." << endl;

cout << "5. Number of nodes." << endl;

cout << "6. Number of leaf nodes." << endl;

cout << "7. Exit." << endl;

do

{

cout << endl;

cout << "Enter choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert : ";

cin >> val1;

insert(val1);

break;

case 2:

cout << "Enter value to insert : ";

cin >> val2;

search(root,val2);

cout << endl;

break;

case 3:

display(root);

cout << endl;

break;

case 4:

cout << height(root);

cout << endl;

break;

case 5:

cout << numberofnode(root);

cout << endl;

break;

case 6:

cout << leafnodes(root);

cout << endl;

break;

case 7:

cout << endl << "EXIT" << endl;

break;

}

} while (choice != 7);

return 0;

}

Text

Description automatically generated Text

Description automatically generated

Depth-First tree traverse:

#include<iostream>

using namespace std;

struct node

{

int data;

node\* left;

node\* right;

};

struct node\* root = NULL;

int numberofnodes = 1;

void insert(int data1)

{

node\* newnode = new node;

newnode->data = data1;

newnode->left = NULL;

newnode->right = NULL;

node\* temp = root;

if(root == NULL)

{

root = newnode;

}

else

{

while (temp != NULL)

{

if (temp->data < data1)

{

if (temp->right == NULL)

{

temp->right = newnode;

break;

}

temp = temp->right;

}

else if (temp->data > data1)

{

if (temp->left == NULL)

{

temp->left = newnode;

break;

}

temp = temp->left;

}

else if (temp->data == data1)

{

cout << "Duplicate value is not allowed" << endl;

break;

}

}

}

}

void inorder(node\* temp)

{

if (temp != NULL)

{

inorder(temp->left);

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

inorder(temp->right);

}

}

void preorder(node\* temp)

{

if (temp != NULL)

{

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

inorder(temp->left);

inorder(temp->right);

}

}

void postorder(node\* temp)

{

if (temp != NULL)

{

inorder(temp->left);

inorder(temp->right);

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

}

}

int main()

{

int choice, val1, val2;

cout << "1. Insert nodes." << endl;

cout << "2. Inorder traversal" << endl;

cout << "3. preorder traversal" << endl;

cout << "4. postorder traversal" << endl;

cout << "5. Exit." << endl;

do

{

cout << endl;

cout << "Enter choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert : ";

cin >> val1;

insert(val1);

break;

case 2:

inorder(root);

cout << endl;

break;

case 3:

preorder(root);

cout << endl;

break;

case 4:

postorder(root);

cout << endl;

break;

case 5:

cout << endl << "EXIT" << endl;

break;

}

} while (choice != 5);

return 0;

}

Text

Description automatically generated Text

Description automatically generated

In order traversal:

A picture containing diagram

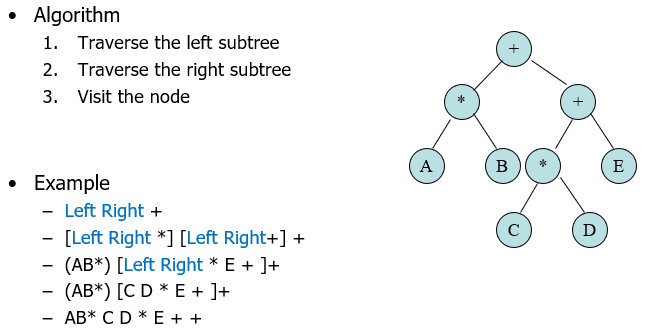
Description automatically generated

Preorder traversal:

A picture containing diagram

Description automatically generated

Post order traversal:



Deletion of node in tree:

#include<iostream>

using namespace std;

struct node

{

int data;

node\* left;

node\* right;

};

struct node\* root = NULL;

int numberofnodes = 1;

void insert(int data1)

{

node\* newnode = new node;

newnode->data = data1;

newnode->left = NULL;

newnode->right = NULL;

node\* temp = root;

if(root == NULL)

{

root = newnode;

}

else

{

while (temp != NULL)

{

if (temp->data < data1)

{

if (temp->right == NULL)

{

temp->right = newnode;

break;

}

temp = temp->right;

}

else if (temp->data > data1)

{

if (temp->left == NULL)

{

temp->left = newnode;

break;

}

temp = temp->left;

}

else if (temp->data == data1)

{

cout << "Duplicate value is not allowed" << endl;

break;

}

}

}

}

void makedeletion(node\*& temp)

{

node\* temp1;

if (temp->right == NULL)

{

temp1 = temp;

temp = temp->left;

delete temp1;

}

else if (temp->left == NULL)

{

temp1 = temp;

temp = temp->right;

delete temp1;

}

else

{

temp1 = temp->right;

while (temp1->left)

{

temp1 = temp1->left;

}

temp1->left = temp->left;

temp1 = temp;

temp = temp->right;

delete temp1;

}

}

void deletenode(node\*& temp, int val)

{

if (temp == NULL)

{

cout << "Value not found" << endl;

}

else if (temp->data < val)

{

deletenode(temp->right, val);

}

else if (temp->data > val)

{

deletenode(temp->left, val);

}

else

{

makedeletion(temp);

}

}

void inorder(node\* temp)

{

if (temp != NULL)

{

inorder(temp->left);

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

inorder(temp->right);

}

}

int main()

{

int choice, val1, val2;

cout << "1. Insert nodes." << endl;

cout << "2. Inorder traversal" << endl;

cout << "3. Delete a node" << endl;

cout << "4. Exit." << endl;

do

{

cout << endl;

cout << "Enter choice : ";

cin >> choice;

switch (choice)

{

case 1:

cout << "Enter value to insert : ";

cin >> val1;

insert(val1);

break;

case 2:

inorder(root);

cout << endl;

break;

case 3:

cout << "Enter value to insert : ";

cin >> val2;

deletenode(root, val2);

cout << endl;

break;

case 4:

cout << endl << "EXIT" << endl;

break;

}

} while (choice != 4);

return 0;

}

Text

Description automatically generated Text

Description automatically generated Text

Description automatically generated

AVL tree:

A picture containing text, whiteboard

Description automatically generated

LL:

A picture containing text, whiteboard

Description automatically generated

RR:

A picture containing text, whiteboard

Description automatically generated

RL:

A picture containing text, whiteboard

Description automatically generated

LR:

A picture containing text, whiteboard

Description automatically generated

https://www.youtube.com/watch?v=LXdi\_4kSd1o