|  |  |
| --- | --- |
| A picture of a winding road and trees  Project data structures and algorithms | Abstract  Concepts of several data structures including linked lists, AVL trees, Hash and heap are implemented to test the efficiency and time complexities of each code type. Team Members:  * Hamid Muzaffar Khan * Laaraib Ahmed * Raja Ehsan Riaz Janjua |

Contents

[Team Members: 0](file:///C:\Users\DELL\Desktop\New%20Microsoft%20Word%20Document.docx#_Toc93324705)

[AVL TREES: 1](#_Toc93324706)

[Hash Tables: 44](#_Toc93324707)

[Linked Lists: 80](#_Toc93324708)

[More Functions on AVLS: 117](#_Toc93324709)

[Heap: 159](#_Toc93324710)

NOTE: The codes are implemented on a short data set(only tuples are reduced, columns are not changed) for faster processing. Big data set is read using hash only as its time complexity is least of all and it still took around 15 minutes to process.

**Source Codes provided are run on visual studio c++ compiler.**

# **AVL TREES:**

#include<iostream>

#include<string>

#include<fstream>

#include<sstream>

using namespace std;

class actor\_movie\_node;

class movie\_actor\_node;

//-----------------Names-------------------avl

class Name\_Data

{

public:

    string name\_id;

    string name;

    string birth\_name;

    int height;

    string bio;

    string birth\_details;

    string date\_of\_birth;

    string place\_of\_birth;

    string death\_details;

    string date\_of\_death;

    string place\_of\_death;

    string reason\_of\_death;

    string spouses\_string;

    int spouses;

    int divorces;

    int spouses\_with\_children;

    int children;

    actor\_movie\_node\* moviestart;

};

class NameNode

{

public:

    Name\_Data data;

    NameNode\* lchild;

    NameNode\* rchild;

    int height1;

    NameNode(actor\_movie\_node\* moviestart, string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children);

};

NameNode::NameNode(actor\_movie\_node\* moviestart, string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children)

{

    data.moviestart = moviestart;

    data.name\_id = name\_id;

    data.name = name;

    data.birth\_name = birth\_name;

    data.bio = bio;

    data.birth\_details = birth\_details;

    data.death\_details = death\_details;

    data.place\_of\_birth = place\_of\_birth;

    data.place\_of\_death = place\_of\_death;

    data.reason\_of\_death = reason\_of\_death;

    data.spouses\_string = spouses\_string;

    int temp;

    stringstream geek(height);

    geek >> temp;

    data.height = temp;

    stringstream geek1(spouses);

    geek1 >> temp;

    data.spouses = temp;

    stringstream geek2(divorces);

    geek2 >> temp;

    data.divorces = temp;

    stringstream geek3(spouses\_with\_children);

    geek3 >> temp;

    data.spouses\_with\_children = temp;

    stringstream geek4(children);

    geek4 >> temp;

    data.children = temp;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class Name\_AVL

{

public:

    NameNode\* root;

    Name\_AVL()

    {

        root = NULL;

    }

    NameNode\* create\_node(string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children)

    {

        NameNode\* newnode = new NameNode(NULL, name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(NameNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(NameNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    NameNode\* rightrotate(NameNode\* y)

    {

        NameNode\* x = y->lchild;

        NameNode\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    NameNode\* leftrotate(NameNode\* x)

    {

        NameNode\* y = x->rchild;

        NameNode\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    NameNode\* insert(NameNode\* n, string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children)

    {

        if (n == NULL)

        {

            return create\_node(name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

        }

        if (name\_id < n->data.name\_id)

        {

            n->lchild = insert(n->lchild, name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

        }

        else if (name\_id > n->data.name\_id)

        {

            n->rchild = insert(n->rchild, name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && name\_id < n->lchild->data.name\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && name\_id > n->rchild->data.name\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && name\_id > n->lchild->data.name\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && name\_id < n->rchild->data.name\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(NameNode\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //print node

        cout << temp->data.spouses << endl;

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

//-----------------Movies-------------------

class MovieData {

public:

    string title\_id;

    string title;

    string original\_title;

    int year;

    string date;

    string genre;

    int duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    movie\_actor\_node\* actor\_Start;

    string description;

    double age\_vote;

    int votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    int meta\_score;

    int reviews\_from\_user;

    int reviews\_from\_critics;

};

class MovieNode {

public:

    MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, movie\_actor\_node\* actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics);

    MovieData data;

    MovieNode\* lchild;

    MovieNode\* rchild;

    int height1;

};

MovieNode::MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, movie\_actor\_node\* actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics) {

    int temp;

    data.title\_id = title\_id;

    data.title = title;

    data.original\_title = original\_title;

    stringstream geek(year);

    geek >> temp;

    data.year = temp;

    data.date = date;

    data.genre = genre;

    stringstream geek1(duration);

    geek >> temp;

    data.duration = temp;

    data.country = country;

    data.language = language;

    data.director = director;

    data.writer = writer;

    data.production\_company = production\_company;

    data.actor\_Start = actor\_start;

    data.description = description;

    stringstream geek2(age\_vote);

    geek >> temp;

    data.age\_vote = double(temp);

    stringstream geek6(age\_vote);

    geek >> temp;

    data.votes = temp;

    data.budget = budget;

    data.usa\_gross\_income = usa\_gross\_income;

    stringstream geek3(meta\_score);

    geek >> temp;

    data.meta\_score = temp;

    data.worldwide\_gross\_income = worldwide\_gross\_income;

    stringstream geek4(reviews\_from\_user);

    geek >> temp;

    data.reviews\_from\_user = temp;

    stringstream geek5(reviews\_from\_critics);

    geek >> temp;

    data.reviews\_from\_critics = temp;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class Movie\_AVL

{

public:

    MovieNode\* root;

    Movie\_AVL()

    {

        root = NULL;

    }

    MovieNode\* create\_node(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, movie\_actor\_node\* actor\_Start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        MovieNode\* newnode = new MovieNode(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_Start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(MovieNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(MovieNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    MovieNode\* rightrotate(MovieNode\* y)

    {

        MovieNode\* x = y->lchild;

        MovieNode\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    MovieNode\* leftrotate(MovieNode\* x)

    {

        MovieNode\* y = x->rchild;

        MovieNode\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    MovieNode\* insert(MovieNode\* n, string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, movie\_actor\_node\* actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        if (n == NULL)

        {

            return create\_node(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        if (title\_id < n->data.title\_id)

        {

            n->lchild = insert(n->lchild, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else if (title\_id > n->data.title\_id)

        {

            n->rchild = insert(n->rchild, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && title\_id < n->lchild->data.title\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && title\_id > n->rchild->data.title\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && title\_id > n->lchild->data.title\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && title\_id < n->rchild->data.title\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(MovieNode\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //print node

        cout << temp->data.genre << endl;

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

//-----------------ratings-------------------------

class Ratings\_data

{

public:

    string imdb\_title\_id;

    double weighted\_average\_vote;

    int total\_votes;

    double mean\_vote;

    int median\_vote;

    int votes\_10;

    int votes\_9;

    int votes\_8;

    int votes\_7;

    int votes\_6;

    int votes\_5;

    int votes\_4;

    int votes\_3;

    int votes\_2;

    int votes\_1;

    double allgenders\_0age\_avg\_vote;

    int allgenders\_0age\_votes;

    double allgenders\_18age\_avg\_vote;

    int allgenders\_18age\_votes;

    double allgenders\_30age\_avg\_vote;

    int allgenders\_30age\_votes;

    double allgenders\_45age\_avg\_vote;

    int allgenders\_45age\_votes;

    double males\_allages\_avg\_vote;

    int males\_allages\_votes;

    double males\_0age\_avg\_vote;

    int males\_0age\_votes;

    double males\_18age\_avg\_vote;

    int males\_18age\_votes;

    double males\_30age\_avg\_vote;

    int males\_30age\_votes;

    double males\_45age\_avg\_vote;

    int males\_45age\_votes;

    double females\_allages\_avg\_vote;

    int females\_allages\_votes;

    double females\_0age\_avg\_vote;

    int females\_0age\_votes;

    double females\_18age\_avg\_vote;

    int females\_18age\_votes;

    double females\_30age\_avg\_vote;

    int females\_30age\_votes;

    double females\_45age\_avg\_vote;

    int females\_45age\_votes;

    double top1000\_voters\_rating;

    int top1000\_voters\_votes;

    double us\_voters\_rating;

    int us\_voters\_votes;

    double non\_us\_voters\_rating;

    int non\_us\_voters\_votes;

};

class RatingsNode {

public:

    RatingsNode(string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string);

    Ratings\_data data;

    RatingsNode\* lchild;

    RatingsNode\* rchild;

    int height1;

};

RatingsNode::RatingsNode(string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes) {

    int temp;

    stringstream geek(allgenders\_0age\_avg\_vote);

    geek >> temp;

    data.allgenders\_0age\_avg\_vote = temp;

    stringstream geek1(allgenders\_0age\_votes);

    geek1 >> temp;

    data.allgenders\_0age\_votes = temp;

    stringstream geek2(allgenders\_18age\_avg\_vote);

    geek2 >> temp;

    data.allgenders\_18age\_avg\_vote = temp;

    stringstream geek3(allgenders\_18age\_votes);

    geek3 >> temp;

    data.allgenders\_18age\_votes = temp;

    stringstream geek4(allgenders\_30age\_avg\_vote);

    geek4 >> temp;

    data.allgenders\_30age\_avg\_vote = temp;

    stringstream geek5(allgenders\_30age\_votes);

    geek5 >> temp;

    data.allgenders\_30age\_votes = temp;

    stringstream geek6(allgenders\_45age\_avg\_vote);

    geek6 >> temp;

    data.allgenders\_45age\_avg\_vote = temp;

    stringstream geek7(allgenders\_45age\_votes);

    geek7 >> temp;

    data.allgenders\_45age\_votes = temp;

    stringstream geek8(females\_0age\_avg\_vote);

    geek8 >> temp;

    data.females\_0age\_avg\_vote = temp;

    stringstream geek9(females\_0age\_votes);

    geek9 >> temp;

    data.females\_0age\_votes = temp;

    stringstream geek10(females\_0age\_avg\_vote);

    geek10 >> temp;

    data.females\_18age\_avg\_vote = temp;

    stringstream geek11(females\_18age\_votes);

    geek11 >> temp;

    data.females\_18age\_votes = temp;

    stringstream geek12(females\_30age\_avg\_vote);

    geek12 >> temp;

    data.females\_30age\_avg\_vote = temp;

    stringstream geek13(females\_30age\_votes);

    geek13 >> temp;

    data.females\_30age\_votes = temp;

    stringstream geek14(females\_45age\_avg\_vote);

    geek14 >> temp;

    data.females\_45age\_avg\_vote = temp;

    stringstream geek15(females\_45age\_votes);

    geek15 >> temp;

    data.females\_45age\_votes = temp;

    stringstream geek16(females\_allages\_avg\_vote);

    geek16 >> temp;

    data.females\_allages\_avg\_vote = temp;

    stringstream geek17(females\_allages\_votes);

    geek17 >> temp;

    data.females\_allages\_votes = temp;

    stringstream geek18(males\_0age\_avg\_vote);

    geek18 >> temp;

    data.males\_0age\_avg\_vote = temp;

    stringstream geek19(males\_0age\_votes);

    geek19 >> temp;

    data.males\_0age\_votes = temp;

    stringstream geek20(males\_0age\_avg\_vote);

    geek20 >> temp;

    data.males\_18age\_avg\_vote = temp;

    stringstream geek21(males\_18age\_votes);

    geek21 >> temp;

    data.males\_18age\_votes = temp;

    stringstream geek22(males\_30age\_avg\_vote);

    geek22 >> temp;

    data.males\_30age\_avg\_vote = temp;

    stringstream geek23(males\_30age\_votes);

    geek23 >> temp;

    data.males\_30age\_votes = temp;

    stringstream geek24(males\_45age\_avg\_vote);

    geek24 >> temp;

    data.males\_45age\_avg\_vote = temp;

    stringstream geek25(males\_45age\_votes);

    geek25 >> temp;

    data.males\_45age\_votes = temp;

    stringstream geek26(males\_allages\_avg\_vote);

    geek26 >> temp;

    data.males\_allages\_avg\_vote = temp;

    stringstream geek27(males\_allages\_votes);

    geek27 >> temp;

    data.males\_allages\_votes = temp;

    stringstream geek28(imdb\_title\_id);

    geek28 >> temp;

    data.imdb\_title\_id = temp;

    stringstream geek29(mean\_vote);

    geek29 >> temp;

    data.mean\_vote = temp;

    stringstream geek30(median\_vote);

    geek30 >> temp;

    data.median\_vote = temp;

    stringstream geek31(non\_us\_voters\_rating);

    geek31 >> temp;

    data.non\_us\_voters\_rating = temp;

    stringstream geek32(non\_us\_voters\_votes);

    geek32 >> temp;

    data.non\_us\_voters\_votes = temp;

    stringstream geek33(top1000\_voters\_rating);

    geek33 >> temp;

    data.top1000\_voters\_rating = temp;

    stringstream geek34(top1000\_voters\_votes);

    geek34 >> temp;

    data.top1000\_voters\_votes = temp;

    stringstream geek35(total\_votes);

    geek35 >> temp;

    data.total\_votes = temp;

    stringstream geek36(us\_voters\_rating);

    geek36 >> temp;

    data.us\_voters\_rating = temp;

    stringstream geek37(us\_voters\_votes);

    geek37 >> temp;

    data.us\_voters\_votes = temp;

    stringstream geek38(votes\_1);

    geek38 >> temp;

    data.votes\_1 = temp;

    stringstream geek39(votes\_10);

    geek39 >> temp;

    data.votes\_10 = temp;

    stringstream geek40(votes\_2);

    geek40 >> temp;

    data.votes\_2 = temp;

    stringstream geek41(votes\_3);

    geek41 >> temp;

    data.votes\_3 = temp;

    stringstream geek42(votes\_4);

    geek42 >> temp;

    data.votes\_4 = temp;

    stringstream geek43(votes\_5);

    geek43 >> temp;

    data.votes\_5 = temp;

    stringstream geek44(votes\_6);

    geek44 >> temp;

    data.votes\_6 = temp;

    stringstream geek45(votes\_7);

    geek45 >> temp;

    data.votes\_7 = temp;

    stringstream geek46(votes\_8);

    geek46 >> temp;

    data.votes\_8 = temp;

    stringstream geek47(votes\_9);

    geek47 >> temp;

    data.votes\_9 = temp;

    stringstream geek48(weighted\_average\_vote);

    geek48 >> temp;

    data.weighted\_average\_vote = temp;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class Ratings\_AVL

{

public:

    RatingsNode\* root;

    Ratings\_AVL()

    {

        root = NULL;

    }

    RatingsNode\* create\_node(string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes)

    {

        RatingsNode\* newnode = new RatingsNode(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(RatingsNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(RatingsNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    RatingsNode\* rightrotate(RatingsNode\* y)

    {

        RatingsNode\* x = y->lchild;

        RatingsNode\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    RatingsNode\* leftrotate(RatingsNode\* x)

    {

        RatingsNode\* y = x->rchild;

        RatingsNode\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    RatingsNode\* insert(RatingsNode\* n, string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes)

    {

        if (n == NULL)

        {

            return create\_node(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

        }

        if (imdb\_title\_id < n->data.imdb\_title\_id)

        {

            n->lchild = insert(n->lchild, imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

        }

        else if (imdb\_title\_id > n->data.imdb\_title\_id)

        {

            n->rchild = insert(n->rchild, imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && imdb\_title\_id < n->lchild->data.imdb\_title\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && imdb\_title\_id > n->rchild->data.imdb\_title\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && imdb\_title\_id > n->lchild->data.imdb\_title\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && imdb\_title\_id < n->rchild->data.imdb\_title\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(RatingsNode\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //print node

        cout << temp->data.imdb\_title\_id << endl;

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

//-----------------title\_principles----------------

class title\_principles\_Data {

public:

    string imdb\_title\_id;

    int order;

    string imdb\_name\_id;

    string category;

    string job;

    string characters;

};

class title\_principles\_Node {

public:

    title\_principles\_Node(string, string, string, string, string, string);

    title\_principles\_Data data;

    title\_principles\_Node\* lchild;

    title\_principles\_Node\* rchild;

    int height1;

};

title\_principles\_Node::title\_principles\_Node(string imdb\_title\_id, string order, string imdb\_name\_id, string category, string job, string characters) {

    int temp;

    data.imdb\_title\_id = imdb\_title\_id;

    stringstream geek(order);

    geek >> temp;

    data.order = temp;

    data.imdb\_name\_id = imdb\_name\_id;

    data.category = category;

    data.job = job;

    data.characters = characters;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class title\_principles\_AVL

{

public:

    title\_principles\_Node\* root;

    title\_principles\_AVL()

    {

        root = NULL;

    }

    title\_principles\_Node\* create\_node(string imdb\_title\_id, string order, string imdb\_name\_id, string category, string job, string characters)

    {

        title\_principles\_Node\* newnode = new title\_principles\_Node(imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(title\_principles\_Node\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(title\_principles\_Node\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    title\_principles\_Node\* rightrotate(title\_principles\_Node\* y)

    {

        title\_principles\_Node\* x = y->lchild;

        title\_principles\_Node\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    title\_principles\_Node\* leftrotate(title\_principles\_Node\* x)

    {

        title\_principles\_Node\* y = x->rchild;

        title\_principles\_Node\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    title\_principles\_Node\* insert(title\_principles\_Node\* n, string imdb\_title\_id, string order, string imdb\_name\_id, string category, string job, string characters)

    {

        if (n == NULL)

        {

            return create\_node(imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

        }

        if (imdb\_name\_id < n->data.imdb\_name\_id)

        {

            n->lchild = insert(n->lchild, imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

        }

        else if (imdb\_name\_id > n->data.imdb\_name\_id)

        {

            n->rchild = insert(n->rchild, imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && imdb\_name\_id < n->lchild->data.imdb\_name\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && imdb\_name\_id > n->rchild->data.imdb\_name\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && imdb\_name\_id > n->lchild->data.imdb\_name\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && imdb\_title\_id < n->rchild->data.imdb\_title\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(title\_principles\_Node\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //print node

        cout << temp->data.category << endl;

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

//----------------------------------------------------------------------------------------------

//----------------------------------------------------------------------------------------------

class movie\_actor\_node

{

public:

    NameNode\* points\_to\_actor;

    movie\_actor\_node\* rch;

    movie\_actor\_node\* lch;

    int height;

    string key\_tits\_id;

    movie\_actor\_node()

    {

        rch = NULL;

        lch = NULL;

    }

};

class movie\_actor\_avl

{

public:

    int max(int a, int b)

    {

        return a > b ? a : b;

    }

    int height(movie\_actor\_node\* temp)

    {

        if (temp == NULL)

            return 0;

        return temp->height;

    }

    int calc\_balance\_spectrum(movie\_actor\_node\* temp)

    {

        return height(temp->rch) - height(temp->lch);

    }

    movie\_actor\_node\* create\_newnode(string val)

    {

        movie\_actor\_node\* newnode = new movie\_actor\_node();//try not passing value or building constructor for newnode

        newnode->lch = NULL;

        newnode->rch = NULL;

        newnode->height = 1;

        newnode->key\_tits\_id = val; /\*cout << newnode->key\_tits\_id;\*/

        return newnode;

    }

    movie\_actor\_node\* left\_rotation(movie\_actor\_node\* temp)

    {

        movie\_actor\_node\* y = temp;

        movie\_actor\_node\* x = temp->rch;

        y->rch = x->lch;

        x->lch = y;

        y->height = max(height(y->rch), height(y->lch)) + 1;

        x->height = max(height(x->rch), height(x->lch)) + 1;

        return x;

    }

    movie\_actor\_node\* right\_rotation(movie\_actor\_node\* temp)

    {

        movie\_actor\_node\* y = temp;

        movie\_actor\_node\* x = temp->lch;

        y->lch = x->rch;

        x->rch = y;

        y->height = max(height(y->rch), height(y->lch)) + 1;

        x->height = max(height(x->rch), height(x->lch)) + 1;

        return x;

    }

    movie\_actor\_node\* insertnode(string val, movie\_actor\_node\* temp)

    {

        if (temp == NULL) {

            /\*cout << "1\n";\*/

            return create\_newnode(val);

        }

        if (val > temp->key\_tits\_id)

            temp->rch = insertnode(val, temp->rch);

        else if (val < temp->key\_tits\_id)

            temp->lch = insertnode(val, temp->lch);

        else {

            cout << "duplicate value input is not allowed\n";

            return temp;

        }

        temp->height = max(height(temp->rch), height(temp->lch)) + 1;

        int bs = calc\_balance\_spectrum(temp);

        if (bs > 1 && val > temp->rch->key\_tits\_id)

        {

            return left\_rotation(temp);

        }

        if (bs > 1 && val < temp->rch->key\_tits\_id)

        {

            temp->rch = right\_rotation(temp->rch);

            return left\_rotation(temp);

        }

        if (bs < -1 && val < temp->lch->key\_tits\_id)

        {

            return right\_rotation(temp);

        }

        if (bs < -1 && val < temp->lch->key\_tits\_id)

        {

            temp->lch = left\_rotation(temp->lch);

            return right\_rotation(temp);

        }

        return temp;

    }

};

void find\_actor\_in\_ma\_avl(movie\_actor\_node\* root, NameNode\* name)

{

    if (root == NULL)

        return;

    if (root->key\_tits\_id == name->data.name)

        root->points\_to\_actor = name;

    find\_actor\_in\_ma\_avl(root->rch, name);

    find\_actor\_in\_ma\_avl(root->lch, name);

}

bool find\_actor\_in\_names(NameNode\* traverser, movie\_actor\_node\* mavl, string token)

{

    if (traverser == NULL) {

        //cout << "actor notfound\n";

        return false;

    }

    find\_actor\_in\_names(traverser->rchild, mavl, token);

    find\_actor\_in\_names(traverser->lchild, mavl, token);

    if (traverser->data.name == token)

    {

        find\_actor\_in\_ma\_avl(mavl, traverser);

        return true;

    }

}

//----------------------------------------------------------------------------------------------

//----------------------------------------------------------------------------------------------

class actor\_movie\_node

{

public:

    MovieNode\* points\_to\_movie;

    actor\_movie\_node\* rch;

    actor\_movie\_node\* lch;

    int height;

    string key\_name\_id;

};

class actor\_movie\_avl

{

public:

    actor\_movie\_node\* root = NULL;

    actor\_movie\_node\* loc = NULL;

    actor\_movie\_node\* ploc = NULL;

    //actor\_movie\_node\* insertnode(string, actor\_movie\_node\*);

    //void printtree(actor\_movie\_node\*);

    int max(int a, int b)

    {

        return a > b ? a : b;

    }

    int height(actor\_movie\_node\* temp)

    {

        if (temp == NULL)

            return 0;

        return temp->height;

    }

    int calc\_balance\_spectrum(actor\_movie\_node\* temp)

    {

        return height(temp->rch) - height(temp->lch);

    }

    actor\_movie\_node\* create\_newnode(string val, MovieNode\* temp)

    {

        actor\_movie\_node\* newnode = new actor\_movie\_node;

        newnode->lch = NULL;

        newnode->rch = NULL;

        newnode->height = 1;

        newnode->points\_to\_movie = temp;

        newnode->key\_name\_id = val;

        return newnode;

    }

    int count = 0;//test

    actor\_movie\_node\* left\_rotation(actor\_movie\_node\* temp)

    {

        actor\_movie\_node\* y = temp;

        actor\_movie\_node\* x = temp->rch;

        y->rch = x->lch;

        x->lch = y;

        y->height = max(height(y->rch), height(y->lch)) + 1;

        x->height = max(height(x->rch), height(x->lch)) + 1;

        return x;

    }

    actor\_movie\_node\* right\_rotation(actor\_movie\_node\* temp)

    {

        actor\_movie\_node\* y = temp;

        actor\_movie\_node\* x = temp->lch;

        y->lch = x->rch;

        x->rch = y;

        y->height = max(height(y->rch), height(y->lch)) + 1;

        x->height = max(height(x->rch), height(x->lch)) + 1;

        return x;

    }

    actor\_movie\_node\* insertnode(string val, actor\_movie\_node\* temp, MovieNode\* movie)

    {

        if (temp == NULL || temp->key\_name\_id == val) {

            //cout << count++;//test

            /\*cout << "inserted\n";\*/

            return create\_newnode(val, movie);

        }

        if (val > temp->key\_name\_id)

            temp->rch = insertnode(val, temp->rch, movie);

        else if (val < temp->key\_name\_id)

            temp->lch = insertnode(val, temp->lch, movie);

        //else {

        //

        //  cout << "duplicate value input is not allowed in actor\_movie avl\n";

        //  return temp;

        //}

        temp->height = max(height(temp->rch), height(temp->lch)) + 1;

        int bs = calc\_balance\_spectrum(temp);

        if (bs > 1 && val > temp->rch->key\_name\_id)

        {

            return left\_rotation(temp);

        }

        if (bs > 1 && val < temp->rch->key\_name\_id)

        {

            temp->rch = right\_rotation(temp->rch);

            return left\_rotation(temp);

        }

        if (bs < -1 && val < temp->lch->key\_name\_id)

        {

            return right\_rotation(temp);

        }

        if (bs < -1 && val < temp->lch->key\_name\_id)

        {

            temp->lch = left\_rotation(temp->lch);

            return right\_rotation(temp);

        }

        return temp;

    }

};

void read(Movie\_AVL\* ml, Ratings\_AVL\* rl, title\_principles\_AVL\* t1, Name\_AVL\* t2) {

    //-----------------name Storage----------------

    int count = 0;

    string name\_id;

    string name;

    string birth\_name;

    string height;

    string bio;

    string birth\_details;

    string date\_of\_birth;

    string place\_of\_birth;

    string death\_details;

    string date\_of\_death; string place\_of\_death; string reason\_of\_death; string spouses\_string; string spouses; string divorces; string spouses\_with\_children; string children;

    actor\_movie\_node\* points\_to\_movie;

    ifstream File3("names.csv");

    if (File3.is\_open()) {

        while (!File3.eof()) {

            if (count == 0) {

                string temp;

                getline(File3, temp);

                count = 1;

            }

            else {

                getline(File3, name\_id, ',');

                getline(File3, name, ',');

                getline(File3, birth\_name, ',');

                getline(File3, height, ',');

                getline(File3, bio, ',');

                getline(File3, birth\_details, ',');

                getline(File3, date\_of\_birth, ',');

                getline(File3, place\_of\_birth, ',');

                getline(File3, death\_details, ',');

                getline(File3, date\_of\_death, ',');

                getline(File3, place\_of\_death, ',');

                getline(File3, reason\_of\_death, ',');

                getline(File3, spouses\_string, ',');

                getline(File3, spouses, ',');

                getline(File3, divorces, ',');

                getline(File3, spouses\_with\_children, ',');

                getline(File3, children);

                t2->root = t2->insert(t2->root, name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

            }

        }

    }

    else {

        cout << "Give Error3";

    }

    //---------Movie List Storage------------

    count = 0;

    string title\_id;

    string title;

    string original\_title;

    string year;

    string date;

    string genre;

    string duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    string actors;

    string description;

    string age\_vote;

    string votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    string meta\_score;

    string reviews\_from\_user;

    string reviews\_from\_critics;

    ifstream File("movies.csv");

    movie\_actor\_avl m\_a\_avl;

    if (File.is\_open()) {

        while (!File.eof()) {

            if (count == 0) {

                string temp;

                getline(File, temp);

                count = 1;

            }

            else {

                getline(File, title\_id, ',');

                getline(File, title, ',');

                getline(File, original\_title, ',');

                getline(File, year, ',');

                getline(File, date, ',');

                getline(File, genre, ',');

                getline(File, duration, ',');

                getline(File, country, ',');

                getline(File, language, ',');

                getline(File, director, ',');

                getline(File, writer, ',');

                getline(File, production\_company, ',');

                getline(File, actors, ',');

                movie\_actor\_node\* actor\_start = NULL;

                string s = actors;

                string delimiter = "-";

                size\_t pos = 0;

                string token;

                NameNode\* traverser = t2->root;

                bool flag = false;

                while ((pos = s.find(delimiter)) != string::npos) {

                    token = s.substr(0, pos);

                    traverser = t2->root;

                    actor\_start = m\_a\_avl.insertnode(token, actor\_start);

                    /\*cout << "che\n";\*/

                    flag = find\_actor\_in\_names(traverser, actor\_start, token);

                    // if (flag == false)

                    //  cout << "actor not found in names file\n\n";

                    s.erase(0, pos + delimiter.length());

                }

                actor\_start = m\_a\_avl.insertnode(s, actor\_start);

                flag = find\_actor\_in\_names(traverser, actor\_start, s);

                // if (flag == false)

                //  cout << "actor not found in names file\n\n";

                getline(File, description, ',');

                getline(File, age\_vote, ',');

                getline(File, votes, ',');

                getline(File, budget, ',');

                getline(File, usa\_gross\_income, ',');

                getline(File, meta\_score, ',');

                getline(File, reviews\_from\_user, ',');

                getline(File, reviews\_from\_critics);

                ml->root = ml->insert(ml->root, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

            }

        }

    }

    else {

        cout << "Give Error\n" << endl;

    }

    //---------Rating List Storage------------

    count = 0;

    string imdb\_title\_id;

    string weighted\_average\_vote;

    string  total\_votes;

    string mean\_vote;

    string median\_vote;

    string votes\_10;

    string votes\_9;

    string votes\_8;

    string votes\_7;

    string  votes\_6;

    string  votes\_5;

    string  votes\_4;

    string  votes\_3;

    string  votes\_2;

    string  votes\_1;

    string allgenders\_0age\_avg\_vote;

    string allgenders\_0age\_votes;

    string allgenders\_18age\_avg\_vote;

    string allgenders\_18age\_votes;

    string allgenders\_30age\_avg\_vote;

    string allgenders\_30age\_votes;

    string allgenders\_45age\_avg\_vote;

    string allgenders\_45age\_votes;

    string males\_allages\_avg\_vote;

    string males\_allages\_votes;

    string males\_0age\_avg\_vote;

    string males\_0age\_votes;

    string males\_18age\_avg\_vote;

    string males\_18age\_votes;

    string males\_30age\_avg\_vote;

    string males\_30age\_votes;

    string males\_45age\_avg\_vote;

    string males\_45age\_votes;

    string females\_allages\_avg\_vote;

    string females\_allages\_votes;

    string females\_0age\_avg\_vote;

    string females\_0age\_votes;

    string females\_18age\_avg\_vote;

    string females\_18age\_votes;

    string females\_30age\_avg\_vote;

    string females\_30age\_votes;

    string females\_45age\_avg\_vote;

    string females\_45age\_votes;

    string top1000\_voters\_rating;

    string top1000\_voters\_votes;

    string us\_voters\_rating;

    string us\_voters\_votes;

    string non\_us\_voters\_rating;

    string non\_us\_voters\_votes;

    ifstream File1("Ratings.csv");

    if (File1.is\_open()) {

        while (!File1.eof()) {

            if (count == 0) {

                string temp;

                getline(File1, temp);

                count = 1;

            }

            else {

                getline(File1, imdb\_title\_id, ',');

                getline(File1, weighted\_average\_vote, ',');

                getline(File1, total\_votes, ',');

                getline(File1, mean\_vote, ',');

                getline(File1, median\_vote, ',');

                getline(File1, votes\_10, ',');

                getline(File1, votes\_9, ',');

                getline(File1, votes\_8, ',');

                getline(File1, votes\_7, ',');

                getline(File1, votes\_6, ',');

                getline(File1, votes\_5, ',');

                getline(File1, votes\_4, ',');

                getline(File1, votes\_3, ',');

                getline(File1, votes\_2, ',');

                getline(File1, votes\_1, ',');

                getline(File1, votes\_9, ',');

                getline(File1, allgenders\_0age\_avg\_vote, ',');

                getline(File1, allgenders\_0age\_votes, ',');

                getline(File1, allgenders\_18age\_avg\_vote, ',');

                getline(File1, allgenders\_18age\_votes, ',');

                getline(File1, allgenders\_18age\_votes, ',');

                getline(File1, allgenders\_30age\_avg\_vote, ',');

                getline(File1, allgenders\_30age\_avg\_vote, ',');

                getline(File1, allgenders\_45age\_avg\_vote, ',');

                getline(File1, allgenders\_45age\_avg\_vote, ',');

                getline(File1, females\_allages\_avg\_vote, ',');

                getline(File1, females\_allages\_votes, ',');

                getline(File1, females\_0age\_avg\_vote, ',');

                getline(File1, females\_0age\_votes, ',');

                getline(File1, females\_18age\_avg\_vote, ',');

                getline(File1, females\_18age\_votes, ',');

                getline(File1, females\_30age\_avg\_vote, ',');

                getline(File1, females\_30age\_votes, ',');

                getline(File1, females\_45age\_avg\_vote, ',');

                getline(File1, top1000\_voters\_rating, ',');

                getline(File1, top1000\_voters\_votes, ',');

                getline(File1, us\_voters\_rating, ',');

                getline(File1, us\_voters\_votes, ',');

                getline(File1, non\_us\_voters\_rating, ',');

                getline(File1, non\_us\_voters\_votes);

                rl->root = rl->insert(rl->root, imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

            }

        }

    }

    else {

        cout << "Give Error1\n";

    }

    //-----------------title\_principles Storage----------------

    count = 0;

    string imdb\_title\_id1;

    string order;

    string imdb\_name\_id;

    string category;

    string job;

    string characters;

    ifstream File2("title\_principals.csv");

    if (File2.is\_open()) {

        while (!File2.eof()) {

            if (count == 0) {

                string temp;

                getline(File2, temp);

                count = 1;

            }

            else {

                getline(File2, imdb\_title\_id1, ',');

                getline(File2, order, ',');

                getline(File2, imdb\_name\_id, ',');

                getline(File2, category, ',');

                getline(File2, job, ',');

                getline(File2, characters);

                t1->root = t1->insert(t1->root, imdb\_title\_id1, order, imdb\_name\_id, category, job, characters);

            }

        }

    }

    else {

        cout << "Give Error2";

    }

}

void Print\_PreOrder1(movie\_actor\_node\* temp)/\*related to movie\_actor node\*/ {

    if (temp == NULL) {

        //Base Case

        return;

    }

    //Go to the left child recursively

    Print\_PreOrder1(temp->lch);

    //print node

    cout << temp->key\_tits\_id << endl;

    //Then go to the right child recursively

    Print\_PreOrder1(temp->rch);

}

//-------------------------------------------------------------------------------------------

//-------------------------------------------------------------------------------------------

actor\_movie\_avl am;

void traverse\_Actorstart(movie\_actor\_node\* temp, MovieNode\* movie)

{

    if (temp == NULL)

        return;

    temp->points\_to\_actor->data.moviestart = am.insertnode(temp->points\_to\_actor->data.name\_id, temp->points\_to\_actor->data.moviestart, movie);

    /\*create\_avl(temp->points\_to\_actor, movie);\*///removed

    traverse\_Actorstart(temp->lch, movie);

    traverse\_Actorstart(temp->rch, movie);

}

void traverse\_movie(MovieNode\* traverser)

{

    if (traverser == NULL)

        return;

    traverse\_Actorstart(traverser->data.actor\_Start, traverser);

    //traversing movie avl

    traverse\_movie(traverser->lchild);

    traverse\_movie(traverser->rchild);

}

//------------------------------------------------------------------

//------------------------printing functions------------------------

void print\_coactors\_of\_an\_actor(string, NameNode\*, Movie\_AVL, Name\_AVL);

void traverse\_actor\_Start1(movie\_actor\_node\* temp, string name, int choice, Movie\_AVL ml, NameNode\* n1, Name\_AVL n11)

{

    if (temp == NULL)

        return;

    if (temp->points\_to\_actor->data.name != name)

    {

        print\_coactors\_of\_an\_actor(temp->points\_to\_actor->data.name, n11.root, ml, n11);

    }

    traverse\_actor\_Start1(temp->lch, name, choice, ml, n1, n11);

    traverse\_actor\_Start1(temp->rch, name, choice, ml, n1, n11);

}

void traverse\_actor\_Start(movie\_actor\_node\* temp, string name, int choice, Movie\_AVL ml, NameNode\* n1)

{

    if (temp == NULL)

        return;

    if (temp->points\_to\_actor->data.name != name)

    {

        cout << temp->points\_to\_actor->data.name << endl;

    }

    traverse\_actor\_Start(temp->lch, name, choice, ml, n1);

    traverse\_actor\_Start(temp->rch, name, choice, ml, n1);

}

void print\_preorder\_order2(MovieNode\* actor\_ki\_movie, string name, int choice, Movie\_AVL ml, NameNode\* n1, Name\_AVL n11)

{

    movie\_actor\_node\* temp = actor\_ki\_movie->data.actor\_Start;

    (choice == 0) ?

        cout << "The co-actors of actor " << name << " are\n" : cout << "The co-actors of coactors of " << name << " are\n\n";

    (choice == 0) ?

        traverse\_actor\_Start(temp, name, choice, ml, n1) : traverse\_actor\_Start1(temp, name, choice, ml, n1, n11);

}

void traverse\_moviestart(actor\_movie\_node\* actor\_ki\_movies, string name, int choice, Movie\_AVL ml, NameNode\* n1, Name\_AVL n11)

{

    /\*cout << "uffffffs\n";\*/

    if (actor\_ki\_movies == NULL)

        return;

    print\_preorder\_order2(actor\_ki\_movies->points\_to\_movie, name, choice, ml, n1, n11);

    traverse\_moviestart(actor\_ki\_movies->lch, name, choice, ml, n1, n11);

    traverse\_moviestart(actor\_ki\_movies->rch, name, choice, ml, n1, n11);

}

void print\_preorder\_order1(NameNode\* n1, string name, int choice, Movie\_AVL ml, Name\_AVL n11)

{

    actor\_movie\_node\* actor\_ki\_movies = n1->data.moviestart;

    traverse\_moviestart(actor\_ki\_movies, name, choice, ml, n1, n11);

}

void print\_coactors\_of\_an\_actor(string name, NameNode\* n1, Movie\_AVL ml, Name\_AVL n11)

{

    if (n1 == NULL) {

        return;

    }

    /\*cout << "passes\n";\*/

    if (name == n1->data.name)

    {

        print\_preorder\_order1(n1, name, 0, ml, n11);

        /\*return;\*/

    }

    print\_coactors\_of\_an\_actor(name, n1->lchild, ml, n11);

    print\_coactors\_of\_an\_actor(name, n1->rchild, ml, n11);

}

void print\_coactors\_of\_coactors\_of\_an\_actor(string name, NameNode\* n1, Movie\_AVL ml, Name\_AVL n11)

{

    if (n1 == NULL) {

        return;

    }

    /\*cout << "passes\n";\*/

    if (name == n1->data.name)

    {

        print\_preorder\_order1(n1, name, 1, ml, n11);

        /\*return;\*/// escape sequemce when name found

    }

    print\_coactors\_of\_coactors\_of\_an\_actor(name, n1->lchild, ml, n11);

    print\_coactors\_of\_coactors\_of\_an\_actor(name, n1->rchild, ml, n11);

}

void main() {

    Movie\_AVL ml;

    Ratings\_AVL rl;

    title\_principles\_AVL t1;

    Name\_AVL n1;

    read(&ml, &rl, &t1, &n1);

    MovieNode\* traverser = ml.root;

    traverse\_movie(traverser);

    print\_coactors\_of\_coactors\_of\_an\_actor("Blanche Bayliss", n1.root, ml, n1);

    getchar();

    getchar();

}

Text

Description automatically generated

# **Hash Tables:**

#include<iostream>

#include<string>

#include<string.h>

#include<fstream>

#include<sstream>

using namespace std;

const int ARR\_SIZE = 23;//hash

static const int PRIME\_CONST = 11;

//Class declaration of actor\_movie\_node and movie\_actor\_node

//to use there members before definition

class actor\_movie\_node;

class movie\_actor\_node;

//class Movie data to store data of Movie

class MovieData {

public:

    //public data members

    string title\_id;

    string title;

    string original\_title;

    int year;

    string date;

    string genre;

    int duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    string actor\_Start;

    string description;

    double age\_vote;

    int votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    int meta\_score;

    int reviews\_from\_user;

    int reviews\_from\_critics;

};

//Class for AVL iplementation

class MovieNode {

public:

    //constructor to pass values to datamembers of MovieNode for a specific node

    MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics);

    //Moviedata tyoe datta variabes to access the data in MovieNode

    MovieData data;

    //Left and Right Child for AVL

    MovieNode\* lch;

    MovieNode\* rch;

    //integer vairable of keep track of height of AVL tree

    int height1;

};

//Paraerterized Constructor for MovieNode class

MovieNode::MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics) {

    //Conversion to original datatype after reading from csv file as strings

    int temp;

    data.title\_id = title\_id;

    data.title = title;

    data.original\_title = original\_title;

    stringstream geek(year);

    geek >> temp;

    data.year = temp;

    data.date = date;

    data.genre = genre;

    stringstream geek1(duration);

    geek >> temp;

    data.duration = temp;

    data.country = country;

    data.language = language;

    data.director = director;

    data.writer = writer;

    data.production\_company = production\_company;

    data.actor\_Start = actor\_start;

    data.description = description;

    stringstream geek2(age\_vote);

    geek >> temp;

    data.age\_vote = double(temp);

    stringstream geek6(age\_vote);

    geek >> temp;

    data.votes = temp;

    data.budget = budget;

    data.usa\_gross\_income = usa\_gross\_income;

    stringstream geek3(meta\_score);

    geek >> temp;

    data.meta\_score = temp;

    data.worldwide\_gross\_income = worldwide\_gross\_income;

    stringstream geek4(reviews\_from\_user);

    geek >> temp;

    data.reviews\_from\_user = temp;

    stringstream geek5(reviews\_from\_critics);

    geek >> temp;

    data.reviews\_from\_critics = temp;

    lch = NULL;

    rch = NULL;

    height1 = 1;

}

//Class movie\_avl\_year for AVL functions implementations based on the value of Year

class movie\_avl\_year

{

public:

    //Function which returns the address of newly created node

    MovieNode\* create\_node(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_Start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        MovieNode\* newnode = new MovieNode(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_Start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        return newnode;

    }

    //Function which returns max of two values

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    //Function which returns the heigth of current node

    int get\_height(MovieNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    //Function to return the balance factor of two nodes(hieght\_of\_lefttree-height \_of\_rightsubtree)

    int getBalanceFactor(MovieNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lch) - get\_height(n->rch));

    }

    //Function which right rotates the passed node

    MovieNode\* rightrotate(MovieNode\* y)

    {

        MovieNode\* x = y->lch;

        MovieNode\* T2 = x->rch;

        x->rch = y;

        y->lch = T2;

        //Update height after Rotations

        y->height1 = max(get\_height(y->rch), get\_height(y->lch)) + 1;

        x->height1 = max(get\_height(x->rch), get\_height(x->lch)) + 1;

        //Returns the new Parent after Rotation

        return x;

    }

    //Function which left rotates the passed node

    MovieNode\* leftrotate(MovieNode\* x)

    {

        MovieNode\* y = x->rch;

        MovieNode\* T2 = y->lch;

        y->lch = x;

        x->rch = T2;

        //Update height after rotations

        x->height1 = max(get\_height(x->rch), get\_height(x->lch)) + 1;

        y->height1 = max(get\_height(y->rch), get\_height(y->lch)) + 1;

        //Returns the new Parent after Rotation

        return y;

    }

    MovieNode\* insert(MovieNode\* rn, string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        //Simple BST insertion

        if (rn == NULL)

        {

            return create\_node(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        if (rn->data.title\_id < title\_id)

        {

            rn->lch = insert(rn->lch, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else if (rn->data.title\_id > title\_id)

        {

            rn->rch = insert(rn->rch, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else//Duplication is not allowed

            return rn;

        //Update heigth After every Insertion

        rn->height1 = 1 + max(get\_height(rn->lch), get\_height(rn->rch));

        //Checking balance node of current node wheter it becomes unbalakced tree or not

        int bf = getBalanceFactor(rn);

        //if node become unbalanced then there are 4 rotations cases

        //right rotate case

        if (bf > 1 && rn->lch->data.title\_id < title\_id)

        {

            return rightrotate(rn);

        }

        //left rotate case

        else if (bf<-1 && rn->rch->data.title\_id >title\_id)

        {

            return leftrotate(rn);

        }

        //left right

        else if (bf > 1 && rn->lch->data.title\_id > title\_id)

        {

            rn->lch = leftrotate(rn->lch);

            return rightrotate(rn);

        }

        //right left

        else if (bf < -1 && rn->rch->data.title\_id < title\_id)

        {

            rn->rch = rightrotate(rn->rch);

            return leftrotate(rn);

        }

        return rn;

    }

};

//Class movie\_avl for AVL functions implementations

class Movie\_AVL

{

public:

    MovieNode\* root;

    Movie\_AVL()

    {

        root = NULL;

    }

    //Function which returns the address of newly created node

    MovieNode\* create\_node(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        MovieNode\* newnode = new MovieNode(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        return newnode;

    }

    //Function which returns max of two values

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    //Function which returns the heigth of current node

    int get\_height(MovieNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    //Function to return the balance factor of two nodes(hieght\_of\_lefttree-height \_of\_rightsubtree)

    int getBalanceFactor(MovieNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lch) - get\_height(n->rch));

    }

    //Function which right rotates the passed node

    MovieNode\* rightrotate(MovieNode\* y)

    {

        MovieNode\* x = y->lch;

        MovieNode\* T2 = x->rch;

        x->rch = y;

        y->lch = T2;

        //Update height after Rotations

        y->height1 = max(get\_height(y->rch), get\_height(y->lch)) + 1;

        x->height1 = max(get\_height(x->rch), get\_height(x->lch)) + 1;

        //Returns the new Parent after Rotation

        return x;

    }

    //Function which left rotates the passed node

    MovieNode\* leftrotate(MovieNode\* x)

    {

        MovieNode\* y = x->rch;

        MovieNode\* T2 = y->lch;

        y->lch = x;

        x->rch = T2;

        //Update height after rotations

        x->height1 = max(get\_height(x->rch), get\_height(x->lch)) + 1;

        y->height1 = max(get\_height(y->rch), get\_height(y->lch)) + 1;

        //Returns the new Parent after Rotation

        return y;

    }

    MovieNode\* insert(MovieNode\* n, string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        //Simple BST insertion

        if (n == NULL)

        {

            return create\_node(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        if (title\_id < n->data.title\_id)

        {

            n->lch = insert(n->lch, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else if (title\_id > n->data.title\_id)

        {

            n->rch = insert(n->rch, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else//Duplication is not allowed

            return n;

        //Update heigth After every Insertion

        n->height1 = 1 + max(get\_height(n->lch), get\_height(n->rch));

        //Checking balance node of current node wheter it becomes unbalakced tree or not

        int bf = getBalanceFactor(n);

        //if node become unbalanced then there are 4 rotations cases

        //right rotate case

        if (bf > 1 && title\_id < n->lch->data.title\_id)

        {

            return rightrotate(n);

        }

        //left rotate case

        else if (bf<-1 && title\_id > n->rch->data.title\_id)

        {

            return leftrotate(n);

        }

        //left right

        else if (bf > 1 && title\_id > n->lch->data.title\_id)

        {

            n->lch = leftrotate(n->lch);

            return rightrotate(n);

        }

        //right left

        else if (bf < -1 && title\_id < n->rch->data.title\_id)

        {

            n->rch = rightrotate(n->rch);

            return leftrotate(n);

        }

        return n;

    }

    //void Print\_PreOrder(MovieNode\* temp) {

    //  if (temp == NULL) {

    //      //Base Case

    //      return;

    //  }

    //  //Go to the left child recursively

    //  Print\_PreOrder(temp->lch);

    //  //print node

    //  cout << temp->data.genre << endl;

    //  //Then go to the right child recursively

    //  Print\_PreOrder(temp->rch);

    //}

};

//Hash Class to implement hash

class movie\_hash\_year

{

    //public data members

public:

    //initializing arr of size ARR\_SIZE to NUL

    MovieNode\* arr\_of\_keys[ARR\_SIZE] = { NULL };

    //FUnction to find hash\_value

    int find\_hash\_value(string);

    //Function to input\_value\_in\_hash

    void input\_value\_in\_hash(int, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string);

};

//global object of movie\_avl\_year to use its functions..

movie\_avl\_year mavl;

int movie\_hash\_year::find\_hash\_value(string year)

{

    //Finding hash index index and returning it

    int temp;

    stringstream geek(year);

    geek >> temp;

    return temp % 23;

}

void movie\_hash\_year::input\_value\_in\_hash(int index, string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

{

    //Calling avl  insert function which returns its root and stores it in index of Hash\_table

    arr\_of\_keys[index] = mavl.insert(arr\_of\_keys[index], title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

}

//Class movie\_avl\_genre to implement avl function based on the value of genre

class movie\_avl\_genre

{

    //Public member functions

public:

    //Function which returns the address of newly created node

    MovieNode\* create\_node(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_Start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        MovieNode\* newnode = new MovieNode(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_Start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        return newnode;

    }

    //Function which returns max of two values

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    //Function which returns the heigth of current node

    int get\_height(MovieNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    //Function to return the balance factor of two nodes(hieght\_of\_lefttree-height \_of\_rightsubtree)

    int getBalanceFactor(MovieNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lch) - get\_height(n->rch));

    }

    //Function which right rotates the passed node

    MovieNode\* rightrotate(MovieNode\* y)

    {

        MovieNode\* x = y->lch;

        MovieNode\* T2 = x->rch;

        x->rch = y;

        y->lch = T2;

        //Update height after Rotations

        y->height1 = max(get\_height(y->rch), get\_height(y->lch)) + 1;

        x->height1 = max(get\_height(x->rch), get\_height(x->lch)) + 1;

        //Returns the new Parent after Rotation

        return x;

    }

    //Function which left rotates the passed node

    MovieNode\* leftrotate(MovieNode\* x)

    {

        MovieNode\* y = x->rch;

        MovieNode\* T2 = y->lch;

        y->lch = x;

        x->rch = T2;

        //Update height after rotations

        x->height1 = max(get\_height(x->rch), get\_height(x->lch)) + 1;

        y->height1 = max(get\_height(y->rch), get\_height(y->lch)) + 1;

        //Returns the new Parent after Rotation

        return y;

    }

    //Function to insert values on AVL tree

    MovieNode\* insert(MovieNode\* rn, string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        //Simple BST insertion

        if (rn == NULL)

        {

            return create\_node(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        if (rn->data.title\_id < title\_id)

        {

            rn->lch = insert(rn->lch, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else if (rn->data.title\_id > title\_id)

        {

            rn->rch = insert(rn->rch, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else//Duplication is not allowed

            return rn;

        //Update heigth After every Insertion

        rn->height1 = 1 + max(get\_height(rn->lch), get\_height(rn->rch));

        //Checking balance node of current node wheter it becomes unbalakced tree or not

        int bf = getBalanceFactor(rn);

        //if node become unbalanced then there are 4 rotations cases

        //right rotate case

        if (bf > 1 && rn->lch->data.title\_id < title\_id)

        {

            return rightrotate(rn);

        }

        //left rotate case

        else if (bf<-1 && rn->rch->data.title\_id > title\_id)

        {

            return leftrotate(rn);

        }

        //left right

        else if (bf > 1 && rn->lch->data.title\_id > title\_id)

        {

            rn->lch = leftrotate(rn->lch);

            return rightrotate(rn);

        }

        //right left

        else if (bf < -1 && rn->rch->data.title\_id < title\_id)

        {

            rn->rch = rightrotate(rn->rch);

            return leftrotate(rn);

        }

        return rn;

    }

};

//CLass to implement hash table

class movie\_hash\_genre

{

public:

    //initiallizing array of size ARR\_SIZE to NULL

    MovieNode\* arr\_of\_keys[ARR\_SIZE] = { NULL };

    // function to return index

    int find\_hash\_value(string);

    //function to input address of avl into hash through chaining

    void input\_value\_in\_hash(int, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string);

};

//global object of class movie\_avl\_genre to use its functions

movie\_avl\_genre mavl1;

int movie\_hash\_genre::find\_hash\_value(string genre)

{

    int sum = 0;

    for (int i = 0; i < genre.length(); i++) {

        sum += (genre[i] \* (int)pow(PRIME\_CONST, i)) % ARR\_SIZE;

    }

    return sum % ARR\_SIZE;

}

void movie\_hash\_genre::input\_value\_in\_hash(int index, string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

{

    //calling  avl of class genre which retruns the root of avl and its inserted to arr at an index

    arr\_of\_keys[index] = mavl1.insert(arr\_of\_keys[index], title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

}

//Function to store data read ffrom Rating.csv

class Ratings\_data

{

public:

    //public data members

    string imdb\_title\_id;

    double weighted\_average\_vote;

    int total\_votes;

    double mean\_vote;

    int median\_vote;

    int votes\_10;

    int votes\_9;

    int votes\_8;

    int votes\_7;

    int votes\_6;

    int votes\_5;

    int votes\_4;

    int votes\_3;

    int votes\_2;

    int votes\_1;

    double allgenders\_0age\_avg\_vote;

    int allgenders\_0age\_votes;

    double allgenders\_18age\_avg\_vote;

    int allgenders\_18age\_votes;

    double allgenders\_30age\_avg\_vote;

    int allgenders\_30age\_votes;

    double allgenders\_45age\_avg\_vote;

    int allgenders\_45age\_votes;

    double males\_allages\_avg\_vote;

    int males\_allages\_votes;

    double males\_0age\_avg\_vote;

    int males\_0age\_votes;

    double males\_18age\_avg\_vote;

    int males\_18age\_votes;

    double males\_30age\_avg\_vote;

    int males\_30age\_votes;

    double males\_45age\_avg\_vote;

    int males\_45age\_votes;

    double females\_allages\_avg\_vote;

    int females\_allages\_votes;

    double females\_0age\_avg\_vote;

    int females\_0age\_votes;

    double females\_18age\_avg\_vote;

    int females\_18age\_votes;

    double females\_30age\_avg\_vote;

    int females\_30age\_votes;

    double females\_45age\_avg\_vote;

    int females\_45age\_votes;

    double top1000\_voters\_rating;

    int top1000\_voters\_votes;

    double us\_voters\_rating;

    int us\_voters\_votes;

    double non\_us\_voters\_rating;

    int non\_us\_voters\_votes;

    MovieNode\* movie\_ptr;

};

//Class use to store data into the data fields of rating which also stores the building varialbles of avl tree

class RatingsNode {

public:

    RatingsNode(string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, MovieNode\*);

    Ratings\_data data;

    RatingsNode\* lchild;

    RatingsNode\* rchild;

    int height1;

};

//Paramerterized constructor which passes values to Data class and stores in the varaible for a specific node

RatingsNode::RatingsNode(string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes, MovieNode\* movie\_ptr) {

    int temp;

    stringstream geek(allgenders\_0age\_avg\_vote);

    geek >> temp;

    data.allgenders\_0age\_avg\_vote = temp;

    stringstream geek1(allgenders\_0age\_votes);

    geek1 >> temp;

    data.allgenders\_0age\_votes = temp;

    stringstream geek2(allgenders\_18age\_avg\_vote);

    geek2 >> temp;

    data.allgenders\_18age\_avg\_vote = temp;

    stringstream geek3(allgenders\_18age\_votes);

    geek3 >> temp;

    data.allgenders\_18age\_votes = temp;

    stringstream geek4(allgenders\_30age\_avg\_vote);

    geek4 >> temp;

    data.allgenders\_30age\_avg\_vote = temp;

    stringstream geek5(allgenders\_30age\_votes);

    geek5 >> temp;

    data.allgenders\_30age\_votes = temp;

    stringstream geek6(allgenders\_45age\_avg\_vote);

    geek6 >> temp;

    data.allgenders\_45age\_avg\_vote = temp;

    stringstream geek7(allgenders\_45age\_votes);

    geek7 >> temp;

    data.allgenders\_45age\_votes = temp;

    stringstream geek8(females\_0age\_avg\_vote);

    geek8 >> temp;

    data.females\_0age\_avg\_vote = temp;

    stringstream geek9(females\_0age\_votes);

    geek9 >> temp;

    data.females\_0age\_votes = temp;

    stringstream geek10(females\_0age\_avg\_vote);

    geek10 >> temp;

    data.females\_18age\_avg\_vote = temp;

    stringstream geek11(females\_18age\_votes);

    geek11 >> temp;

    data.females\_18age\_votes = temp;

    stringstream geek12(females\_30age\_avg\_vote);

    geek12 >> temp;

    data.females\_30age\_avg\_vote = temp;

    stringstream geek13(females\_30age\_votes);

    geek13 >> temp;

    data.females\_30age\_votes = temp;

    stringstream geek14(females\_45age\_avg\_vote);

    geek14 >> temp;

    data.females\_45age\_avg\_vote = temp;

    stringstream geek15(females\_45age\_votes);

    geek15 >> temp;

    data.females\_45age\_votes = temp;

    stringstream geek16(females\_allages\_avg\_vote);

    geek16 >> temp;

    data.females\_allages\_avg\_vote = temp;

    stringstream geek17(females\_allages\_votes);

    geek17 >> temp;

    data.females\_allages\_votes = temp;

    stringstream geek18(males\_0age\_avg\_vote);

    geek18 >> temp;

    data.males\_0age\_avg\_vote = temp;

    stringstream geek19(males\_0age\_votes);

    geek19 >> temp;

    data.males\_0age\_votes = temp;

    stringstream geek20(males\_0age\_avg\_vote);

    geek20 >> temp;

    data.males\_18age\_avg\_vote = temp;

    stringstream geek21(males\_18age\_votes);

    geek21 >> temp;

    data.males\_18age\_votes = temp;

    stringstream geek22(males\_30age\_avg\_vote);

    geek22 >> temp;

    data.males\_30age\_avg\_vote = temp;

    stringstream geek23(males\_30age\_votes);

    geek23 >> temp;

    data.males\_30age\_votes = temp;

    stringstream geek24(males\_45age\_avg\_vote);

    geek24 >> temp;

    data.males\_45age\_avg\_vote = temp;

    stringstream geek25(males\_45age\_votes);

    geek25 >> temp;

    data.males\_45age\_votes = temp;

    stringstream geek26(males\_allages\_avg\_vote);

    geek26 >> temp;

    data.males\_allages\_avg\_vote = temp;

    stringstream geek27(males\_allages\_votes);

    geek27 >> temp;

    data.males\_allages\_votes = temp;

    data.imdb\_title\_id = imdb\_title\_id;

    stringstream geek29(mean\_vote);

    geek29 >> temp;

    data.mean\_vote = temp;

    stringstream geek30(median\_vote);

    geek30 >> temp;

    data.median\_vote = temp;

    stringstream geek31(non\_us\_voters\_rating);

    geek31 >> temp;

    data.non\_us\_voters\_rating = temp;

    stringstream geek32(non\_us\_voters\_votes);

    geek32 >> temp;

    data.non\_us\_voters\_votes = temp;

    stringstream geek33(top1000\_voters\_rating);

    geek33 >> temp;

    data.top1000\_voters\_rating = temp;

    stringstream geek34(top1000\_voters\_votes);

    geek34 >> temp;

    data.top1000\_voters\_votes = temp;

    stringstream geek35(total\_votes);

    geek35 >> temp;

    data.total\_votes = temp;

    stringstream geek36(us\_voters\_rating);

    geek36 >> temp;

    data.us\_voters\_rating = temp;

    stringstream geek37(us\_voters\_votes);

    geek37 >> temp;

    data.us\_voters\_votes = temp;

    stringstream geek38(votes\_1);

    geek38 >> temp;

    data.votes\_1 = temp;

    stringstream geek39(votes\_10);

    geek39 >> temp;

    data.votes\_10 = temp;

    stringstream geek40(votes\_2);

    geek40 >> temp;

    data.votes\_2 = temp;

    stringstream geek41(votes\_3);

    geek41 >> temp;

    data.votes\_3 = temp;

    stringstream geek42(votes\_4);

    geek42 >> temp;

    data.votes\_4 = temp;

    stringstream geek43(votes\_5);

    geek43 >> temp;

    data.votes\_5 = temp;

    stringstream geek44(votes\_6);

    geek44 >> temp;

    data.votes\_6 = temp;

    stringstream geek45(votes\_7);

    geek45 >> temp;

    data.votes\_7 = temp;

    stringstream geek46(votes\_8);

    geek46 >> temp;

    data.votes\_8 = temp;

    stringstream geek47(votes\_9);

    geek47 >> temp;

    data.votes\_9 = temp;

    stringstream geek48(weighted\_average\_vote);

    geek48 >> temp;

    data.weighted\_average\_vote = temp;

    //string pointer of movie node

    data.movie\_ptr = movie\_ptr;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

//class to implement functions related to Ratin AVL

class Ratings\_AVL

{

public:

    //public member function and data membrs

    RatingsNode\* root;

    Ratings\_AVL()

    {

        root = NULL;

    }

    //Function to create a node

    RatingsNode\* create\_node(string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes, MovieNode\* movie\_ptr)

    {

        RatingsNode\* newnode = new RatingsNode(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes, movie\_ptr);

        return newnode;

    }

    //Function to returna max of two values

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    //function whihch returns hieght of current node

    int get\_height(RatingsNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    //function which returns the balance factor of the current node

    int getBalanceFactor(RatingsNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    //Function to Right Rotate the current node

    RatingsNode\* rightrotate(RatingsNode\* y)

    {

        RatingsNode\* x = y->lchild;

        RatingsNode\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    //Function to left rotate the current node

    RatingsNode\* leftrotate(RatingsNode\* x)

    {

        RatingsNode\* y = x->rchild;

        RatingsNode\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    //Function to insert node in the AVL tree

    RatingsNode\* insert(RatingsNode\* n, string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes, MovieNode\* movie\_ptr)

    {

        //simple BST insertion procedure

        if (n == NULL)

        {

            return create\_node(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes, movie\_ptr);

        }

        if (imdb\_title\_id < n->data.imdb\_title\_id)

        {

            n->lchild = insert(n->lchild, imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes, movie\_ptr);

        }

        else if (imdb\_title\_id > n->data.imdb\_title\_id)

        {

            n->rchild = insert(n->rchild, imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes, movie\_ptr);

        }

        else//Duplication is not allowed

            return n;

        //Update heigth After every Insertion

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        //Checking balance node of current node wheter it becomes unbalakced tree or not

        int bf = getBalanceFactor(n);

        //if node become unbalanced then there are 4 rotations cases

        //right rotate case

        if (bf > 1 && imdb\_title\_id < n->lchild->data.imdb\_title\_id)

        {

            return rightrotate(n);

        }

        //left rotate case

        else if (bf<-1 && imdb\_title\_id > n->rchild->data.imdb\_title\_id)

        {

            return leftrotate(n);

        }

        //left right

        else if (bf > 1 && imdb\_title\_id > n->lchild->data.imdb\_title\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        //right left

        else if (bf < -1 && imdb\_title\_id < n->rchild->data.imdb\_title\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

};

//Class to implement Hash Table for Ratings file

class ratings\_hash

{

    //public data members and member function

public:

    RatingsNode\* arr\_of\_keys[ARR\_SIZE] = { NULL };

    //Function to find hash index

    int find\_hash\_value(string);

    //TO input avl in index position thorugh chainin

    void input\_value\_in\_hash(int, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, MovieNode\*);

};

//Global object of Ratings\_AVL to use its functions

Ratings\_AVL ravl1;

int ratings\_hash::find\_hash\_value(string ratings)

{

    double temp;

    stringstream geek(ratings);

    geek >> temp;

    if (temp > 0 && temp <= 1)

    {

        return 0;

    }

    else if (temp > 1 && temp <= 2)

    {

        return 1;

    }

    else if (temp > 2 && temp <= 3)

    {

        return 2;

    }

    else if (temp > 3 && temp <= 4)

    {

        return 3;

    }

    else if (temp > 4 && temp <= 5)

    {

        return 4;

    }

    else if (temp > 5 && temp <= 6)

    {

        return 5;

    }

    else if (temp > 6 && temp <= 7)

    {

        return 6;

    }

    else if (temp > 7 && temp <= 8)

    {

        return 7;

    }

    else if (temp > 8 && temp <= 9)

    {

        return 8;

    }

    else if (temp > 9 && temp <= 10)

    {

        return 9;

    }

}

void ratings\_hash::input\_value\_in\_hash(int index, string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes, MovieNode\* movie\_ptr)

{

    arr\_of\_keys[index] = ravl1.insert(arr\_of\_keys[index], imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes, movie\_ptr);

}

//----------------------------------------------------------------------------------------------

//----------------------------------------------------------------------------------------------

//Function to READ two files

//Movies.csv and Ratings.csv

//Created an AVL tree just for Movie file and 3 Hash tables 2 for Movies bases of different values and 1 avl for rating files

void read(Movie\_AVL\* ml, movie\_hash\_year\* mhash, movie\_hash\_genre\* mhash1, ratings\_hash\* rhash) {

    //---------Movie  Storage------------

    int count = 0;

    string title\_id;

    string title;

    string original\_title;

    string year;

    string date;

    string genre;

    string duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    string actors;

    string description;

    string age\_vote;

    string votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    string meta\_score;

    string reviews\_from\_user;

    string reviews\_from\_critics;

    ifstream File("movies.csv");

    if (File.is\_open()) {

        while (!File.eof()) {

            if (count == 0) {

                string temp;

                getline(File, temp);

                count = 1;

            }

            else {

                getline(File, title\_id, ',');

                getline(File, title, ',');

                getline(File, original\_title, ',');

                getline(File, year, ',');

                getline(File, date, ',');

                getline(File, genre, ',');

                getline(File, duration, ',');

                getline(File, country, ',');

                getline(File, language, ',');

                getline(File, director, ',');

                getline(File, writer, ',');

                getline(File, production\_company, ',');

                getline(File, actors, ',');

                getline(File, description, ',');

                getline(File, age\_vote, ',');

                getline(File, votes, ',');

                getline(File, budget, ',');

                getline(File, usa\_gross\_income, ',');

                getline(File, meta\_score, ',');

                getline(File, reviews\_from\_user, ',');

                getline(File, reviews\_from\_critics);

                //Creating Hash Tables based on the value of year

                mhash->input\_value\_in\_hash(mhash->find\_hash\_value(year), title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

                //Creating Hash based on the value of genre

                //1 column in csv file of movies contains many genre separated by '-' so delimiting it and making hash values on there individual basis

                string s = genre;

                string delimiter = "-";

                size\_t pos = 0;

                string token;

                bool flag = false;

                int index;

                while ((pos = s.find(delimiter)) != string::npos) {

                    token = s.substr(0, pos);

                    index = mhash1->find\_hash\_value(token);

                    mhash1->input\_value\_in\_hash(index, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

                    s.erase(0, pos + delimiter.length());

                }

                index = mhash1->find\_hash\_value(s);

                mhash1->input\_value\_in\_hash(index, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

                ml->root = ml->insert(ml->root, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

            }

        }

    }

    else {

        cout << "Give Error\n" << endl;

    }

    //Now Reading file Ratings

    count = 0;

    string imdb\_title\_id;

    string weighted\_average\_vote;

    string  total\_votes;

    string mean\_vote;

    string median\_vote;

    string votes\_10;

    string votes\_9;

    string votes\_8;

    string votes\_7;

    string  votes\_6;

    string  votes\_5;

    string  votes\_4;

    string  votes\_3;

    string  votes\_2;

    string  votes\_1;

    string allgenders\_0age\_avg\_vote;

    string allgenders\_0age\_votes;

    string allgenders\_18age\_avg\_vote;

    string allgenders\_18age\_votes;

    string allgenders\_30age\_avg\_vote;

    string allgenders\_30age\_votes;

    string allgenders\_45age\_avg\_vote;

    string allgenders\_45age\_votes;

    string males\_allages\_avg\_vote;

    string males\_allages\_votes;

    string males\_0age\_avg\_vote;

    string males\_0age\_votes;

    string males\_18age\_avg\_vote;

    string males\_18age\_votes;

    string males\_30age\_avg\_vote;

    string males\_30age\_votes;

    string males\_45age\_avg\_vote;

    string males\_45age\_votes;

    string females\_allages\_avg\_vote;

    string females\_allages\_votes;

    string females\_0age\_avg\_vote;

    string females\_0age\_votes;

    string females\_18age\_avg\_vote;

    string females\_18age\_votes;

    string females\_30age\_avg\_vote;

    string females\_30age\_votes;

    string females\_45age\_avg\_vote;

    string females\_45age\_votes;

    string top1000\_voters\_rating;

    string top1000\_voters\_votes;

    string us\_voters\_rating;

    string us\_voters\_votes;

    string non\_us\_voters\_rating;

    string non\_us\_voters\_votes;

    ifstream File1("ratings.csv");

    if (File1.is\_open()) {

        while (!File1.eof()) {

            if (count == 0) {

                string temp;

                getline(File1, temp);

                count = 1;

            }

            else {

                getline(File1, imdb\_title\_id, ',');

                getline(File1, weighted\_average\_vote, ',');

                getline(File1, total\_votes, ',');

                getline(File1, mean\_vote, ',');

                getline(File1, median\_vote, ',');

                getline(File1, votes\_10, ',');

                getline(File1, votes\_9, ',');

                getline(File1, votes\_8, ',');

                getline(File1, votes\_7, ',');

                getline(File1, votes\_6, ',');

                getline(File1, votes\_5, ',');

                getline(File1, votes\_4, ',');

                getline(File1, votes\_3, ',');

                getline(File1, votes\_2, ',');

                getline(File1, votes\_1, ',');

                getline(File1, allgenders\_0age\_avg\_vote, ',');

                getline(File1, allgenders\_0age\_votes, ',');

                getline(File1, allgenders\_18age\_avg\_vote, ',');

                getline(File1, allgenders\_18age\_votes, ',');

                getline(File1, allgenders\_30age\_avg\_vote, ',');

                getline(File1, allgenders\_30age\_votes, ',');

                getline(File1, allgenders\_45age\_avg\_vote, ',');

                getline(File1, allgenders\_45age\_votes, ',');

                getline(File1, males\_0age\_avg\_vote, ',');

                getline(File1, males\_0age\_votes, ',');

                getline(File1, males\_18age\_avg\_vote, ',');

                getline(File1, males\_18age\_votes, ',');

                getline(File1, males\_30age\_avg\_vote, ',');

                getline(File1, males\_30age\_votes, ',');

                getline(File1, males\_45age\_avg\_vote, ',');

                getline(File1, males\_45age\_votes, ',');

                getline(File1, females\_allages\_avg\_vote, ',');

                getline(File1, females\_allages\_votes, ',');

                getline(File1, females\_0age\_avg\_vote, ',');

                getline(File1, females\_0age\_votes, ',');

                getline(File1, females\_18age\_avg\_vote, ',');

                getline(File1, females\_18age\_votes, ',');

                getline(File1, females\_30age\_avg\_vote, ',');

                getline(File1, females\_30age\_votes, ',');

                getline(File1, females\_45age\_avg\_vote, ',');

                getline(File1, top1000\_voters\_rating, ',');

                getline(File1, top1000\_voters\_votes, ',');

                getline(File1, us\_voters\_rating, ',');

                getline(File1, us\_voters\_votes, ',');

                getline(File1, non\_us\_voters\_rating, ',');

                getline(File1, non\_us\_voters\_votes);

                //Creating Hash Table on the basis of Rating(wieghted average votes)

                MovieNode\* traverser = ml->root;

                //Traversing through whole movie avl and finsing the AVL node of same imdb titles ad in rating

                while (traverser != NULL && traverser->data.title\_id != imdb\_title\_id) {

                    if (traverser->data.title\_id > imdb\_title\_id) {

                        traverser = traverser->lch;

                    }

                    else {

                        traverser = traverser->rch;

                    }

                }

                //checking if the upper while loop breaks on equality

                if (traverser != NULL) {

                    //creating hash table on the values of weighted average vote and aslo the Address of Movie node of same imdb titles....

                    rhash->input\_value\_in\_hash(rhash->find\_hash\_value(weighted\_average\_vote), imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes, traverser);

                }

                //  rl->insert(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

            }

        }

    }

    else {

        cout << "Give Error1\n";

    }

}

//-----------------------------------------------------------

//-----------------------------------------------------------

//Function to Travers movies

void traverse\_movies(MovieNode\* start)

{

    if (start == NULL) {

        return;

    }

    cout << start->data.title << endl;

    traverse\_movies(start->lch);

    traverse\_movies(start->rch);

}

//Function check specific genre

void print\_specific\_genre(movie\_hash\_genre mh1, string genre)

{

    cout << "All " << genre << " movies are :\n";

    int index = mh1.find\_hash\_value(genre);

    traverse\_movies(mh1.arr\_of\_keys[index]);

}

//-----------------------------------------------------------

//-----------------------------------------------------------

//Function to traverse moveis based on year

void traverse\_movies\_per\_year(MovieNode\* start, int year)

{

    if (start == NULL) {

        return;

    }

    else {

        if(year==start->data.year)

            cout << start->data.original\_title << endl;

        traverse\_movies\_per\_year(start->lch, year);

        traverse\_movies\_per\_year(start->rch, year);

    }

}

//Function to print al movies of a certain year

void print\_selected\_year(movie\_hash\_year mh, string year)

{

    cout << "All movies in year " << year << " are :\n";

    int index = mh.find\_hash\_value(year);

    int temp;

    stringstream geek(year);

    geek >> temp;

    traverse\_movies\_per\_year(mh.arr\_of\_keys[index], temp);

}

//-----------------------------------------------------------

void traverse\_movies3(RatingsNode\* r)

{

    if (r == NULL)

        return;

    cout << r->data.movie\_ptr->data.title << endl;

    traverse\_movies3(r->rchild);

    traverse\_movies3(r->lchild);

}

void print\_in\_avgevote(string rating, ratings\_hash rhash)

{

    int index = rhash.find\_hash\_value(rating);

    cout << "All movies in weighted average weight of range " << index + 1 << " and " << index << " are \n";

    traverse\_movies3(rhash.arr\_of\_keys[index]);

}

int main() {

    Movie\_AVL ml;

    movie\_hash\_year mhash;

    movie\_hash\_genre mhash1;

    ratings\_hash rhash1;

    bool menu = true;

    int option;

    string rating = "";

    string genre = "";

    string year = "";

    read(&ml, &mhash, &mhash1, &rhash1);

    cout << "\t\t\t\t\t\t----IMDB----" << endl;

    cout << "\n\t\t\t\t\t     1. Genre Wise Movies" << endl;

    cout << "\n\t\t\t\t\t     2. Rating Search" << endl;

    cout << "\n\t\t\t\t\t     3. Year Wise Movies" << endl;

    cout << "\n\t\t\t\t\t\t  4. Exit" << endl;

    while (menu) {

        cout << "\nEnter Option : ";

        cin >> option;

        switch (option)

        {

        case 1:

            cout << "Enter Genre : ";

            cin >> genre;

            print\_specific\_genre(mhash1, genre);

            break;

        case 2:

            cout << "Enter Rating : ";

            cin >> rating;

            print\_in\_avgevote(rating, rhash1);

            break;

        case 3:

            cout << "Enter Year : ";

            cin >> year;

            print\_selected\_year(mhash, year);

            break;

        default:

            menu = false;

            break;

        }

    }

    return 0;

}

***NOTE: Big data set is used in hash code with partial output screenshots attached below.***

Text

Description automatically generated

Shape

Description automatically generated

Text

Description automatically generated

Shape

Description automatically generated with low confidence

Text

Description automatically generated

# Linked Lists:

#include<iostream>

#include<string>

#include<fstream>

#include<sstream>

#include<algorithm>

using namespace std;

class actor\_movie\_node;

class MovieNode;

//-----------------Names-------------------link list

class Name\_Data

{

public:

    string name\_id;

    string name;

    string birth\_name;

    int height;

    string bio;

    string birth\_details;

    string date\_of\_birth;

    string place\_of\_birth;

    string death\_details;

    string date\_of\_death; string place\_of\_death; string reason\_of\_death; string spouses\_string; int spouses; int divorces; int spouses\_with\_children; int children;

};

//Name Node in a Linked List

class NameNode

{

public:

    NameNode(string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children);

    Name\_Data data;

    NameNode\* next;

};

//Name Node Constructor to get data

NameNode::NameNode(string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children)

{

    data.name\_id = name\_id;

    data.name = name;

    data.birth\_name = birth\_name;

    data.bio = bio;

    data.birth\_details = birth\_details;

    data.death\_details = death\_details;

    data.place\_of\_birth = place\_of\_birth;

    data.place\_of\_death = place\_of\_death;

    data.reason\_of\_death = reason\_of\_death;

    data.spouses\_string = spouses\_string;

    int temp;

    stringstream geek(height);

    geek >> temp;

    data.height = temp;

    stringstream geek1(spouses);

    geek1 >> temp;

    data.spouses = temp;

    stringstream geek2(divorces);

    geek2 >> temp;

    data.divorces = temp;

    stringstream geek3(spouses\_with\_children);

    geek3 >> temp;

    data.spouses\_with\_children = temp;

    stringstream geek4(children);

    geek4 >> temp;

    data.children = temp;

}

//List of names to Name\_Nodes

class NameList

{

public:

    NameNode\* start;

    NameNode\* last;

    NameList();

    //Insertion into the list

    void insert(string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children);

    void display();

};

//NameList Constructor that initializes the list

NameList::NameList()

{

    start = NULL;

    last = NULL;

}

//Insert Function

void NameList::insert(string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children)

{

    NameNode\* newnode = new NameNode(name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

    if (start == NULL) {

        start = newnode;

        last = newnode;

    }

    else {

        last->next = newnode;

        last = newnode;

    }

}

//Display Name List

void NameList::display()

{

    NameNode\* temp = start;

    while (temp != NULL) {

        cout << temp->data.name << endl;

        temp = temp->next;

    }

}

//Node contain pointer towards a movie

class movie\_node\_ptr {

public:

    movie\_node\_ptr();

    MovieNode\* m\_ptr;

    movie\_node\_ptr\* next;

};

movie\_node\_ptr::movie\_node\_ptr() {

    m\_ptr = NULL;

    next = NULL;

}

//Actor Node for actor list and contains pointer towards name list nodes

class Actor\_Node {

public:

    Actor\_Node();

    NameNode\* ptr;

    movie\_node\_ptr\* movie\_ptr;

    Actor\_Node\* next;

};

//Actor Node Constructor

Actor\_Node::Actor\_Node() {

    ptr = NULL;

    movie\_ptr = NULL;

    next = NULL;

}

//Contains Actor list functions

class Actor\_List {

public:

    Actor\_List();

    Actor\_Node\* start;

    Actor\_Node\* last;

    void insert(NameNode\*);

    void display();

};

Actor\_List::Actor\_List() {

    start = NULL;

    last = NULL;

}

//insertion just like previous fields

void Actor\_List::insert(NameNode\* ptr) {

    Actor\_Node\* newnode = new Actor\_Node();

    newnode->ptr = ptr;

    if (start == NULL) {

        start = newnode;

        last = newnode;

    }

    else {

        last->next = newnode;

        last = newnode;

    }

}

//Display actors in list

void Actor\_List::display() {

    Actor\_Node\* triv = start;

    while (triv != NULL) {

        cout << triv->ptr->data.name << endl;

        triv = triv->next;

    }

}

//Genre\_Node is the node of genre list

class Genre\_Node {

public:

    Genre\_Node();

    Genre\_Node\* next;

    movie\_node\_ptr\* movie\_ptr;

    string genre;

};

//Genre Node initializer

Genre\_Node::Genre\_Node() {

    next = NULL;

    movie\_ptr = NULL;

}

//Genre List that contains genre and their movies pointer

class Genre\_List {

public:

    Genre\_List();

    Genre\_Node\* start;

    Genre\_Node\* last;

    void insert(string);

    void display();

};

//Genre List

Genre\_List::Genre\_List() {

    start = NULL;

    last = NULL;

}

//Insertion

void Genre\_List::insert(string genre) {

    Genre\_Node\* newnode = new Genre\_Node();

    newnode->genre = genre;

    if (start == NULL) {

        start = newnode;

        last = newnode;

    }

    else {

        last->next = newnode;

        last = newnode;

    }

}

void Genre\_List::display() {

    Genre\_Node\* traverser = start;

    while (traverser != NULL) {

        cout << traverser->genre << endl;

        traverser = traverser->next;

    }

}

//Director Node for directors

class Director\_Node {

public:

    Director\_Node();

    NameNode\* ptr;

    movie\_node\_ptr\* movie\_ptr;

    Director\_Node\* next;

};

Director\_Node::Director\_Node() {

    ptr = NULL;

    movie\_ptr = NULL;

    next = NULL;

}

class Director\_List {

public:

    Director\_List();

    Director\_Node\* start;

    Director\_Node\* last;

    void insert(NameNode\*);

    void display();

};

Director\_List::Director\_List() {

    start = NULL;

    last = NULL;

}

void Director\_List::insert(NameNode\* ptr) {

    Director\_Node\* newnode = new Director\_Node();

    newnode->ptr = ptr;

    if (start == NULL) {

        start = newnode;

        last = newnode;

    }

    else {

        last->next = newnode;

        last = newnode;

    }

}

void Director\_List::display() {

    Director\_Node\* triv = start;

    while (triv != NULL) {

        cout << triv->ptr->data.name << endl;

        triv = triv->next;

    }

}

//Actor Pointer Node that contain pointer that points to the actors

class actor\_node\_ptr

{

public:

    Actor\_Node\* ptr;

    actor\_node\_ptr\* next;

    actor\_node\_ptr();

};

actor\_node\_ptr::actor\_node\_ptr()

{

    ptr = NULL;

    next = NULL;

}

//-----------------Movies-------------------

class MovieData {

public:

    string title\_id;

    string title;

    string original\_title;

    int year;

    string date;

    string genre;

    int duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    actor\_node\_ptr \* actor\_Start;

    string description;

    double age\_vote;

    int votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    int meta\_score;

    int reviews\_from\_user;

    int reviews\_from\_critics;

};

//Movie Node for movie

class MovieNode {

public:

    MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, actor\_node\_ptr\* actor\_Start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics);

    MovieNode\* next;

    MovieData data;

};

MovieNode::MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, actor\_node\_ptr\* actor\_Start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics) {

    int temp;

    data.title\_id = title\_id;

    data.title = title;

    data.original\_title = original\_title;

    stringstream geek(year);

    geek >> temp;

    data.year = temp;

    data.date = date;

    data.genre = genre;

    stringstream geek1(duration);

    geek >> temp;

    data.duration = temp;

    data.country = country;

    data.language = language;

    data.director = director;

    data.writer = writer;

    data.production\_company = production\_company;

    data.actor\_Start = actor\_Start;

    data.description = description;

    stringstream geek2(age\_vote);

    geek >> temp;

    data.age\_vote = double(temp);

    stringstream geek6(age\_vote);

    geek >> temp;

    data.votes = temp;

    data.budget = budget;

    data.usa\_gross\_income = usa\_gross\_income;

    stringstream geek3(meta\_score);

    geek >> temp;

    data.meta\_score = temp;

    data.worldwide\_gross\_income = worldwide\_gross\_income;

    stringstream geek4(reviews\_from\_user);

    geek >> temp;

    data.reviews\_from\_user = temp;

    stringstream geek5(reviews\_from\_critics);

    geek >> temp;

    data.reviews\_from\_critics = temp;

}

class MovieList {

public:

    MovieNode\* start;

    MovieNode\* last;

    MovieList();

    void insert(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, actor\_node\_ptr\* actor\_Start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics);

    void display();

};

MovieList::MovieList() {

    start = NULL;

    last = NULL;

}

//insertion and dislay

void MovieList::insert(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, actor\_node\_ptr\* actor\_Start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics) {

    MovieNode\* newnode = new MovieNode(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_Start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

    if (start == NULL) {

        start = newnode;

        last = newnode;

    }

    else {

        last->next = newnode;

        last = newnode;

    }

}

void MovieList::display() {

    MovieNode\* temp = start;

    while (temp != NULL) {

        cout << temp->data.title << endl;

        temp = temp->next;

    }

}

//Node that contains pointer towards a movie done by an actor

class actor\_movie\_node {

public:

    MovieNode\* ptr;

    actor\_movie\_node\* next;

    actor\_movie\_node();

};

actor\_movie\_node::actor\_movie\_node() {

    ptr = NULL;

    next = NULL;

}

//-----------------ratings-------------------

class Ratings\_data

{

public:

    string imdb\_title\_id;

    double weighted\_average\_vote;

    int total\_votes;

    double mean\_vote;

    int median\_vote;

    int votes\_10;

    int votes\_9;

    int votes\_8;

    int votes\_7;

    int votes\_6;

    int votes\_5;

    int votes\_4;

    int votes\_3;

    int votes\_2;

    int votes\_1;

    double allgenders\_0age\_avg\_vote;

    int allgenders\_0age\_votes;

    double allgenders\_18age\_avg\_vote;

    int allgenders\_18age\_votes;

    double allgenders\_30age\_avg\_vote;

    int allgenders\_30age\_votes;

    double allgenders\_45age\_avg\_vote;

    int allgenders\_45age\_votes;

    double males\_allages\_avg\_vote;

    int males\_allages\_votes;

    double males\_0age\_avg\_vote;

    int males\_0age\_votes;

    double males\_18age\_avg\_vote;

    int males\_18age\_votes;

    double males\_30age\_avg\_vote;

    int males\_30age\_votes;

    double males\_45age\_avg\_vote;

    int males\_45age\_votes;

    double females\_allages\_avg\_vote;

    int females\_allages\_votes;

    double females\_0age\_avg\_vote;

    int females\_0age\_votes;

    double females\_18age\_avg\_vote;

    int females\_18age\_votes;

    double females\_30age\_avg\_vote;

    int females\_30age\_votes;

    double females\_45age\_avg\_vote;

    int females\_45age\_votes;

    double top1000\_voters\_rating;

    int top1000\_voters\_votes;

    double us\_voters\_rating;

    int us\_voters\_votes;

    double non\_us\_voters\_rating;

    int non\_us\_voters\_votes;

};

class RatingsNode {

public:

    RatingsNode(string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string);

    Ratings\_data data;

    RatingsNode\* next;

};

RatingsNode::RatingsNode(string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes) {

    int temp;

    stringstream geek(allgenders\_0age\_avg\_vote);

    geek >> temp;

    data.allgenders\_0age\_avg\_vote = temp;

    stringstream geek1(allgenders\_0age\_votes);

    geek1 >> temp;

    data.allgenders\_0age\_votes = temp;

    stringstream geek2(allgenders\_18age\_avg\_vote);

    geek2 >> temp;

    data.allgenders\_18age\_avg\_vote = temp;

    stringstream geek3(allgenders\_18age\_votes);

    geek3 >> temp;

    data.allgenders\_18age\_votes = temp;

    stringstream geek4(allgenders\_30age\_avg\_vote);

    geek4 >> temp;

    data.allgenders\_30age\_avg\_vote = temp;

    stringstream geek5(allgenders\_30age\_votes);

    geek5 >> temp;

    data.allgenders\_30age\_votes = temp;

    stringstream geek6(allgenders\_45age\_avg\_vote);

    geek6 >> temp;

    data.allgenders\_45age\_avg\_vote = temp;

    stringstream geek7(allgenders\_45age\_votes);

    geek7 >> temp;

    data.allgenders\_45age\_votes = temp;

    stringstream geek8(females\_0age\_avg\_vote);

    geek8 >> temp;

    data.females\_0age\_avg\_vote = temp;

    stringstream geek9(females\_0age\_votes);

    geek9 >> temp;

    data.females\_0age\_votes = temp;

    stringstream geek10(females\_0age\_avg\_vote);

    geek10 >> temp;

    data.females\_18age\_avg\_vote = temp;

    stringstream geek11(females\_18age\_votes);

    geek11 >> temp;

    data.females\_18age\_votes = temp;

    stringstream geek12(females\_30age\_avg\_vote);

    geek12 >> temp;

    data.females\_30age\_avg\_vote = temp;

    stringstream geek13(females\_30age\_votes);

    geek13 >> temp;

    data.females\_30age\_votes = temp;

    stringstream geek14(females\_45age\_avg\_vote);

    geek14 >> temp;

    data.females\_45age\_avg\_vote = temp;

    stringstream geek15(females\_45age\_votes);

    geek15 >> temp;

    data.females\_45age\_votes = temp;

    stringstream geek16(females\_allages\_avg\_vote);

    geek16 >> temp;

    data.females\_allages\_avg\_vote = temp;

    stringstream geek17(females\_allages\_votes);

    geek17 >> temp;

    data.females\_allages\_votes = temp;

    stringstream geek18(males\_0age\_avg\_vote);

    geek18 >> temp;

    data.males\_0age\_avg\_vote = temp;

    stringstream geek19(males\_0age\_votes);

    geek19 >> temp;

    data.males\_0age\_votes = temp;

    stringstream geek20(males\_0age\_avg\_vote);

    geek20 >> temp;

    data.males\_18age\_avg\_vote = temp;

    stringstream geek21(males\_18age\_votes);

    geek21 >> temp;

    data.males\_18age\_votes = temp;

    stringstream geek22(males\_30age\_avg\_vote);

    geek22 >> temp;

    data.males\_30age\_avg\_vote = temp;

    stringstream geek23(males\_30age\_votes);

    geek23 >> temp;

    data.males\_30age\_votes = temp;

    stringstream geek24(males\_45age\_avg\_vote);//conversion of string to double

    geek24 >> temp;

    data.males\_45age\_avg\_vote = temp;

    stringstream geek25(males\_45age\_votes);

    geek25 >> temp;

    data.males\_45age\_votes = temp;

    stringstream geek26(males\_allages\_avg\_vote);

    geek26 >> temp;

    data.males\_allages\_avg\_vote = temp;

    stringstream geek27(males\_allages\_votes);

    geek27 >> temp;

    data.males\_allages\_votes = temp;

    stringstream geek28(imdb\_title\_id);

    geek28 >> temp;

    data.imdb\_title\_id = temp;

    stringstream geek29(mean\_vote);

    geek29 >> temp;

    data.mean\_vote = temp;

    stringstream geek30(median\_vote);

    geek30 >> temp;

    data.median\_vote = temp;

    stringstream geek31(non\_us\_voters\_rating);

    geek31 >> temp;

    data.non\_us\_voters\_rating = temp;

    stringstream geek32(non\_us\_voters\_votes);

    geek32 >> temp;

    data.non\_us\_voters\_votes = temp;

    stringstream geek33(top1000\_voters\_rating);

    geek33 >> temp;

    data.top1000\_voters\_rating = temp;

    stringstream geek34(top1000\_voters\_votes);

    geek34 >> temp;

    data.top1000\_voters\_votes = temp;

    stringstream geek35(total\_votes);

    geek35 >> temp;

    data.total\_votes = temp;

    stringstream geek36(us\_voters\_rating);

    geek36 >> temp;

    data.us\_voters\_rating = temp;

    stringstream geek37(us\_voters\_votes);

    geek37 >> temp;

    data.us\_voters\_votes = temp;

    stringstream geek38(votes\_1);

    geek38 >> temp;

    data.votes\_1 = temp;

    stringstream geek39(votes\_10);

    geek39 >> temp;

    data.votes\_10 = temp;

    stringstream geek40(votes\_2);

    geek40 >> temp;

    data.votes\_2 = temp;

    stringstream geek41(votes\_3);

    geek41 >> temp;

    data.votes\_3 = temp;

    stringstream geek42(votes\_4);

    geek42 >> temp;

    data.votes\_4 = temp;

    stringstream geek43(votes\_5);

    geek43 >> temp;

    data.votes\_5 = temp;

    stringstream geek44(votes\_6);

    geek44 >> temp;

    data.votes\_6 = temp;

    stringstream geek45(votes\_7);

    geek45 >> temp;

    data.votes\_7 = temp;

    stringstream geek46(votes\_8);

    geek46 >> temp;

    data.votes\_8 = temp;

    stringstream geek47(votes\_9);

    geek47 >> temp;

    data.votes\_9 = temp;

    stringstream geek48(weighted\_average\_vote);

    geek48 >> temp;

    data.weighted\_average\_vote = temp;

}

class RatingsList {

public:

    RatingsNode\* start;

    RatingsNode\* last;

    RatingsList();

    void insert(string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string);

    void display();

};

RatingsList::RatingsList() {

    start = NULL;

    last = NULL;

}

void RatingsList::insert(string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes) {

    RatingsNode\* newnode = new RatingsNode(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

    if (start == NULL) {

        start = newnode;

        last = newnode;

    }

    else {

        last->next = newnode;

        last = newnode;

    }

}

void RatingsList::display() {

    RatingsNode\* temp = start;

    while (temp != NULL) {

        cout << temp->data.allgenders\_0age\_avg\_vote << endl;

        temp = temp->next;

    }

}

//-----------------title\_principles----------------

class title\_principles\_Data {

public:

    string imdb\_title\_id;

    int order;

    string imdb\_name\_id;

    string category;

    string job;

    string characters;

};

class title\_principles\_Node {

public:

    title\_principles\_Node(string, string, string, string, string, string);

    title\_principles\_Data data;

    title\_principles\_Node\* next;

};

title\_principles\_Node::title\_principles\_Node(string imdb\_title\_id, string order, string imdb\_name\_id, string category, string job, string characters) {

    int temp;

    data.imdb\_title\_id = imdb\_title\_id;

    stringstream geek(order);

    geek >> temp;

    data.order = temp;

    data.imdb\_name\_id = imdb\_name\_id;

    data.category = category;

    data.job = job;

    data.characters = characters;

}

class title\_principles\_Node\_List {

public:

    title\_principles\_Node\* start;

    title\_principles\_Node\* last;

    title\_principles\_Node\_List();

    void insert(string, string, string, string, string, string);

    void display();

};

title\_principles\_Node\_List::title\_principles\_Node\_List() {

    start = NULL;

    last = NULL;

}

void title\_principles\_Node\_List::insert(string imdb\_title\_id, string order, string imdb\_name\_id, string category, string job, string characters) {

    title\_principles\_Node\* newnode = new title\_principles\_Node(imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

    if (start == NULL) {

        start = newnode;

        last = newnode;

    }

    else {

        last->next = newnode;

        last = newnode;

    }

}

void title\_principles\_Node\_List::display() {

    title\_principles\_Node\* temp = start;

    while (temp != NULL) {

        cout << temp->data.imdb\_title\_id << endl;

        temp = temp->next;

    }

}

void read(MovieList\* ml, RatingsList\* rl, title\_principles\_Node\_List\* t1, NameList\* t2, Actor\_List\* al, Director\_List\* dl,Genre\_List\* gl){

    //-----------------name Storage----------------

    int count = 0;

    string name\_id;

    string name;

    string birth\_name;

    string height;

    string bio;

    string birth\_details;

    string date\_of\_birth;

    string place\_of\_birth;

    string death\_details;

    string date\_of\_death; string place\_of\_death; string reason\_of\_death; string spouses\_string; string spouses; string divorces; string spouses\_with\_children; string children;

    ifstream File3("names.csv");

    if (File3.is\_open()) {

        while (!File3.eof()) {

            if (count == 0) {

                string temp;

                getline(File3, temp);

                count = 1;

            }

            else {

                getline(File3, name\_id, ',');

                getline(File3, name, ',');

                getline(File3, birth\_name, ',');

                getline(File3, height, ',');

                getline(File3, bio, ',');

                getline(File3, birth\_details, ',');

                getline(File3, date\_of\_birth, ',');

                getline(File3, place\_of\_birth, ',');

                getline(File3, death\_details, ',');

                getline(File3, date\_of\_death, ',');

                getline(File3, place\_of\_death, ',');

                getline(File3, reason\_of\_death, ',');

                getline(File3, spouses\_string, ',');

                getline(File3, spouses, ',');

                getline(File3, divorces, ',');

                getline(File3, spouses\_with\_children, ',');

                getline(File3, children);

                t2->insert(name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

            }

        }

    }

    else {

        cout << "Give Error3";

    }

    //---------------Title Principal Storage-------------

    count = 0;

    string imdb\_title\_id1;

    string order;

    string imdb\_name\_id1;

    string category;

    string job;

    string characters;

    ifstream File2("title\_principals.csv");

    if (File2.is\_open()) {

        while (!File2.eof()) {

            if (count == 0) {

                string temp;

                getline(File2, temp);

                count = 1;

            }

            else {

                getline(File2, imdb\_title\_id1, ',');

                getline(File2, order, ',');

                getline(File2, imdb\_name\_id1, ',');

                getline(File2, category, ',');

                getline(File2, job, ',');

                getline(File2, characters);

                t1->insert(imdb\_title\_id1, order, imdb\_name\_id1, category, job, characters);

            }

        }

    }

    else {

        cout << "Give Error2";

    }

    //----------------Actor List Formation----------------

    NameNode\* name\_list\_traverser = t2->start;

    while (name\_list\_traverser != NULL) {

        string IMDB\_Name\_ID;

        title\_principles\_Node\* title\_principles\_traverser = t1->start;

        while (title\_principles\_traverser != NULL) {

            if (name\_list\_traverser->data.name\_id == title\_principles\_traverser->data.imdb\_name\_id) {

                if (title\_principles\_traverser->data.category == "actor" || title\_principles\_traverser->data.category == "actress") {

                    al->insert(name\_list\_traverser);

                    break;

                }

            }

            title\_principles\_traverser = title\_principles\_traverser->next;

        }

        name\_list\_traverser = name\_list\_traverser->next;

    }

    //Director List Formation

    name\_list\_traverser = t2->start;

    while (name\_list\_traverser != NULL) {

        string IMDB\_Name\_ID;

        title\_principles\_Node\* title\_principles\_traverser = t1->start;

        while (title\_principles\_traverser != NULL) {

            if (name\_list\_traverser->data.name\_id == title\_principles\_traverser->data.imdb\_name\_id) {

                if (title\_principles\_traverser->data.category == "director") {

                    dl->insert(name\_list\_traverser);

                    break;

                }

            }

            title\_principles\_traverser = title\_principles\_traverser->next;

        }

        name\_list\_traverser = name\_list\_traverser->next;

    }

    //---------Movie List Storage------------

    count = 0;

    string title\_id;

    string title;

    string original\_title;

    string year;

    string date;

    string genre;

    string duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    string actors;

    string description;

    string age\_vote;

    string votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    string meta\_score;

    string reviews\_from\_user;

    string reviews\_from\_critics;

    ifstream File("movies.csv");

    if (File.is\_open()) {

        while (!File.eof()) {

            if (count == 0) {

                string temp;

                getline(File, temp);

                count = 1;

            }

            else {

                getline(File, title\_id, ',');

                getline(File, title, ',');

                getline(File, original\_title, ',');

                getline(File, year, ',');

                getline(File, date, ',');

                getline(File, genre, ',');

                getline(File, duration, ',');

                getline(File, country, ',');

                getline(File, language, ',');

                getline(File, director, ',');

                getline(File, writer, ',');

                getline(File, production\_company, ',');

                getline(File, actors, ',');

                //Add List of all pointers of actors act in a movie

                actor\_node\_ptr\* actor\_Start = NULL;

                string s = actors;

                string delimiter = "-";

                size\_t pos = 0;

                string token;

                Actor\_Node\* traverser = al->start;

                actor\_node\_ptr\* traverser1 = NULL;

                while ((pos = s.find(delimiter)) != string::npos) {

                    token = s.substr(0, pos);

                    traverser = al->start;

                    while (traverser != NULL && traverser->ptr->data.name != token) {

                        traverser = traverser->next;

                    }

                    if (traverser != NULL) {

                        actor\_node\_ptr\* newnode = new actor\_node\_ptr();

                        newnode->ptr = traverser;

                        if (actor\_Start == NULL)

                        {

                            actor\_Start = newnode;

                        }

                        else {

                            traverser1 = actor\_Start;

                            while (traverser1 != NULL && traverser1->next != NULL) {

                                traverser1 = traverser1->next;

                            }

                            traverser1->next = newnode;

                        }

                    }

                    s.erase(0, pos + delimiter.length());

                }

                traverser = al->start;

                while (traverser != NULL && traverser->ptr->data.name != s) {

                    traverser = traverser->next;

                }

                if (traverser != NULL) {

                    actor\_node\_ptr\* newnode = new actor\_node\_ptr();

                    newnode->ptr = traverser;

                    traverser1 = actor\_Start;

                    while (traverser1 != NULL && traverser1->next != NULL) {

                        traverser1 = traverser1->next;

                    }

                    traverser1->next = newnode;

                }

                getline(File, description, ',');

                getline(File, age\_vote, ',');

                getline(File, votes, ',');

                getline(File, budget, ',');

                getline(File, usa\_gross\_income, ',');

                getline(File, meta\_score, ',');

                getline(File, reviews\_from\_user, ',');

                getline(File, reviews\_from\_critics);

                ml->insert(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_Start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

            }

        }

        File.close();

    }

    else {

        cout << "Give Error\n" << endl;

    }

    //---------Rating List Storage------------

    count = 0;

    string imdb\_title\_id;

    string weighted\_average\_vote;

    string  total\_votes;

    string mean\_vote;

    string median\_vote;

    string votes\_10;

    string votes\_9;

    string votes\_8;

    string votes\_7;

    string  votes\_6;

    string  votes\_5;

    string  votes\_4;

    string  votes\_3;

    string  votes\_2;

    string  votes\_1;

    string allgenders\_0age\_avg\_vote;

    string allgenders\_0age\_votes;

    string allgenders\_18age\_avg\_vote;

    string allgenders\_18age\_votes;

    string allgenders\_30age\_avg\_vote;

    string allgenders\_30age\_votes;

    string allgenders\_45age\_avg\_vote;

    string allgenders\_45age\_votes;

    string males\_allages\_avg\_vote;

    string males\_allages\_votes;

    string males\_0age\_avg\_vote;

    string males\_0age\_votes;

    string males\_18age\_avg\_vote;

    string males\_18age\_votes;

    string males\_30age\_avg\_vote;

    string males\_30age\_votes;

    string males\_45age\_avg\_vote;

    string males\_45age\_votes;

    string females\_allages\_avg\_vote;

    string females\_allages\_votes;

    string females\_0age\_avg\_vote;

    string females\_0age\_votes;

    string females\_18age\_avg\_vote;

    string females\_18age\_votes;

    string females\_30age\_avg\_vote;

    string females\_30age\_votes;

    string females\_45age\_avg\_vote;

    string females\_45age\_votes;

    string top1000\_voters\_rating;

    string top1000\_voters\_votes;

    string us\_voters\_rating;

    string us\_voters\_votes;

    string non\_us\_voters\_rating;

    string non\_us\_voters\_votes;

    ifstream File1("ratings.csv");

    if (File1.is\_open()) {

        while (!File1.eof()) {

            if (count == 0) {

                string temp;

                getline(File1, temp);

                count = 1;

            }

            else {

                getline(File1, imdb\_title\_id, ',');

                getline(File1, weighted\_average\_vote, ',');

                getline(File1, total\_votes, ',');

                getline(File1, mean\_vote, ',');

                getline(File1, median\_vote, ',');

                getline(File1, votes\_10, ',');

                getline(File1, votes\_9, ',');

                getline(File1, votes\_8, ',');

                getline(File1, votes\_7, ',');

                getline(File1, votes\_6, ',');

                getline(File1, votes\_5, ',');

                getline(File1, votes\_4, ',');

                getline(File1, votes\_3, ',');

                getline(File1, votes\_2, ',');

                getline(File1, votes\_1, ',');

                getline(File1, votes\_9, ',');

                getline(File1, allgenders\_0age\_avg\_vote, ',');

                getline(File1, allgenders\_0age\_votes, ',');

                getline(File1, allgenders\_18age\_avg\_vote, ',');

                getline(File1, allgenders\_18age\_votes, ',');

                getline(File1, allgenders\_18age\_votes, ',');

                getline(File1, allgenders\_30age\_avg\_vote, ',');

                getline(File1, allgenders\_30age\_avg\_vote, ',');

                getline(File1, allgenders\_45age\_avg\_vote, ',');

                getline(File1, allgenders\_45age\_avg\_vote, ',');

                getline(File1, females\_allages\_avg\_vote, ',');

                getline(File1, females\_allages\_votes, ',');

                getline(File1, females\_0age\_avg\_vote, ',');

                getline(File1, females\_0age\_votes, ',');

                getline(File1, females\_18age\_avg\_vote, ',');

                getline(File1, females\_18age\_votes, ',');

                getline(File1, females\_30age\_avg\_vote, ',');

                getline(File1, females\_30age\_votes, ',');

                getline(File1, females\_45age\_avg\_vote, ',');

                getline(File1, top1000\_voters\_rating, ',');

                getline(File1, top1000\_voters\_votes, ',');

                getline(File1, us\_voters\_rating, ',');

                getline(File1, us\_voters\_votes, ',');

                getline(File1, non\_us\_voters\_rating, ',');

                getline(File1, non\_us\_voters\_votes);

                rl->insert(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

            }

        }

    }

    else {

        cout << "Give Error1\n";

    }

    //Set pointers towards movies of actors

    Actor\_Node\* actor\_triv = al->start;

    while (actor\_triv != NULL) {

        string actor\_name = actor\_triv->ptr->data.name;

        MovieNode\* movie\_triv = ml->start;

        while (movie\_triv!=NULL){

            actor\_node\_ptr\* movie\_actor\_triv = movie\_triv->data.actor\_Start;

            while (movie\_actor\_triv!=NULL) {

                if (actor\_name == movie\_actor\_triv->ptr->ptr->data.name) {

                    break;

                }

                movie\_actor\_triv = movie\_actor\_triv->next;

            }

            if (movie\_actor\_triv != NULL) {

                movie\_node\_ptr\* newnode = new movie\_node\_ptr();

                newnode->m\_ptr = movie\_triv;

                if (actor\_triv->movie\_ptr == NULL) {

                    actor\_triv->movie\_ptr = newnode;

                }

                else {

                    movie\_node\_ptr\* traverser = actor\_triv->movie\_ptr;

                    while (traverser!=NULL && traverser->next != NULL) {

                        traverser = traverser->next;

                    }

                    traverser->next = newnode;

                }

            }

            movie\_triv = movie\_triv->next;

        }

        actor\_triv = actor\_triv->next;

    }

    //Pointers of movies directed by a director

    Director\_Node\* director\_triv = dl->start;

    while (director\_triv != NULL) {

        string director\_name = director\_triv->ptr->data.name;

        MovieNode\* movie\_triv = ml->start;

        while (movie\_triv != NULL) {

            if (movie\_triv->data.director==director\_name) {

                movie\_node\_ptr\* newnode = new movie\_node\_ptr();

                newnode->m\_ptr = movie\_triv;

                if (director\_triv->movie\_ptr == NULL) {

                    director\_triv->movie\_ptr = newnode;

                }

                else {

                    movie\_node\_ptr\* traverser = director\_triv->movie\_ptr;

                    while (traverser!=NULL && traverser->next != NULL) {

                        traverser = traverser->next;

                    }

                    traverser->next = newnode;

                }

            }

            movie\_triv = movie\_triv->next;

        }

        director\_triv = director\_triv->next;

    }

    //Store all Genres

    MovieNode\* m\_traverser = ml->start;

    bool rep;

    while (m\_traverser != NULL)

    {

        Genre\_Node\* g\_traverser = gl->start;

        string s = m\_traverser->data.genre;

        string delimiter = "-";

        size\_t pos = 0;

        string token;

        while ((pos = s.find(delimiter)) != string::npos) {

            token = s.substr(0, pos);

            rep = true;

            Genre\_Node\* triv = gl->start;

            while (triv != NULL)

            {

                if (triv->genre == token)

                    rep = false;

                triv = triv->next;

            }

            if (rep == true)

                gl->insert(token);

            s.erase(0, pos + delimiter.length());

        }

        token = s.substr(0, pos);

        rep = true;

        Genre\_Node\*triv = gl->start;

        while (triv != NULL)

        {

            if (triv->genre == token)

                rep = false;

            triv = triv->next;

        }

        if (rep == true)

            gl->insert(token);

        m\_traverser = m\_traverser->next;

    }

    //Maintain pointers towards movies of certain genre

    Genre\_Node\* g\_triv = gl->start;

    while (g\_triv != NULL) {

        string genre\_name = g\_triv->genre;

        MovieNode\* movie\_triv = ml->start;

        while (movie\_triv != NULL) {

            string s = movie\_triv->data.genre;

            string delimiter = "-";

            size\_t pos = 0;

            string token;

            while ((pos = s.find(delimiter)) != string::npos) {

                token = s.substr(0, pos);

                if (token == genre\_name) {

                    movie\_node\_ptr\* newnode = new movie\_node\_ptr();

                    newnode->m\_ptr = movie\_triv;

                    if (g\_triv->movie\_ptr == NULL) {

                        g\_triv->movie\_ptr = newnode;

                    }

                    else {

                        movie\_node\_ptr\* traverser = g\_triv->movie\_ptr;

                        while (traverser->next != NULL) {

                            traverser = traverser->next;

                        }

                        traverser->next = newnode;

                    }

                }

                s.erase(0, pos + delimiter.length());

            }

            if (s == genre\_name) {

                movie\_node\_ptr\* newnode = new movie\_node\_ptr();

                newnode->m\_ptr = movie\_triv;

                if (g\_triv->movie\_ptr == NULL) {

                    g\_triv->movie\_ptr = newnode;

                }

                else {

                    movie\_node\_ptr\* traverser = g\_triv->movie\_ptr;

                    while (traverser->next != NULL) {

                        traverser = traverser->next;

                    }

                    traverser->next = newnode;

                }

            }

            movie\_triv = movie\_triv->next;

        }

        g\_triv = g\_triv->next;

    }

}

//Determines the coactors by using the existing lists

void co\_actors\_of\_co\_actors(string name, Actor\_List\* al, bool stopper = true) {

    Actor\_Node\* actor\_traverser = al->start;

    while (actor\_traverser != NULL && actor\_traverser->ptr->data.name != name) {

        actor\_traverser = actor\_traverser->next;

    }

    if (actor\_traverser != NULL && actor\_traverser->ptr->data.name == name) {

        movie\_node\_ptr\* movie\_traverser = actor\_traverser->movie\_ptr;

        while (movie\_traverser != NULL) {

            actor\_node\_ptr\* co\_actor\_traverser = movie\_traverser->m\_ptr->data.actor\_Start;

            while (co\_actor\_traverser != NULL) {

                if (stopper == true) {

                    if(co\_actor\_traverser->ptr->ptr->data.name != name)

                        cout << "\nThe co\_actors of co\_actor " << co\_actor\_traverser->ptr->ptr->data.name << " are : " << endl;

                    else

                        cout << "The co\_actors of actor " << co\_actor\_traverser->ptr->ptr->data.name << " are : " << endl;

                    co\_actors\_of\_co\_actors(co\_actor\_traverser->ptr->ptr->data.name, al, false);

                }

                else {

                    if (co\_actor\_traverser->ptr->ptr->data.name != name) {

                        cout << co\_actor\_traverser->ptr->ptr->data.name << endl;

                    }

                }

                co\_actor\_traverser = co\_actor\_traverser->next;

            }

            movie\_traverser = movie\_traverser->next;

        }

    }

    else {

        cout << "Actor Not Found" << endl;

    }

}

void co\_actors(string name, Actor\_List\* nl) {

    Actor\_Node\* actor\_traverser = nl->start;

    while (actor\_traverser != NULL && actor\_traverser->ptr->data.name != name) {

        actor\_traverser = actor\_traverser->next;

    }

    if (actor\_traverser != NULL && actor\_traverser->ptr->data.name == name) {

        movie\_node\_ptr\* movie\_traverser = actor\_traverser->movie\_ptr;

        while (movie\_traverser != NULL) {

            actor\_node\_ptr\* co\_actor\_traverser = movie\_traverser->m\_ptr->data.actor\_Start;

            cout << "Co\_Actors of actor " << name << " are : " << endl;

            while (co\_actor\_traverser != NULL) {

                if (co\_actor\_traverser->ptr->ptr->data.name != name)

                    cout << co\_actor\_traverser->ptr->ptr->data.name << endl;

                co\_actor\_traverser = co\_actor\_traverser->next;

            }

            movie\_traverser = movie\_traverser->next;

        }

    }

    else {

        cout << "Actor Not Found" << endl;

    }

}

//Movies of Director

void director\_movies(string name, Director\_List\* dl)

{

    Director\_Node\* d\_traverser = dl->start;

    while (d\_traverser != NULL) {

        if (d\_traverser->ptr->data.name == name)

            break;

        d\_traverser = d\_traverser->next;

    }

    if (d\_traverser != NULL) {

        movie\_node\_ptr\* movie\_ptr = d\_traverser->movie\_ptr;

        cout << name << " Movies : " << endl;

        while (movie\_ptr != NULL) {

            cout << "\t" << movie\_ptr->m\_ptr->data.title << endl;

            movie\_ptr = movie\_ptr->next;

        }

    }

    else {

        cout << "Director Not Found" << endl;

    }

}

//Movies with certain genre and director

void director\_movie\_genre(string genre,Genre\_List\* gl) {

    Genre\_Node\* g\_triv = gl->start;

    while (g\_triv != NULL)

    {

        if (g\_triv->genre == genre)

            break;

        g\_triv = g\_triv->next;

    }

    if (g\_triv != NULL)

    {

        movie\_node\_ptr\* m\_triv = g\_triv->movie\_ptr;

        while (m\_triv != NULL)

        {

            cout << m\_triv->m\_ptr->data.director << endl;

            m\_triv = m\_triv->next;

        }

        if (m\_triv == NULL)

            cout << "Director Not Found" << endl;

    }

    else {

        cout << "Genre Not Found" << endl;

    }

}

//Movies of certain genre

void genre\_movies(string genre, Genre\_List\* gl) {

    Genre\_Node\* g\_triv = gl->start;

    while (g\_triv != NULL)

    {

        if (g\_triv->genre == genre)

            break;

        g\_triv = g\_triv->next;

    }

    if (g\_triv != NULL)

    {

        movie\_node\_ptr\* m\_triv = g\_triv->movie\_ptr;

        while (m\_triv != NULL)

        {

            cout << m\_triv->m\_ptr->data.title << endl;

            m\_triv = m\_triv->next;

        }

    }

    else {

        cout << "Genre Not Found" << endl;

    }

}

int main() {

    bool menu = true;

    int option;

    string Genre = "";

    string Director = "";

    string Actor = "";

    Actor\_List al;

    MovieList ml;

    RatingsList rl;

    NameList n1;

    Director\_List dl;

    title\_principles\_Node\_List t1;

    Genre\_List gl;

    read(&ml, &rl, &t1, &n1, &al, &dl, &gl);

    cout << "\t\t\t\t\t\t----IMDB----" << endl;

    cout << "\n\t\t\t\t\t  1. Search Movie by Genre" << endl;

    cout << "\n\t\t\t\t\t  2. Search Director Movies" << endl;

    cout << "\n\t\t\t\t3. Search Director by his directed Movie's Genre" << endl;

    cout << "\n\t\t\t\t\t  4. Search Co-Actors of Co-Actors" << endl;

    cout << "\n\t\t\t\t\t  5. Search Co-Actors" << endl;

    cout << "\n\t\t\t\t\t\t  6. Exit" << endl;

    while (menu) {

        cout << "\nEnter Option : ";

        cin >> option;

        switch (option)

        {

        case 1:

            cout << "Enter Genre : ";

            cin >> Genre;

            genre\_movies(Genre, &gl);

            break;

        case 2:

            cout << "Enter Director Name : ";

            cin >> Director;

            director\_movies(Director, &dl);

            break;

        case 3:

            cout << "Enter Genre : ";

            cin >> Genre;

            director\_movie\_genre(Genre, &gl);

            break;

        case 4:

            cout << "Enter Actor Name : ";

            cin >> Actor;

            co\_actors(Actor, &al);

            break;

        case 5:

            cout << "Enter Actor Name : ";

            cin >> Actor;

            co\_actors\_of\_co\_actors(Actor, &al);

            break;

        default:

            menu = false;

            break;

        }

    }

    return 0;

}

# More Functions on AVLS:

#include<iostream>

#include<string>

#include<fstream>

#include<sstream>

using namespace std;

class actor\_movie\_node;

class movie\_actor\_node;

//-----------------Names-------------------Simple AVL

class Name\_Data

{

public:

    string name\_id;

    string name;

    string birth\_name;

    int height;

    string bio;

    string birth\_details;

    string date\_of\_birth;

    string place\_of\_birth;

    string death\_details;

    string date\_of\_death;

    string place\_of\_death;

    string reason\_of\_death;

    string spouses\_string;

    int spouses;

    int divorces;

    int spouses\_with\_children;

    int children;

    actor\_movie\_node\* moviestart;

};

class NameNode

{

public:

    Name\_Data data;

    NameNode\* lchild;

    NameNode\* rchild;

    int height1;

    NameNode(actor\_movie\_node\* moviestart, string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children);

};

NameNode::NameNode(actor\_movie\_node\* moviestart, string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children)

{

    data.moviestart = moviestart;

    data.name\_id = name\_id;

    data.name = name;

    data.birth\_name = birth\_name;

    data.bio = bio;

    data.birth\_details = birth\_details;

    data.death\_details = death\_details;

    data.place\_of\_birth = place\_of\_birth;

    data.place\_of\_death = place\_of\_death;

    data.reason\_of\_death = reason\_of\_death;

    data.spouses\_string = spouses\_string;

    int temp;

    stringstream geek(height);

    geek >> temp;

    data.height = temp;

    stringstream geek1(spouses);

    geek1 >> temp;

    data.spouses = temp;

    stringstream geek2(divorces);

    geek2 >> temp;

    data.divorces = temp;

    stringstream geek3(spouses\_with\_children);

    geek3 >> temp;

    data.spouses\_with\_children = temp;

    stringstream geek4(children);

    geek4 >> temp;

    data.children = temp;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class Name\_AVL

{

public:

    NameNode\* root;

    Name\_AVL()

    {

        root = NULL;

    }

    NameNode\* create\_node(string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children)

    {

        NameNode\* newnode = new NameNode(NULL, name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(NameNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(NameNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    NameNode\* rightrotate(NameNode\* y)

    {

        NameNode\* x = y->lchild;

        NameNode\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    NameNode\* leftrotate(NameNode\* x)

    {

        NameNode\* y = x->rchild;

        NameNode\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    NameNode\* insert(NameNode\* n, string name\_id, string name, string birth\_name, string height, string bio, string birth\_details, string date\_of\_birth, string place\_of\_birth, string death\_details, string date\_of\_death, string place\_of\_death, string reason\_of\_death, string spouses\_string, string spouses, string divorces, string spouses\_with\_children, string children)

    {

        if (n == NULL)

        {

            return create\_node(name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

        }

        if (name\_id < n->data.name\_id)

        {

            n->lchild = insert(n->lchild, name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

        }

        else if (name\_id > n->data.name\_id)

        {

            n->rchild = insert(n->rchild, name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && name\_id < n->lchild->data.name\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && name\_id > n->rchild->data.name\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && name\_id > n->lchild->data.name\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && name\_id < n->rchild->data.name\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(NameNode\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //print node

        cout << temp->data.spouses << endl;

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

//-----------------Movies-------------------

class MovieData {

public:

    string title\_id;

    string title;

    string original\_title;

    int year;

    string date;

    string genre;

    int duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    string actor\_Start;

    string description;

    double age\_vote;

    int votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    int meta\_score;

    int reviews\_from\_user;

    int reviews\_from\_critics;

};

class MovieNode {

public:

    MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics);

    MovieData data;

    MovieNode\* lchild;

    MovieNode\* rchild;

    int height1;

};

MovieNode::MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics) {

    int temp;

    data.title\_id = title\_id;

    data.title = title;

    data.original\_title = original\_title;

    stringstream geek(year);

    geek >> temp;

    data.year = temp;

    data.date = date;

    data.genre = genre;

    stringstream geek1(duration);

    geek >> temp;

    data.duration = temp;

    data.country = country;

    data.language = language;

    data.director = director;

    data.writer = writer;

    data.production\_company = production\_company;

    data.actor\_Start = actor\_start;

    data.description = description;

    stringstream geek2(age\_vote);

    geek >> temp;

    data.age\_vote = double(temp);

    stringstream geek6(age\_vote);

    geek >> temp;

    data.votes = temp;

    data.budget = budget;

    data.usa\_gross\_income = usa\_gross\_income;

    stringstream geek3(meta\_score);

    geek >> temp;

    data.meta\_score = temp;

    data.worldwide\_gross\_income = worldwide\_gross\_income;

    stringstream geek4(reviews\_from\_user);

    geek >> temp;

    data.reviews\_from\_user = temp;

    stringstream geek5(reviews\_from\_critics);

    geek >> temp;

    data.reviews\_from\_critics = temp;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class Movie\_AVL

{

public:

    MovieNode\* root;

    Movie\_AVL()

    {

        root = NULL;

    }

    MovieNode\* create\_node(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_Start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        MovieNode\* newnode = new MovieNode(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_Start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(MovieNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(MovieNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    MovieNode\* rightrotate(MovieNode\* y)

    {

        MovieNode\* x = y->lchild;

        MovieNode\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    MovieNode\* leftrotate(MovieNode\* x)

    {

        MovieNode\* y = x->rchild;

        MovieNode\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    MovieNode\* insert(MovieNode\* n, string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actor\_start, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        if (n == NULL)

        {

            return create\_node(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        if (title\_id < n->data.title\_id)

        {

            n->lchild = insert(n->lchild, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else if (title\_id > n->data.title\_id)

        {

            n->rchild = insert(n->rchild, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actor\_start, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && title\_id < n->lchild->data.title\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && title\_id > n->rchild->data.title\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && title\_id > n->lchild->data.title\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && title\_id < n->rchild->data.title\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(MovieNode\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //print node

        cout << temp->data.title << endl;

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

//-----------------ratings-------------------------

class Ratings\_data

{

public:

    string imdb\_title\_id;

    double weighted\_average\_vote;

    int total\_votes;

    double mean\_vote;

    int median\_vote;

    int votes\_10;

    int votes\_9;

    int votes\_8;

    int votes\_7;

    int votes\_6;

    int votes\_5;

    int votes\_4;

    int votes\_3;

    int votes\_2;

    int votes\_1;

    double allgenders\_0age\_avg\_vote;

    int allgenders\_0age\_votes;

    double allgenders\_18age\_avg\_vote;

    int allgenders\_18age\_votes;

    double allgenders\_30age\_avg\_vote;

    int allgenders\_30age\_votes;

    double allgenders\_45age\_avg\_vote;

    int allgenders\_45age\_votes;

    double males\_allages\_avg\_vote;

    int males\_allages\_votes;

    double males\_0age\_avg\_vote;

    int males\_0age\_votes;

    double males\_18age\_avg\_vote;

    int males\_18age\_votes;

    double males\_30age\_avg\_vote;

    int males\_30age\_votes;

    double males\_45age\_avg\_vote;

    int males\_45age\_votes;

    double females\_allages\_avg\_vote;

    int females\_allages\_votes;

    double females\_0age\_avg\_vote;

    int females\_0age\_votes;

    double females\_18age\_avg\_vote;

    int females\_18age\_votes;

    double females\_30age\_avg\_vote;

    int females\_30age\_votes;

    double females\_45age\_avg\_vote;

    int females\_45age\_votes;

    double top1000\_voters\_rating;

    int top1000\_voters\_votes;

    double us\_voters\_rating;

    int us\_voters\_votes;

    double non\_us\_voters\_rating;

    int non\_us\_voters\_votes;

};

class RatingsNode {

public:

    RatingsNode(string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string, string);

    Ratings\_data data;

    RatingsNode\* lchild;

    RatingsNode\* rchild;

    int height1;

};

RatingsNode::RatingsNode(string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes) {

    int temp;

    stringstream geek(allgenders\_0age\_avg\_vote);

    geek >> temp;

    data.allgenders\_0age\_avg\_vote = temp;

    stringstream geek1(allgenders\_0age\_votes);

    geek1 >> temp;

    data.allgenders\_0age\_votes = temp;

    stringstream geek2(allgenders\_18age\_avg\_vote);

    geek2 >> temp;

    data.allgenders\_18age\_avg\_vote = temp;

    stringstream geek3(allgenders\_18age\_votes);

    geek3 >> temp;

    data.allgenders\_18age\_votes = temp;

    stringstream geek4(allgenders\_30age\_avg\_vote);

    geek4 >> temp;

    data.allgenders\_30age\_avg\_vote = temp;

    stringstream geek5(allgenders\_30age\_votes);

    geek5 >> temp;

    data.allgenders\_30age\_votes = temp;

    stringstream geek6(allgenders\_45age\_avg\_vote);

    geek6 >> temp;

    data.allgenders\_45age\_avg\_vote = temp;

    stringstream geek7(allgenders\_45age\_votes);

    geek7 >> temp;

    data.allgenders\_45age\_votes = temp;

    stringstream geek8(females\_0age\_avg\_vote);

    geek8 >> temp;

    data.females\_0age\_avg\_vote = temp;

    stringstream geek9(females\_0age\_votes);

    geek9 >> temp;

    data.females\_0age\_votes = temp;

    stringstream geek10(females\_0age\_avg\_vote);

    geek10 >> temp;

    data.females\_18age\_avg\_vote = temp;

    stringstream geek11(females\_18age\_votes);

    geek11 >> temp;

    data.females\_18age\_votes = temp;

    stringstream geek12(females\_30age\_avg\_vote);

    geek12 >> temp;

    data.females\_30age\_avg\_vote = temp;

    stringstream geek13(females\_30age\_votes);

    geek13 >> temp;

    data.females\_30age\_votes = temp;

    stringstream geek14(females\_45age\_avg\_vote);

    geek14 >> temp;

    data.females\_45age\_avg\_vote = temp;

    stringstream geek15(females\_45age\_votes);

    geek15 >> temp;

    data.females\_45age\_votes = temp;

    stringstream geek16(females\_allages\_avg\_vote);

    geek16 >> temp;

    data.females\_allages\_avg\_vote = temp;

    stringstream geek17(females\_allages\_votes);

    geek17 >> temp;

    data.females\_allages\_votes = temp;

    stringstream geek18(males\_0age\_avg\_vote);

    geek18 >> temp;

    data.males\_0age\_avg\_vote = temp;

    stringstream geek19(males\_0age\_votes);

    geek19 >> temp;

    data.males\_0age\_votes = temp;

    stringstream geek20(males\_0age\_avg\_vote);

    geek20 >> temp;

    data.males\_18age\_avg\_vote = temp;

    stringstream geek21(males\_18age\_votes);

    geek21 >> temp;

    data.males\_18age\_votes = temp;

    stringstream geek22(males\_30age\_avg\_vote);

    geek22 >> temp;

    data.males\_30age\_avg\_vote = temp;

    stringstream geek23(males\_30age\_votes);

    geek23 >> temp;

    data.males\_30age\_votes = temp;

    stringstream geek24(males\_45age\_avg\_vote);

    geek24 >> temp;

    data.males\_45age\_avg\_vote = temp;

    stringstream geek25(males\_45age\_votes);

    geek25 >> temp;

    data.males\_45age\_votes = temp;

    stringstream geek26(males\_allages\_avg\_vote);

    geek26 >> temp;

    data.males\_allages\_avg\_vote = temp;

    stringstream geek27(males\_allages\_votes);

    geek27 >> temp;

    data.males\_allages\_votes = temp;

    stringstream geek28(imdb\_title\_id);

    geek28 >> temp;

    data.imdb\_title\_id = temp;

    stringstream geek29(mean\_vote);

    geek29 >> temp;

    data.mean\_vote = temp;

    stringstream geek30(median\_vote);

    geek30 >> temp;

    data.median\_vote = temp;

    stringstream geek31(non\_us\_voters\_rating);

    geek31 >> temp;

    data.non\_us\_voters\_rating = temp;

    stringstream geek32(non\_us\_voters\_votes);

    geek32 >> temp;

    data.non\_us\_voters\_votes = temp;

    stringstream geek33(top1000\_voters\_rating);

    geek33 >> temp;

    data.top1000\_voters\_rating = temp;

    stringstream geek34(top1000\_voters\_votes);

    geek34 >> temp;

    data.top1000\_voters\_votes = temp;

    stringstream geek35(total\_votes);

    geek35 >> temp;

    data.total\_votes = temp;

    stringstream geek36(us\_voters\_rating);

    geek36 >> temp;

    data.us\_voters\_rating = temp;

    stringstream geek37(us\_voters\_votes);

    geek37 >> temp;

    data.us\_voters\_votes = temp;

    stringstream geek38(votes\_1);

    geek38 >> temp;

    data.votes\_1 = temp;

    stringstream geek39(votes\_10);

    geek39 >> temp;

    data.votes\_10 = temp;

    stringstream geek40(votes\_2);

    geek40 >> temp;

    data.votes\_2 = temp;

    stringstream geek41(votes\_3);

    geek41 >> temp;

    data.votes\_3 = temp;

    stringstream geek42(votes\_4);

    geek42 >> temp;

    data.votes\_4 = temp;

    stringstream geek43(votes\_5);

    geek43 >> temp;

    data.votes\_5 = temp;

    stringstream geek44(votes\_6);

    geek44 >> temp;

    data.votes\_6 = temp;

    stringstream geek45(votes\_7);

    geek45 >> temp;

    data.votes\_7 = temp;

    stringstream geek46(votes\_8);

    geek46 >> temp;

    data.votes\_8 = temp;

    stringstream geek47(votes\_9);

    geek47 >> temp;

    data.votes\_9 = temp;

    stringstream geek48(weighted\_average\_vote);

    geek48 >> temp;

    data.weighted\_average\_vote = temp;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class Ratings\_AVL

{

public:

    RatingsNode\* root;

    Ratings\_AVL()

    {

        root = NULL;

    }

    RatingsNode\* create\_node(string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes)

    {

        RatingsNode\* newnode = new RatingsNode(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(RatingsNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(RatingsNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    RatingsNode\* rightrotate(RatingsNode\* y)

    {

        RatingsNode\* x = y->lchild;

        RatingsNode\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    RatingsNode\* leftrotate(RatingsNode\* x)

    {

        RatingsNode\* y = x->rchild;

        RatingsNode\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    RatingsNode\* insert(RatingsNode\* n, string imdb\_title\_id, string weighted\_average\_vote, string  total\_votes, string mean\_vote, string median\_vote, string votes\_10, string votes\_9, string votes\_8, string votes\_7, string votes\_6, string votes\_5, string votes\_4, string votes\_3, string votes\_2, string votes\_1, string allgenders\_0age\_avg\_vote, string allgenders\_0age\_votes, string allgenders\_18age\_avg\_vote, string allgenders\_18age\_votes, string allgenders\_30age\_avg\_vote, string allgenders\_30age\_votes, string allgenders\_45age\_avg\_vote, string allgenders\_45age\_votes, string males\_allages\_avg\_vote, string males\_allages\_votes, string males\_0age\_avg\_vote, string males\_0age\_votes, string males\_18age\_avg\_vote, string males\_18age\_votes, string males\_30age\_avg\_vote, string males\_30age\_votes, string males\_45age\_avg\_vote, string males\_45age\_votes, string females\_allages\_avg\_vote, string females\_allages\_votes, string females\_0age\_avg\_vote, string females\_0age\_votes, string females\_18age\_avg\_vote, string females\_18age\_votes, string females\_30age\_avg\_vote, string females\_30age\_votes, string females\_45age\_avg\_vote, string females\_45age\_votes, string top1000\_voters\_rating, string top1000\_voters\_votes, string us\_voters\_rating, string us\_voters\_votes, string non\_us\_voters\_rating, string non\_us\_voters\_votes)

    {

        if (n == NULL)

        {

            return create\_node(imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

        }

        if (imdb\_title\_id < n->data.imdb\_title\_id)

        {

            n->lchild = insert(n->lchild, imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

        }

        else if (imdb\_title\_id > n->data.imdb\_title\_id)

        {

            n->rchild = insert(n->rchild, imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && imdb\_title\_id < n->lchild->data.imdb\_title\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && imdb\_title\_id > n->rchild->data.imdb\_title\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && imdb\_title\_id > n->lchild->data.imdb\_title\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && imdb\_title\_id < n->rchild->data.imdb\_title\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(RatingsNode\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //print node

        cout << temp->data.imdb\_title\_id << endl;

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

//-----------------title\_principles----------------

class title\_principles\_Data {

public:

    string imdb\_title\_id;

    int order;

    string imdb\_name\_id;

    string category;

    string job;

    string characters;

};

class title\_principles\_Node {

public:

    title\_principles\_Node(string, string, string, string, string, string);

    title\_principles\_Data data;

    title\_principles\_Node\* lchild;

    title\_principles\_Node\* rchild;

    int height1;

};

title\_principles\_Node::title\_principles\_Node(string imdb\_title\_id, string order, string imdb\_name\_id, string category, string job, string characters) {

    int temp;

    data.imdb\_title\_id = imdb\_title\_id;

    stringstream geek(order);

    geek >> temp;

    data.order = temp;

    data.imdb\_name\_id = imdb\_name\_id;

    data.category = category;

    data.job = job;

    data.characters = characters;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class title\_principles\_AVL

{

public:

    title\_principles\_Node\* root;

    title\_principles\_AVL()

    {

        root = NULL;

    }

    title\_principles\_Node\* create\_node(string imdb\_title\_id, string order, string imdb\_name\_id, string category, string job, string characters)

    {

        title\_principles\_Node\* newnode = new title\_principles\_Node(imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(title\_principles\_Node\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(title\_principles\_Node\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    title\_principles\_Node\* rightrotate(title\_principles\_Node\* y)

    {

        title\_principles\_Node\* x = y->lchild;

        title\_principles\_Node\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    title\_principles\_Node\* leftrotate(title\_principles\_Node\* x)

    {

        title\_principles\_Node\* y = x->rchild;

        title\_principles\_Node\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    title\_principles\_Node\* insert(title\_principles\_Node\* n, string imdb\_title\_id, string order, string imdb\_name\_id, string category, string job, string characters)

    {

        if (n == NULL)

        {

            return create\_node(imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

        }

        if (imdb\_name\_id < n->data.imdb\_name\_id)

        {

            n->lchild = insert(n->lchild, imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

        }

        else if (imdb\_name\_id > n->data.imdb\_name\_id)

        {

            n->rchild = insert(n->rchild, imdb\_title\_id, order, imdb\_name\_id, category, job, characters);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && imdb\_name\_id < n->lchild->data.imdb\_name\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && imdb\_name\_id > n->rchild->data.imdb\_name\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && imdb\_name\_id > n->lchild->data.imdb\_name\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && imdb\_title\_id < n->rchild->data.imdb\_title\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(title\_principles\_Node\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //print node

        cout << temp->data.category << endl;

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

//----------------------------------------------------------------------------------------------

//----------------------------------------------------------------------------------------------

class movie\_actor\_node

{

public:

    NameNode\* points\_to\_actor;

    movie\_actor\_node\* rch;

    movie\_actor\_node\* lch;

    int height;

    string key\_tits\_id;

    movie\_actor\_node()

    {

        rch = NULL;

        lch = NULL;

    }

};

class movie\_actor\_avl

{

public:

    int max(int a, int b)

    {

        return a > b ? a : b;

    }

    int height(movie\_actor\_node\* temp)

    {

        if (temp == NULL)

            return 0;

        return temp->height;

    }

    int calc\_balance\_spectrum(movie\_actor\_node\* temp)

    {

        return height(temp->rch) - height(temp->lch);

    }

    movie\_actor\_node\* create\_newnode(string val)

    {

        movie\_actor\_node\* newnode = new movie\_actor\_node();//try not passing value or building constructor for newnode

        newnode->lch = NULL;

        newnode->rch = NULL;

        newnode->height = 1;

        newnode->key\_tits\_id = val; /\*cout << newnode->key\_tits\_id;\*/

        return newnode;

    }

    movie\_actor\_node\* left\_rotation(movie\_actor\_node\* temp)

    {

        movie\_actor\_node\* y = temp;

        movie\_actor\_node\* x = temp->rch;

        y->rch = x->lch;

        x->lch = y;

        y->height = max(height(y->rch), height(y->lch)) + 1;

        x->height = max(height(x->rch), height(x->lch)) + 1;

        return x;

    }

    movie\_actor\_node\* right\_rotation(movie\_actor\_node\* temp)

    {

        movie\_actor\_node\* y = temp;

        movie\_actor\_node\* x = temp->lch;

        y->lch = x->rch;

        x->rch = y;

        y->height = max(height(y->rch), height(y->lch)) + 1;

        x->height = max(height(x->rch), height(x->lch)) + 1;

        return x;

    }

    movie\_actor\_node\* insertnode(string val, movie\_actor\_node\* temp)

    {

        if (temp == NULL) {

            /\*cout << "1\n";\*/

            return create\_newnode(val);

        }

        if (val > temp->key\_tits\_id)

            temp->rch = insertnode(val, temp->rch);

        else if (val < temp->key\_tits\_id)

            temp->lch = insertnode(val, temp->lch);

        else {

            cout << "duplicate value input is not allowed\n";

            return temp;

        }

        temp->height = max(height(temp->rch), height(temp->lch)) + 1;

        int bs = calc\_balance\_spectrum(temp);

        if (bs > 1 && val > temp->rch->key\_tits\_id)

        {

            return left\_rotation(temp);

        }

        if (bs > 1 && val < temp->rch->key\_tits\_id)

        {

            temp->rch = right\_rotation(temp->rch);

            return left\_rotation(temp);

        }

        if (bs < -1 && val < temp->lch->key\_tits\_id)

        {

            return right\_rotation(temp);

        }

        if (bs < -1 && val < temp->lch->key\_tits\_id)

        {

            temp->lch = left\_rotation(temp->lch);

            return right\_rotation(temp);

        }

        return temp;

    }

};

void find\_actor\_in\_ma\_avl(movie\_actor\_node\* root, NameNode\* name)

{

    if (root == NULL)

        return;

    if (root->key\_tits\_id == name->data.name)

        root->points\_to\_actor = name;

    find\_actor\_in\_ma\_avl(root->rch, name);

    find\_actor\_in\_ma\_avl(root->lch, name);

}

bool find\_actor\_in\_names(NameNode\* traverser, movie\_actor\_node\* mavl, string token)

{

    if (traverser == NULL) {

        //cout << "actor notfound\n";

        return false;

    }

    find\_actor\_in\_names(traverser->rchild, mavl, token);

    find\_actor\_in\_names(traverser->lchild, mavl, token);

    if (traverser->data.name == token)

    {

        find\_actor\_in\_ma\_avl(mavl, traverser);

        return true;

    }

}

//----------------------------------------------------------------------------------------------

//----------------------------------------------------------------------------------------------

class actor\_movie\_node

{

public:

    MovieNode\* points\_to\_movie;

    actor\_movie\_node\* rch;

    actor\_movie\_node\* lch;

    int height;

    string key\_name\_id;

};

class actor\_movie\_avl

{

public:

    actor\_movie\_node\* root = NULL;

    actor\_movie\_node\* loc = NULL;

    actor\_movie\_node\* ploc = NULL;

    //actor\_movie\_node\* insertnode(string, actor\_movie\_node\*);

    //void printtree(actor\_movie\_node\*);

    int max(int a, int b)

    {

        return a > b ? a : b;

    }

    int height(actor\_movie\_node\* temp)

    {

        if (temp == NULL)

            return 0;

        return temp->height;

    }

    int calc\_balance\_spectrum(actor\_movie\_node\* temp)

    {

        return height(temp->rch) - height(temp->lch);

    }

    actor\_movie\_node\* create\_newnode(string val, MovieNode\* temp)

    {

        actor\_movie\_node\* newnode = new actor\_movie\_node;

        newnode->lch = NULL;

        newnode->rch = NULL;

        newnode->height = 1;

        newnode->points\_to\_movie = temp;

        newnode->key\_name\_id = val;

        return newnode;

    }

    int count = 0;//test

    actor\_movie\_node\* left\_rotation(actor\_movie\_node\* temp)

    {

        actor\_movie\_node\* y = temp;

        actor\_movie\_node\* x = temp->rch;

        y->rch = x->lch;

        x->lch = y;

        y->height = max(height(y->rch), height(y->lch)) + 1;

        x->height = max(height(x->rch), height(x->lch)) + 1;

        return x;

    }

    actor\_movie\_node\* right\_rotation(actor\_movie\_node\* temp)

    {

        actor\_movie\_node\* y = temp;

        actor\_movie\_node\* x = temp->lch;

        y->lch = x->rch;

        x->rch = y;

        y->height = max(height(y->rch), height(y->lch)) + 1;

        x->height = max(height(x->rch), height(x->lch)) + 1;

        return x;

    }

    actor\_movie\_node\* insertnode(string val, actor\_movie\_node\* temp, MovieNode\* movie)

    {

        if (temp == NULL || temp->key\_name\_id == val) {

            //cout << count++;//test

            /\*cout << "inserted\n";\*/

            return create\_newnode(val, movie);

        }

        if (val > temp->key\_name\_id)

            temp->rch = insertnode(val, temp->rch, movie);

        else if (val < temp->key\_name\_id)

            temp->lch = insertnode(val, temp->lch, movie);

        //else {

        //

        //  cout << "duplicate value input is not allowed in actor\_movie avl\n";

        //  return temp;

        //}

        temp->height = max(height(temp->rch), height(temp->lch)) + 1;

        int bs = calc\_balance\_spectrum(temp);

        if (bs > 1 && val > temp->rch->key\_name\_id)

        {

            return left\_rotation(temp);

        }

        if (bs > 1 && val < temp->rch->key\_name\_id)

        {

            temp->rch = right\_rotation(temp->rch);

            return left\_rotation(temp);

        }

        if (bs < -1 && val < temp->lch->key\_name\_id)

        {

            return right\_rotation(temp);

        }

        if (bs < -1 && val < temp->lch->key\_name\_id)

        {

            temp->lch = left\_rotation(temp->lch);

            return right\_rotation(temp);

        }

        return temp;

    }

};

void read(Movie\_AVL\* ml, Ratings\_AVL\* rl, title\_principles\_AVL\* t1, Name\_AVL\* t2) {

    //-----------------name Storage----------------

    int count = 0;

    string name\_id;

    string name;

    string birth\_name;

    string height;

    string bio;

    string birth\_details;

    string date\_of\_birth;

    string place\_of\_birth;

    string death\_details;

    string date\_of\_death; string place\_of\_death; string reason\_of\_death; string spouses\_string; string spouses; string divorces; string spouses\_with\_children; string children;

    actor\_movie\_node\* points\_to\_movie;

    ifstream File3("names.csv");

    if (File3.is\_open()) {

        while (!File3.eof()) {

            /\*cout << "read1\n"; cout << "read11\n";\*/

            if (count == 0) {

                string temp;

                getline(File3, temp);

                count = 1;

            }

            else {

                getline(File3, name\_id, ',');

                getline(File3, name, ',');

                getline(File3, birth\_name, ',');

                getline(File3, height, ',');

                getline(File3, bio, ',');

                getline(File3, birth\_details, ',');

                getline(File3, date\_of\_birth, ',');

                getline(File3, place\_of\_birth, ',');

                getline(File3, death\_details, ',');

                getline(File3, date\_of\_death, ',');

                getline(File3, place\_of\_death, ',');

                getline(File3, reason\_of\_death, ',');

                getline(File3, spouses\_string, ',');

                getline(File3, spouses, ',');

                getline(File3, divorces, ',');

                getline(File3, spouses\_with\_children, ',');

                getline(File3, children);

                t2->root = t2->insert(t2->root, name\_id, name, birth\_name, height, bio, birth\_details, date\_of\_birth, place\_of\_birth, death\_details, date\_of\_death, place\_of\_death, reason\_of\_death, spouses\_string, spouses, divorces, spouses\_with\_children, children);

            }

        }

    }

    else {

        cout << "Give Error3";

    }

    //---------Movie List Storage------------

    count = 0;

    string title\_id;

    string title;

    string original\_title;

    string year;

    string date;

    string genre;

    string duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    string actors;

    string description;

    string age\_vote;

    string votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    string meta\_score;

    string reviews\_from\_user;

    string reviews\_from\_critics;

    ifstream File("movies.csv");

    movie\_actor\_avl m\_a\_avl;

    if (File.is\_open()) {

        while (!File.eof()) {

            /\*cout << "read2\n"; cout << "read22\n";\*/

            if (count == 0) {

                string temp;

                getline(File, temp);

                count = 1;

            }

            else {

                getline(File, title\_id, ',');

                getline(File, title, ',');

                getline(File, original\_title, ',');

                getline(File, year, ',');

                getline(File, date, ',');

                getline(File, genre, ',');

                getline(File, duration, ',');

                getline(File, country, ',');

                getline(File, language, ',');

                getline(File, director, ',');

                getline(File, writer, ',');

                getline(File, production\_company, ',');

                getline(File, actors, ',');

                getline(File, description, ',');

                getline(File, age\_vote, ',');

                getline(File, votes, ',');

                getline(File, budget, ',');

                getline(File, usa\_gross\_income, ',');

                getline(File, meta\_score, ',');

                getline(File, reviews\_from\_user, ',');

                getline(File, reviews\_from\_critics);

                ml->root = ml->insert(ml->root, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

            }

        }

    }

    else {

        cout << "Give Error\n" << endl;

    }

    //---------Rating List Storage------------

    count = 0;

    string imdb\_title\_id;

    string weighted\_average\_vote;

    string  total\_votes;

    string mean\_vote;

    string median\_vote;

    string votes\_10;

    string votes\_9;

    string votes\_8;

    string votes\_7;

    string  votes\_6;

    string  votes\_5;

    string  votes\_4;

    string  votes\_3;

    string  votes\_2;

    string  votes\_1;

    string allgenders\_0age\_avg\_vote;

    string allgenders\_0age\_votes;

    string allgenders\_18age\_avg\_vote;

    string allgenders\_18age\_votes;

    string allgenders\_30age\_avg\_vote;

    string allgenders\_30age\_votes;

    string allgenders\_45age\_avg\_vote;

    string allgenders\_45age\_votes;

    string males\_allages\_avg\_vote;

    string males\_allages\_votes;

    string males\_0age\_avg\_vote;

    string males\_0age\_votes;

    string males\_18age\_avg\_vote;

    string males\_18age\_votes;

    string males\_30age\_avg\_vote;

    string males\_30age\_votes;

    string males\_45age\_avg\_vote;

    string males\_45age\_votes;

    string females\_allages\_avg\_vote;

    string females\_allages\_votes;

    string females\_0age\_avg\_vote;

    string females\_0age\_votes;

    string females\_18age\_avg\_vote;

    string females\_18age\_votes;

    string females\_30age\_avg\_vote;

    string females\_30age\_votes;

    string females\_45age\_avg\_vote;

    string females\_45age\_votes;

    string top1000\_voters\_rating;

    string top1000\_voters\_votes;

    string us\_voters\_rating;

    string us\_voters\_votes;

    string non\_us\_voters\_rating;

    string non\_us\_voters\_votes;

    ifstream File1("Ratings.csv");

    if (File1.is\_open()) {

        while (!File1.eof()) {

            /\*cout << "read3\n"; cout << "read33\n";\*/

            if (count == 0) {

                string temp;

                getline(File1, temp);

                count = 1;

            }

            else {

                getline(File1, imdb\_title\_id, ',');

                getline(File1, weighted\_average\_vote, ',');

                getline(File1, total\_votes, ',');

                getline(File1, mean\_vote, ',');

                getline(File1, median\_vote, ',');

                getline(File1, votes\_10, ',');

                getline(File1, votes\_9, ',');

                getline(File1, votes\_8, ',');

                getline(File1, votes\_7, ',');

                getline(File1, votes\_6, ',');

                getline(File1, votes\_5, ',');

                getline(File1, votes\_4, ',');

                getline(File1, votes\_3, ',');

                getline(File1, votes\_2, ',');

                getline(File1, votes\_1, ',');

                getline(File1, votes\_9, ',');

                getline(File1, allgenders\_0age\_avg\_vote, ',');

                getline(File1, allgenders\_0age\_votes, ',');

                getline(File1, allgenders\_18age\_avg\_vote, ',');

                getline(File1, allgenders\_18age\_votes, ',');

                getline(File1, allgenders\_18age\_votes, ',');

                getline(File1, allgenders\_30age\_avg\_vote, ',');

                getline(File1, allgenders\_30age\_avg\_vote, ',');

                getline(File1, allgenders\_45age\_avg\_vote, ',');

                getline(File1, allgenders\_45age\_avg\_vote, ',');

                getline(File1, females\_allages\_avg\_vote, ',');

                getline(File1, females\_allages\_votes, ',');

                getline(File1, females\_0age\_avg\_vote, ',');

                getline(File1, females\_0age\_votes, ',');

                getline(File1, females\_18age\_avg\_vote, ',');

                getline(File1, females\_18age\_votes, ',');

                getline(File1, females\_30age\_avg\_vote, ',');

                getline(File1, females\_30age\_votes, ',');

                getline(File1, females\_45age\_avg\_vote, ',');

                getline(File1, top1000\_voters\_rating, ',');

                getline(File1, top1000\_voters\_votes, ',');

                getline(File1, us\_voters\_rating, ',');

                getline(File1, us\_voters\_votes, ',');

                getline(File1, non\_us\_voters\_rating, ',');

                getline(File1, non\_us\_voters\_votes);

                rl->root = rl->insert(rl->root, imdb\_title\_id, weighted\_average\_vote, total\_votes, mean\_vote, median\_vote, votes\_10, votes\_9, votes\_8, votes\_7, votes\_6, votes\_5, votes\_4, votes\_3, votes\_2, votes\_1, allgenders\_0age\_avg\_vote, allgenders\_0age\_votes, allgenders\_18age\_avg\_vote, allgenders\_18age\_votes, allgenders\_30age\_avg\_vote, allgenders\_30age\_votes, allgenders\_45age\_avg\_vote, allgenders\_45age\_votes, males\_allages\_avg\_vote, males\_allages\_votes, males\_0age\_avg\_vote, males\_0age\_votes, males\_18age\_avg\_vote, males\_18age\_votes, males\_30age\_avg\_vote, males\_30age\_votes, males\_45age\_avg\_vote, males\_45age\_votes, females\_allages\_avg\_vote, females\_allages\_votes, females\_0age\_avg\_vote, females\_0age\_votes, females\_18age\_avg\_vote, females\_18age\_votes, females\_30age\_avg\_vote, females\_30age\_votes, females\_45age\_avg\_vote, females\_45age\_votes, top1000\_voters\_rating, top1000\_voters\_votes, us\_voters\_rating, us\_voters\_votes, non\_us\_voters\_rating, non\_us\_voters\_votes);

            }

        }

    }

    else {

        cout << "Give Error1\n";

    }

    //-----------------title\_principles Storage----------------

    count = 0;

    string imdb\_title\_id1;

    string order;

    string imdb\_name\_id;

    string category;

    string job;

    string characters;

    ifstream File2("title\_principals.csv");

    if (File2.is\_open()) {

        while (!File2.eof()) {

            /\*cout << "read4\n"; cout << "read44\n";\*/

            if (count == 0) {

                string temp;

                getline(File2, temp);

                count = 1;

            }

            else {

                getline(File2, imdb\_title\_id1, ',');

                getline(File2, order, ',');

                getline(File2, imdb\_name\_id, ',');

                getline(File2, category, ',');

                getline(File2, job, ',');

                getline(File2, characters);

                t1->root = t1->insert(t1->root, imdb\_title\_id1, order, imdb\_name\_id, category, job, characters);

            }

        }

    }

    else {

        cout << "Give Error2";

    }

}

//-------------------------------------------------------------------------------------------

//-------------------------------------------------------------------------------------------

void movie\_country(string name, MovieNode\* nl) {

    if (nl == NULL)

        return;

    if (nl->data.title == name)

    {

        cout << nl->data.country << endl;

    }

    else if (nl->data.title > name)

        movie\_country(name, nl->lchild);

    else

        movie\_country(name, nl->rchild);

    return;

}

//-------------------------------------------------------------------------------------------

//-------------------------------------------------------------------------------------------

void movie\_rates(MovieNode\* nl, RatingsNode\* rl, int votes)

{

    if (rl == NULL)

        return;

    if (rl->data.total\_votes >= votes)

        cout << nl->data.title << endl;

    else if (rl->data.imdb\_title\_id > nl->data.title\_id)

        movie\_rates(nl, rl->lchild, votes);

    else

        movie\_rates(nl, rl->rchild, votes);

}

void movie\_ratings(MovieNode\* nl, RatingsNode\* rl, int votes) {//output all movies with votes above given total votes

    if (nl == NULL)

        return;

    movie\_rates(nl, rl, votes);

    movie\_ratings(nl->rchild, rl, votes);

    movie\_ratings(nl->lchild, rl, votes);

}

//-------------------------------------------------------------------------------------------

//-------------------------------------------------------------------------------------------

void traverse\_movie\_node(string name, title\_principles\_Node\* roottitle, MovieNode\* rootmovie)

{

    if (rootmovie == NULL)

        return;

    if (rootmovie->data.title\_id == roottitle->data.imdb\_title\_id)

    {

        if (rootmovie->data.director == name)

            cout << "The movie for entered director is : " << rootmovie->data.title;

    }

    else if (rootmovie->data.title\_id > roottitle->data.imdb\_title\_id)

        traverse\_movie\_node(name, roottitle, rootmovie->lchild);

    else

        traverse\_movie\_node(name, roottitle, rootmovie->rchild);

    return;

}

void traverse\_titleprincipals(string name, NameNode\* rootname, title\_principles\_Node\* roottitle, MovieNode\* rootmovie)

{

    if (roottitle == NULL)

        return;

    if (rootname->data.name\_id == roottitle->data.imdb\_name\_id)

    {

        traverse\_movie\_node(name, roottitle, rootmovie);

    }

    else if (rootname->data.name\_id > roottitle->data.imdb\_name\_id)

        traverse\_titleprincipals(name, rootname, roottitle->rchild, rootmovie);

    else

        traverse\_titleprincipals(name, rootname, roottitle->lchild, rootmovie);

    return;

}

void director\_movies(string name, NameNode\* rootname, title\_principles\_Node\* roottitle, MovieNode\* rootmovie)

{

    if (rootname == NULL)

        return;

    if (name == rootname->data.name)

    {

        traverse\_titleprincipals(name, rootname, roottitle, rootmovie);

    }

    director\_movies(name, rootname->rchild, roottitle, rootmovie);

    director\_movies(name, rootname->lchild, roottitle, rootmovie);

}

void main() {

    Movie\_AVL ml;

    Ratings\_AVL rl;

    title\_principles\_AVL t1;

    Name\_AVL n1;

    read(&ml, &rl, &t1, &n1);

    cout << "The country for movie input is : \n\n";

    movie\_country("L'Inferno", ml.root);

    cout << "\n\nMovie with votes more than input are : \n\n" << endl;

    movie\_ratings(ml.root, rl.root, 150);

    cout << "\n\nDirector movies : " << endl;

    director\_movies("Alexander Black", n1.root, t1.root, ml.root);

    getchar();

    getchar();

}



# Heap:

#include<iostream>

#include<string>

#include<string.h>

#include<fstream>

#include<sstream>

using namespace std;//heap

class MovieData {

public:

    string title\_id;

    string title;

    string original\_title;

    int year;

    string date;

    string genre;

    int duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    string actors;

    string description;

    double age\_vote;

    int votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    int meta\_score;

    int reviews\_from\_user;

    int reviews\_from\_critics;

};

class MovieNode {

public:

    MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actors, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics);

    MovieData data;

    MovieNode\* lchild;

    MovieNode\* rchild;

    int height1;

};

MovieNode::MovieNode(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actors, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics) {

    int temp;

    data.title\_id = title\_id;

    data.title = title;

    data.original\_title = original\_title;

    stringstream geek(year);

    geek >> temp;

    data.year = temp;

    data.date = date;

    data.genre = genre;

    stringstream geek1(duration);

    geek >> temp;

    data.duration = temp;

    data.country = country;

    data.language = language;

    data.director = director;

    data.writer = writer;

    data.production\_company = production\_company;

    data.actors = actors;

    data.description = description;

    stringstream geek2(age\_vote);

    geek >> temp;

    data.age\_vote = double(temp);

    stringstream geek6(age\_vote);

    geek >> temp;

    data.votes = temp;

    data.budget = budget;

    data.usa\_gross\_income = usa\_gross\_income;

    stringstream geek3(meta\_score);

    geek >> temp;

    data.meta\_score = temp;

    data.worldwide\_gross\_income = worldwide\_gross\_income;

    stringstream geek4(reviews\_from\_user);

    geek >> temp;

    data.reviews\_from\_user = temp;

    stringstream geek5(reviews\_from\_critics);

    geek >> temp;

    data.reviews\_from\_critics = temp;

    lchild = NULL;

    rchild = NULL;

    height1 = 1;

}

class Movie\_AVL

{

public:

    MovieNode\* root;

    Movie\_AVL()

    {

        root = NULL;

    }

    MovieNode\* create\_node(string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actors, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        MovieNode\* newnode = new MovieNode(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        return newnode;

    }

    int max(int a, int b)

    {

        if (a > b)return a;

        return b;

    }

    int get\_height(MovieNode\* temp) {

        if (temp == NULL) {

            return 0;

        }

        return temp->height1;

    }

    int getBalanceFactor(MovieNode\* n)

    {

        if (n == NULL)

        {

            return 0;

        }

        return (get\_height(n->lchild) - get\_height(n->rchild));

    }

    MovieNode\* rightrotate(MovieNode\* y)

    {

        MovieNode\* x = y->lchild;

        MovieNode\* T2 = x->rchild;

        x->rchild = y;

        y->lchild = T2;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        return x;

    }

    MovieNode\* leftrotate(MovieNode\* x)

    {

        MovieNode\* y = x->rchild;

        MovieNode\* T2 = y->lchild;

        y->lchild = x;

        x->rchild = T2;

        x->height1 = max(get\_height(x->rchild), get\_height(x->lchild)) + 1;

        y->height1 = max(get\_height(y->rchild), get\_height(y->lchild)) + 1;

        return y;

    }

    MovieNode\* insert(MovieNode\* n, string title\_id, string title, string original\_title, string year, string date, string genre, string duration, string country, string language, string director, string writer, string production\_company, string actors, string description, string age\_vote, string votes, string budget, string usa\_gross\_income, string worldwide\_gross\_income, string meta\_score, string reviews\_from\_user, string reviews\_from\_critics)

    {

        if (n == NULL)

        {

            return create\_node(title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        if (title\_id < n->data.title\_id)

        {

            n->lchild = insert(n->lchild, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else if (title\_id > n->data.title\_id)

        {

            n->rchild = insert(n->rchild, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

        }

        else

            return n;

        n->height1 = 1 + max(get\_height(n->lchild), get\_height(n->rchild));

        int bf = getBalanceFactor(n);

        if (bf > 1 && title\_id < n->lchild->data.title\_id)

        {

            return rightrotate(n);

        }

        else if (bf<-1 && title\_id > n->rchild->data.title\_id)

        {

            return leftrotate(n);

        }

        else if (bf > 1 && title\_id > n->lchild->data.title\_id)

        {

            n->lchild = leftrotate(n->lchild);

            return rightrotate(n);

        }

        else if (bf < -1 && title\_id < n->rchild->data.title\_id)

        {

            n->rchild = rightrotate(n->rchild);

            return leftrotate(n);

        }

        return n;

    }

    void Print\_PreOrder(MovieNode\* temp) {

        if (temp == NULL) {

            //Base Case

            return;

        }

        //Go to the left child recursively

        Print\_PreOrder(temp->lchild);

        //print node

        cout << temp->data.genre << endl;

        //Then go to the right child recursively

        Print\_PreOrder(temp->rchild);

    }

};

class rating\_movie\_node {

public:

    string IMDB\_title\_ID;

    MovieNode\* movie\_ptr;

    double mean\_vote;

    void insert(string, MovieNode\*, string);

};

void rating\_movie\_node::insert(string IMDB\_title\_ID, MovieNode\* movie\_ptr, string s\_mean\_vote) {

    this->IMDB\_title\_ID = IMDB\_title\_ID;

    this->movie\_ptr = movie\_ptr;

    double temp;

    stringstream geek(s\_mean\_vote);

    geek >> temp;

    this->mean\_vote = temp;

}

void swap(rating\_movie\_node\*\* x, rating\_movie\_node\*\* y);

class MaxHeap{

    rating\_movie\_node\*\* harr;

    int capacity;

    int heap\_size;

public:

    MaxHeap(int capacity);

    int parent(int i) { return (i - 1) / 2; }

    int left(int i) { return (2 \* i + 1); }

    int right(int i) { return (2 \* i + 2); }

    void insertKey(rating\_movie\_node\* k);

    void display();

};

MaxHeap::MaxHeap(int cap)

{

    heap\_size = 0;

    capacity = cap;

    harr = new rating\_movie\_node\*[cap];

}

void MaxHeap::insertKey(rating\_movie\_node\* k)

{

    if (heap\_size == capacity)

    {

        cout << "\nOverflow: Could not insertKey\n";

        return;

    }

    heap\_size++;

    int i = heap\_size - 1;

    harr[i] = k;

    while (i != 0 && harr[parent(i)]->mean\_vote < harr[i]->mean\_vote)

    {

        swap(&harr[i], &harr[parent(i)]);

        i = parent(i);

    }

}

void MaxHeap::display() {

    for (size\_t i = 0; i < heap\_size; i++) {

        cout << harr[i]->movie\_ptr->data.title\_id << endl;

    }

}

void swap(rating\_movie\_node\*\* x, rating\_movie\_node\*\* y)

{

    rating\_movie\_node\* temp = \*x;

    \*x = \*y;

    \*y = temp;

}

void read(Movie\_AVL\* ml, MaxHeap\* h) {

    int count = 0;

    string title\_id;

    string title;

    string original\_title;

    string year;

    string date;

    string genre;

    string duration;

    string country;

    string language;

    string director;

    string writer;

    string production\_company;

    string actors;

    string description;

    string age\_vote;

    string votes;

    string budget;

    string usa\_gross\_income;

    string worldwide\_gross\_income;

    string meta\_score;

    string reviews\_from\_user;

    string reviews\_from\_critics;

    ifstream File("movies.csv");

    if (File.is\_open()) {

        while (!File.eof()) {

            if (count == 0) {

                string temp;

                getline(File, temp);

                count = 1;

            }

            else {

                getline(File, title\_id, ',');

                getline(File, title, ',');

                getline(File, original\_title, ',');

                getline(File, year, ',');

                getline(File, date, ',');

                getline(File, genre, ',');

                getline(File, duration, ',');

                getline(File, country, ',');

                getline(File, language, ',');

                getline(File, director, ',');

                getline(File, writer, ',');

                getline(File, production\_company, ',');

                getline(File, actors, ',');

                getline(File, description, ',');

                getline(File, age\_vote, ',');

                getline(File, votes, ',');

                getline(File, budget, ',');

                getline(File, usa\_gross\_income, ',');

                getline(File, meta\_score, ',');

                getline(File, reviews\_from\_user, ',');

                getline(File, reviews\_from\_critics);

                ml->root = ml->insert(ml->root, title\_id, title, original\_title, year, date, genre, duration, country, language, director, writer, production\_company, actors, description, age\_vote, votes, budget, usa\_gross\_income, worldwide\_gross\_income, meta\_score, reviews\_from\_user, reviews\_from\_critics);

            }

        }

    }

    else {

        cout << "Give Error\n" << endl;

    }

    count = 0;

    string imdb\_title\_id;

    MovieNode\* movie\_ptr;

    string mean\_vote;

    ifstream File1("ratings.csv");

    while (!File1.eof()) {

        string temp;

        if (count == 0) {

            getline(File1, temp);

            count = 1;

        }

        else {

            getline(File1, imdb\_title\_id, ',');

            getline(File1, temp, ',');

            getline(File1, temp, ',');

            getline(File1, mean\_vote, ',');

            getline(File1, temp, '\n');

            MovieNode\* traverver = ml->root;

            while (traverver != NULL && traverver->data.title\_id!=imdb\_title\_id) {

                if (imdb\_title\_id < traverver->data.title\_id) {

                    traverver = traverver->lchild;

                }

                else {

                    traverver = traverver->rchild;

                }

            }

            if (traverver != NULL) {

                movie\_ptr = traverver;

                rating\_movie\_node\* newnode = new rating\_movie\_node();

                newnode->insert(imdb\_title\_id, movie\_ptr, mean\_vote);

                h->insertKey(newnode);

            }

        }

    }

    File.close();

}

int main()

{

    Movie\_AVL ml;

    MaxHeap h(11);

    read(&ml,&h);

    cout<<"Movies according to their heighest mean votes are : "<<endl;

    h.display();

    getchar();

    return 0;

}

Text

Description automatically generated