1/18/2022

Group Members:

Shifa Imran (331352) & Haleema Sadia (332423)

Submitted to: DR. YASIR Faheem

Project Report

Data Structures and Algorithms

Project Description

**Netflix Movies Dataset Based Term Project**

There is a .csv file containing a dataset of 6235 movies and tv shows. For each of the 6235 movies and tv shows, details of 12 parameters have been provided like title, genre(listed\_in), directors, cast, country etc. We will use this dataset to create a database using various data structures studied in the class. One of our first tasks is to write a parser that extracts information from the csv file for each of the movies and tv shows.

**----------------------------------------------------Shows---------------------------------------------------**

A **Shows** Class that should have two doubly linked lists:

One linked list for Country**:**

* Store data of the movies which were shot in that country .

Second linked list for genre**:**

* Stores data of the movies of a specific genre

--------------------------------------------------**ShowsNode class---------------------------------**

We will store all the titles of movies and tv shows given in the dataset in AVL tree. For doing so, we shall need to define a ShowsNode class.

Class **ShowsNode**

{

Shows data;

ShowsNode \*leftchild;

ShowsNode \*rightchild;

DoublyLinkedList<Actor> pActor\_data; // contains actors of a specific show

DoublyLinkedList<Director> pDirector\_data; // contain directors of a specific show

}

**------------------------------------------ Release Year class--------------------------------------**

The Release Year class should provide the following functionalities:

It contains a doubly linked list of shows so that we can get the movies which are released in a specific year.

**------------------------------------------ Director Class--------------------------------------**

**DirectorNode** class which will store

a) Director’s name

b) A list of movies or tv shows he has directed.

**Hint:** store a list of pointers to the objects of the movie nodes a director has directed.

c) Provide functionality to Search directors by name. The search should return the name of directors along with the details of the movies they have directed.

**------------------------------------------ Actor Class--------------------------------------**

**Actor Node** class which should store:

a) The Name of an actor

b) A list of the shows in which he/she has acted. Remember, shows’ data is already stored in a list of type ShowsNode class. Thus, the list of cast’s movies should contain only pointers to the ShowsNodes in which he/she has acted.

**------------------------------------------ All Netflix Class--------------------------------------**

It contains the objects of AVL trees of Shows, Director and Actor class and objects of doubly linked list of Country, genre and release year.

* A method/function that reads the .csv file one line at a time.
* Automatically inserts all the entries of that line in various fields of the data part in the ShowsNode object. Note that the data field is of type Shows.
* Finally, insert the newly created ShowsNode object in the list.

|  |
| --- |
| **Source Code with Snapshot of Main Functions**  #include<iostream>  #include<fstream>  #include<string.h>  #include<sstream>  #include<ctime>  using *namespace* std;  *class* Node  {  *public:*      string data1;     //names  };  *template* <*class* T>  *class* ListNode  {  *public:*      T\* data;          //pointer variable      ListNode\* next;      ListNode\* prev;  };  *template* <*class* T>  *class* DoublyLinkedList  {  *public:*      ListNode<T>\* first; // special variable which stores address of head node.      ListNode<T>\* last; // special variable which stores address of the last node.  *int* length = 0;      ListNode<T>\* ploc; //to be used by Search(value) method to store address of logical    predecessor of value in a list.      ListNode<T>\* loc; //to be used by Search(value) method to store address of the node containing the searched value in a list. If it is not found it contains NULL.      DoublyLinkedList()      {          first = NULL;          last = NULL;          length = 0;          ploc = NULL;          loc = NULL;      }  *bool* isEmpty()      {          return first == NULL;      }  *void* PrintList(*bool* *dir*)      {          if (!isEmpty())          {              ListNode<T>\* temp = NULL;              if (*dir*)//print in forward direction              {                  temp = first;                  while (temp != NULL)                  {                      cout << temp->data->data1 << " \n";                      temp = temp->next;                  }              }              else //reverse order printing              {                  temp = last;                  while (temp != NULL)                  {                      cout << temp->data->data1 << " \n";                      temp = temp->prev;                  }                  cout << endl;              }          }          else              cout << "List is Empty" << endl;      }      //create here printshow details ftn and call it in netflix class ftn      //void PrintShowsDetails      // it will sort on the basis of release year  *void* Sort\_RealeaseYear\_withShows(*bool* *dir*)      {          if (!isEmpty())          {              ListNode<T>\* temp = NULL;              if (*dir*)//print in forward direction              {                  temp = first;                  while (temp != NULL)                  {                      cout << "Release Year: \n";                      cout << temp->data->data1 << " \n";                      cout << "Movies: \n";                      temp->data->pShows\_data.PrintList(true);                      temp = temp->next;                  }              }              else //reverse order printing              {                  temp = last;                  while (temp != NULL)                  {                      cout << temp->data->data1 << " \n";                      temp = temp->prev;                  }                  cout << endl;              }          }          else              cout << "List is Empty" << endl;      }  *void* InsertAtFront(T\* *value*)      {          ListNode<T>\* newnode = new ListNode<T>();          newnode->data = *value*;          if (isEmpty())          {              first = newnode;              last = newnode;          }          else          {              newnode->next = first;              first->prev = newnode;              first = newnode;          }          length++;      }  *void* InsertAtEnd(T\* *value*)      {          ListNode<T>\* newnode = new ListNode<T>();          newnode->data = *value*;          if (isEmpty())          {              first = newnode;              last = newnode;          }          else          {              last->next = newnode;              newnode->prev = last;              last = newnode;          }      }  *void* Search(string *value*)      {          loc = first;          ploc = NULL;          if (isEmpty())              return;          while (loc != NULL && loc->data->data1 < *value*)          {              ploc = loc;              loc = loc->next;          }          //Assign Null to Loc\_ is value not found,          //and its logical is anywhere in the list          //except after the last node.          if (loc != NULL && loc->data->data1 != *value*)              loc = NULL;      }  *void* InsertSorted(T\* *value*)      {          Search(*value*->data1);          //insert if value not found          //duplication not allowed          if (loc != NULL)          {              return;          }          else          {              if (ploc == NULL)                  InsertAtFront(*value*);              else if (ploc == last)                  InsertAtEnd(*value*);              else  //insert after ploc              {                  ListNode<T>\* newnode = new ListNode<T>();                  newnode->data = *value*;                  newnode->next = ploc->next;                  newnode->prev = ploc;                  ploc->next->prev = newnode;                  ploc->next = newnode;              }          }      }      // this function is required in AVL class for function PrintDirectorlikesSpecificGenre() , it searches genre and keeps tracks of genre count  *void* CountWithGenre(string *value*, *int*\* *count*, *bool* *dir*)      {          if (!isEmpty())          {              ListNode<T>\* temp = NULL;        // Pshowsdata    T is <shows>              if (*dir*) //print in forward direction              {                  temp = first;                  while (temp != NULL)                  {                      //cout << temp->data->data1 << " \n";                      temp->data->genre.Search(*value*);     //data is shows and then data has genre                      if (temp->data->genre.loc != NULL)                      {                          (\**count*)++;                      }                      temp = temp->next;                  }              }              else //reverse order printing              {                  temp = last;                  while (temp != NULL)                  {                      (\**count*)++;          //cout << temp->data->data1 << " \n";                      temp = temp->prev;                  }                  cout << endl;              }          }          else              cout << "List is Empty" << endl;      }  };  *template* <*class* T>  *class* AVL  {  *public:*      T\* root = NULL;      T\* loc = NULL;      T\* ploc = NULL;      //string director;      AVL()      {          root = NULL;      }  *bool* isEmpty()      {          return root == NULL;      }  *void* Search(string *value*)      {          ploc = NULL;          loc = root;          if (!isEmpty())          {              while (loc != NULL && loc->data->data1 != *value*)              {                  ploc = loc;                  if (*value* < loc->data->data1)                      loc = loc->lchild;                  else                      loc = loc->rchild;              } // end while          }//end if      }//end search        T\* InsertValue(T\* *value*, T\* *ploc*)      {          if (*ploc* == NULL)          {  *value*->height = 1;              return *value*;          }          else          {              Search(*value*->data->data1);              if (loc != NULL)                  cout << "";              else              {                  if (*value*->data->data1 < *ploc*->data->data1)                  {  *ploc*->lchild = InsertValue(*value*, *ploc*->lchild);                  }                  else if (*value*->data->data1 > *ploc*->data->data1)                  {  *ploc*->rchild = InsertValue(*value*, *ploc*->rchild);                  }                  else                      return *ploc*;              } //end else          } //end outer else  *ploc*->height = max(height(*ploc*->lchild), height(*ploc*->rchild)) + 1;  *int* balance = getBf(*ploc*);          if (balance > 1 && *value*->data->data1 < *ploc*->lchild->data->data1)              return rightRotate(*ploc*);          if (balance < -1 && *value*->data->data1 > *ploc*->rchild->data->data1)              return leftRotate(*ploc*);          // Left Right Case          if (balance > 1 && *value*->data->data1 > *ploc*->lchild->data->data1)          {  *ploc*->lchild = leftRotate(*ploc*->lchild);              return rightRotate(*ploc*);          }          // Right Left Case          if (balance < -1 && *value*->data->data1 < root->rchild->data->data1)          {  *ploc*->rchild = rightRotate(*ploc*->rchild);              return leftRotate(*ploc*);          }          return *ploc*;      } //end insertvalue  *int* height(T\* *node*)      {          if (*node* == NULL)              return 0;          else          {              return *node*->height;          }      }  *int* getBf(T\* *node*)      {          if (*node* == NULL)              return 0;          return height(*node*->lchild) - height(*node*->rchild);      }  *int* max(*int* *a*, *int* *b*)      {          if (*a* > *b*)              return *a*;          else              return *b*;      }      T\* leftRotate(T\* *x*)      {          T\* y = *x*->rchild;          T\* temp = y->lchild;          y->lchild = *x*;          if (y->lchild != NULL)          {  *x*->rchild = temp;          }  *x*->height = max(height(*x*->lchild), height(*x*->rchild)) + 1;          y->height = max(height(y->lchild), height(y->rchild)) + 1;          // Return new root          return y;      }      T\* rightRotate(T\* *x*)      {          T\* y = *x*->lchild;          T\* temp = y->rchild;          y->rchild = *x*;          if (y->rchild != NULL)          {  *x*->lchild = temp;          }  *x*->height = max(height(*x*->lchild), height(*x*->rchild)) + 1;          y->height = max(height(y->lchild), height(y->rchild)) + 1;          // Return new root          return y;      }  *void* PreOrder(T\* *ptree*)      {          if (*ptree* != NULL)          {  *ptree*->pShows\_data.PrintList(true);//data->data1 << " \n";              PreOrder(*ptree*->lchild);              PreOrder(*ptree*->rchild);          }      }  *void* InOrder(T\* *ptree*)      {          if (*ptree* != NULL)          {              InOrder(*ptree*->lchild);              cout << *ptree*->data->data1 << " ";              InOrder(*ptree*->rchild);          }      }  *void* PostOrder(T\* *ptree*)      {          if (*ptree* != NULL)          {              PostOrder(*ptree*->lchild);              PostOrder(*ptree*->rchild);              cout << *ptree*->data->data1 << " \n";          }      }      //this fn. print all dataset  *void* PrintInOrder(T\* *ptree*)      {          if (*ptree* != NULL)          {              PrintInOrder(*ptree*->lchild);              cout << "\nShow id: " << *ptree*->data->show\_id << " \n";              cout << "Type: " << *ptree*->data->type << " \n";              cout << "Title: " << *ptree*->data->data1 << " \n";              cout << "Country: ";  *ptree*->data->country.PrintList(true);              cout << "\nRelease Date: " << *ptree*->data->date\_added << " \n";              cout << "Release Year: " << *ptree*->data->release\_year << " \n";              cout << "Rating: " << *ptree*->data->rating << " \n";              cout << "Duration: " << *ptree*->data->duration << " \n";              cout << "Genre: ";  *ptree*->data->genre.PrintList(true);              cout << "\nDirector: ";  *ptree*->pDirector\_data.PrintList(true);              cout << "\nCast: ";  *ptree*->pActor\_data.PrintList(true);              cout << "\nDescription: " << *ptree*->data->description << " \n";              cout << "Release Year: " << *ptree*->data->release\_year << " \n";              PrintInOrder(*ptree*->rchild);          }      }      //this function sorts titles  *void* SortTitles(T\* *ptree*)           //inorder for titles sorting      {          if (*ptree* != NULL)          {              SortTitles(*ptree*->lchild);              cout << "Title: " << *ptree*->data->data1 << " \n";              SortTitles(*ptree*->rchild);          }      }      //this function prints shows of a specific director afer finding director of the input show title  *void* PrintShows\_SpecificDirector(T\* *ptree*, string *value*)      {          if (*ptree* != NULL)          {              PrintShows\_SpecificDirector(*ptree*->lchild, *value*);  *ptree*->pShows\_data.Search(*value*);              if (*ptree*->pShows\_data.loc != NULL)              {  *ptree*->pShows\_data.PrintList(true);                  cout << *ptree*->data->data1 << " \n";              }              PrintShows\_SpecificDirector(*ptree*->rchild, *value*);          }      }  //director root      // prints director names based on specific input genre  *void* PrintDirector\_SpecificGenre(T\* *ptree*, string *value*)   //shows root      {          if (*ptree* != NULL)          {              PrintDirector\_SpecificGenre(*ptree*->lchild, *value*);  *ptree*->data->genre.Search(*value*);              if (*ptree*->data->genre.loc != NULL)              {                  cout << "Title::" << *ptree*->data->data1 << " \n";  *ptree*->pDirector\_data.PrintList(true);              }              PrintDirector\_SpecificGenre(*ptree*->rchild, *value*);          }      }      // this function searches movies based on Type "Movie" and displays all show details  *void* Search\_based\_on\_Type(T\* *ptree*)      {          if (*ptree* != NULL)          {              Search\_based\_on\_Type(*ptree*->lchild);              if (*ptree*->data->type == "Movie")              {                  cout << "\n\n\n\n\n\n\nShow id: " << *ptree*->data->show\_id << " \n";                  cout << "Type: " << *ptree*->data->type << " \n";                  cout << "Title: " << *ptree*->data->data1 << " \n";                  cout << "Country: ";  *ptree*->data->country.PrintList(true);                  cout << "\nRelease Date: " << *ptree*->data->date\_added << " \n";                  cout << "Release Year: " << *ptree*->data->release\_year << " \n";                  cout << "Rating: " << *ptree*->data->rating << " \n";                  cout << "Duration: " << *ptree*->data->duration << " \n";                  cout << "Genre: ";  *ptree*->data->genre.PrintList(true);                  cout << "\nCast: ";  *ptree*->pActor\_data.PrintList(true);                  cout << "\nDirector::";                  cout << "\nDescription: " << *ptree*->data->description << " \n";              }              Search\_based\_on\_Type(*ptree*->rchild);          }      }      // this function searches movies based on Type "TvShow" and displays all show details  *void* Search\_based\_onTvShow(T\* *ptree*)      {          if (*ptree* != NULL)          {              Search\_based\_onTvShow(*ptree*->lchild);              if (*ptree*->data->type == "TV Show")              {                  cout << "\n\n\n\n\n\n\nShow id: " << *ptree*->data->show\_id << " \n";                  cout << "Type: " << *ptree*->data->type << " \n";                  cout << "Title: " << *ptree*->data->data1 << " \n";                  cout << "Country: ";  *ptree*->data->country.PrintList(true);                  cout << "\nRelease Date: " << *ptree*->data->date\_added << " \n";                  cout << "Release Year: " << *ptree*->data->release\_year << " \n";                  cout << "Rating: " << *ptree*->data->rating << " \n";                  cout << "Duration: " << *ptree*->data->duration << " \n";                  cout << "Genre: ";  *ptree*->data->genre.PrintList(true);                  cout << "\nCast: ";  *ptree*->pActor\_data.PrintList(true);                  cout << "\nDirector::";                  cout << "\nDescription: " << *ptree*->data->description << " \n";              }              Search\_based\_onTvShow(*ptree*->rchild);          }      }      // this function searches movies based on release year mentioned and displays all show details of that year  *void* Search\_based\_on\_ReleaseYear(T\* *ptree*, string *ry*, *int*\* *count*)      {          if (*ptree* != NULL)          {              Search\_based\_on\_ReleaseYear(*ptree*->lchild, *ry*, *count*);              if ((*ptree*->data->release\_year) == *ry*)              {                  cout << "\nShow id: " << *ptree*->data->show\_id << " \n";                  cout << "Type: " << *ptree*->data->type << " \n";                  cout << "Title: " << *ptree*->data->data1 << " \n";                  cout << "Country: ";  *ptree*->data->country.PrintList(true);                  cout << "\nRelease Date: " << *ptree*->data->date\_added << " \n";                  cout << "Release Year: " << *ptree*->data->release\_year << " \n";                  cout << "Rating: " << *ptree*->data->rating << " \n";                  cout << "Duration: " << *ptree*->data->duration << " \n";                  cout << "Genre: ";  *ptree*->data->genre.PrintList(true);                  cout << "\nCast: ";  *ptree*->pActor\_data.PrintList(true);                  cout << "\nDirector::";                  cout << "\nDescription: " << *ptree*->data->description << " \n";              }              Search\_based\_on\_ReleaseYear(*ptree*->rchild, *ry*, *count*);          }      }      // this function searches movies based on rating mentioned and displays all show details of that rating  *void* Search\_based\_on\_Rating(T\* *ptree*, string *rate*, *int*\* *count*)      {          if (*ptree* != NULL)          {              Search\_based\_on\_Rating(*ptree*->lchild, *rate*, *count*);              if ((*ptree*->data->rating) == *rate*)              {                  cout << "\n\n\n\n\n\n\nShow id: " << *ptree*->data->show\_id << " \n";                  cout << "Type: " << *ptree*->data->type << " \n";                  cout << "Title: " << *ptree*->data->data1 << " \n";                  cout << "Country: ";  *ptree*->data->country.PrintList(true);                  cout << "\nRelease Date: " << *ptree*->data->date\_added << " \n";                  cout << "Release Year: " << *ptree*->data->release\_year << " \n";                  cout << "Rating: " << *ptree*->data->rating << " \n";                  cout << "Duration: " << *ptree*->data->duration << " \n";                  cout << "Genre: ";  *ptree*->data->genre.PrintList(true);                  cout << "\nCast: ";  *ptree*->pActor\_data.PrintList(true);                  cout << "\nDirector::";                  cout << "\nDescription: " << *ptree*->data->description << " \n";              }              Search\_based\_on\_Rating(*ptree*->rchild, *rate*, *count*);          }      }      // this function searches movies based on title mentioned and displays all show details of that title  *void* Search\_based\_on\_Title(T\* *ptree*, string *title*, *int*\* *count*)      {          if (*ptree* != NULL)          {              Search\_based\_on\_Title(*ptree*->lchild, *title*, *count*);              if ((*ptree*->data->data1) == *title*)              {                  (\**count*)++;                  cout << "\n\n\n\n\n\n\nShow id: " << *ptree*->data->show\_id << " \n";                  cout << "Type: " << *ptree*->data->type << " \n";                  cout << "Title: " << *ptree*->data->data1 << " \n";                  cout << "Country: ";  *ptree*->data->country.PrintList(true);                  cout << "\nRelease Date: " << *ptree*->data->date\_added << " \n";                  cout << "Release Year: " << *ptree*->data->release\_year << " \n";                  cout << "Rating: " << *ptree*->data->rating << " \n";                  cout << "Duration: " << *ptree*->data->duration << " \n";                  cout << "Genre: ";  *ptree*->data->genre.PrintList(true);                  cout << "\nCast: ";  *ptree*->pActor\_data.PrintList(true);                  cout << "\nDirector::";                  cout << "\nDescription: " << *ptree*->data->description << " \n";              }              Search\_based\_on\_Title(*ptree*->rchild, *title*, *count*);          }      }      // this function searches year displays its cast and title of that year  *void* PrintCast\_based\_on\_ReleaseYear(T\* *ptree*, string *ry*, *int*\* *count*)      {          if (*ptree* != NULL)          {              PrintCast\_based\_on\_ReleaseYear(*ptree*->lchild, *ry*, *count*);              if ((*ptree*->data->release\_year) == *ry*)              {                  cout << "Title: " << *ptree*->data->data1 << " \n";                  cout << "\nCast: ";  *ptree*->pActor\_data.PrintList(true);              }              PrintCast\_based\_on\_ReleaseYear(*ptree*->rchild, *ry*, *count*);          }      }      // this function searches rating displays its cast and title of that rating  *void* PrintCast\_based\_on\_Rating(T\* *ptree*, string *rate*, *int*\* *count*)      {          if (*ptree* != NULL)          {              PrintCast\_based\_on\_Rating(*ptree*->lchild, *rate*, *count*);              if ((*ptree*->data->rating) == *rate*)              {                  cout << "Title: " << *ptree*->data->data1 << " \n";                  cout << "Cast: ";  *ptree*->pActor\_data.PrintList(true);              }              PrintCast\_based\_on\_Rating(*ptree*->rchild, *rate*, *count*);          }      }      // this function searches title displays its cast  *void* PrintCast\_based\_on\_Title(T\* *ptree*, string *title*, *int*\* *count*)      {          if (*ptree* != NULL)          {              PrintCast\_based\_on\_Title(*ptree*->lchild, *title*, *count*);              if ((*ptree*->data->data1) == *title*)              {                  cout << "Cast: ";  *ptree*->pActor\_data.PrintList(true);                  cout << "\n";              }              PrintCast\_based\_on\_Title(*ptree*->rchild, *title*, *count*);          }      }      // it prints only those director names who worked on the input genre >=2 times  *void* PrintDirectorlikesSpecificGenre(T\* *ptree*, string *value*)   //director root      {  *int* count = 0;          if (*ptree* != NULL)          {              PrintDirectorlikesSpecificGenre(*ptree*->lchild, *value*);              // ptree->data->genre.Search(value);  *ptree*->pShows\_data.CountWithGenre(*value*, &count, true);  // fn. called from DLL class.              if (count >= 2)              {                  cout << *ptree*->data->data1 << endl;       //print director names of that genre              }              PrintDirectorlikesSpecificGenre(*ptree*->rchild, *value*);          }      }  };  *class* Shows  {  *public:*      // declaring attributes of each movie/TVShow      string show\_id;      string type;      string data1; //title of movie or show      DoublyLinkedList<Node> country;      string date\_added;      string release\_year;      string rating;      string duration;      DoublyLinkedList<Node> genre;      string description;      //constructor      Shows()      {          show\_id = '\0';          type = '\0';          data1 = '\0';          country;          date\_added = '\0';          release\_year = '\0';          rating = '\0';          duration = '\0';          genre;          description = '\0';      }  };  *class* Actor  {  *public:*      string data1;   // name of actor  };  *class* Director  {  *public:*      string data1;     //director name  };  *class* ShowsNode  {  *public:*      Shows\* data;            // pointer variable pointing to the members of show class      ShowsNode\* lchild;      ShowsNode\* rchild;      //Actor\* pActor\_data;      DoublyLinkedList<Actor> pActor\_data;          // contains actors of a specific show      DoublyLinkedList<Director> pDirector\_data;    // contain directors of a specific show  *int* height;      //constructor      ShowsNode()      {          data = NULL;          lchild = NULL;          rchild = NULL;          height = 0;      }  };  *class* Director\_Node  {  *public:*      Director\* data;                       // pointer variable pointing to the members of director class      Director\_Node\* lchild;      Director\_Node\* rchild;      //Shows\* pShows\_data;      DoublyLinkedList<Shows> pShows\_data;    //list of shows for a specific director  *int* height;      //constructor      Director\_Node()      {          data = NULL;          lchild = NULL;          rchild = NULL;          height = 0;      }  };  *class* ReleaseYear  {  *public:*      string data1;                             //release year      DoublyLinkedList<Shows> pShows\_data;      // list of shows in a specific release year  };  *class* Actor\_Node  {  *public:*      Actor\* data;                //pointer variable pointing to the members of actor class      Actor\_Node\* lchild;      Actor\_Node\* rchild;      //Shows\* pShows\_data;      DoublyLinkedList<Shows> pShows\_data;        //list of shows for a specific cast member  *int* height;      //constructor      Actor\_Node()      {          data = NULL;          lchild = NULL;          rchild = NULL;          height = 0;      }  };  *class* All\_Netflix  {  *public:*      //declaration of objects for all classes that are either in AVL or Doubly LinkedList      AVL<ShowsNode> Shows1;      AVL<Director\_Node> Director1;      AVL<Actor\_Node> Actor1;      DoublyLinkedList<Node> Country1;      DoublyLinkedList<Node> Genre1;      DoublyLinkedList<ReleaseYear> ReleaseYear1;      /\*All\_Netflix()      {          Shows1 = new AVL();          Director1 = new AVL();          Actor1 = new AVL();      }\*/      // read() funtion that parses our dataset - reads and stores in specific data structures.  *void* read()      {          ifstream fin;     // ofject of ifstream class for file reading          string line;          string substr;  *int* pos = 0;          //char l;          //char line[1000];          fin.open("netflix\_titles.csv");          getline(fin, line);   // reading top row of dataset (column names)          while (fin.good())      // read until eof          {              getline(fin, line, ',');           // as csv is comma separated file              // cout << line << endl;              Shows\* show1 = new Shows();        //dynamic object of Shows class              ShowsNode\* SN = new ShowsNode();    //dynamic object of ShowsNode class              SN->data = show1;              show1->show\_id = line;              //type              getline(fin, line, ',');              show1->type = line;              //title              getline(fin, line, ',');              show1->data1 = line;              //Director              getline(fin, line, ',');    //read all directors              if (line != "")                 // if line is not empty              {                  if (line.find('~') != 4294967295)    //(line.at(0) == '"')  if tilda not found, then it gives string max value mentioned here                  {  // more than 1 directtor                      //substr = line.substr(1, (line.size() - 1));                      if (line.find('~') != 4294967295)                      {                          pos = line.find('~');                          substr = line.substr(0, pos);   //1st director                          line = line.substr(pos + 1, (line.size() - 1));   //remaining directors stored in line                      }                      else                      {                          substr = line.substr(1, (line.size() - 1));                      }                      Director1.Search(substr);                      if (Director1.loc == NULL)                      {                          Director\* director1 = new Director();                          director1->data1 = substr;                          Director\_Node\* DN = new Director\_Node();                          DN->data = director1;                          DN->pShows\_data.InsertSorted(show1);                          Director1.root = Director1.InsertValue(DN, Director1.root);                          SN->pDirector\_data.InsertSorted(director1);                      }                      else                      {                          Director1.loc->pShows\_data.InsertSorted(show1);                          SN->pDirector\_data.InsertSorted(Director1.loc->data);                      }                      while (line.find('~') != 4294967295)   //remaining directors excluding last one                      {                          //getline(fin, line, '~');                          pos = line.find('~');                          substr = line.substr(1, pos - 1);                          line = line.substr(pos + 1, (line.size() - 1));                          //if (line.at(line.size() - 1) == '"')                          //{                              //Director\* director1 = new Director();                          //substr = line.substr(0, (line.size() - 1));                          Director1.Search(substr);                          if (Director1.loc == NULL)                          {                              Director\* director1 = new Director();                              director1->data1 = substr;                              Director\_Node\* DN = new Director\_Node();                              DN->data = director1;                              DN->pShows\_data.InsertSorted(show1);                              Director1.root = Director1.InsertValue(DN, Director1.root);                              SN->pDirector\_data.InsertSorted(director1);                          }                          else                          {                              Director1.loc->pShows\_data.InsertSorted(show1);                              SN->pDirector\_data.InsertSorted(Director1.loc->data);                          }                          // break;                           //}                      }                      line = line.substr(1, (line.size() - 1));   // store last director                      Director1.Search(line);                      if (Director1.loc == NULL)                      {                          Director\* director1 = new Director();                          //substr = line.substr(0, (line.size() - 2));                          director1->data1 = line;                          Director\_Node\* DN = new Director\_Node();                          DN->data = director1;                          DN->pShows\_data.InsertSorted(show1);                          Director1.root = Director1.InsertValue(DN, Director1.root);                          SN->pDirector\_data.InsertSorted(director1);                      }                      else                      {                          Director1.loc->pShows\_data.InsertSorted(show1);                          SN->pDirector\_data.InsertSorted(Director1.loc->data);                      }                  }                  else      // if only 1 director                  {                      Director1.Search(line);                      if (Director1.loc == NULL)                      {                          Director\* director1 = new Director();                          //substr = line.substr(0, (line.size() - 2));                          director1->data1 = line;                          Director\_Node\* DN = new Director\_Node();                          DN->data = director1;                          DN->pShows\_data.InsertSorted(show1);                          Director1.root = Director1.InsertValue(DN, Director1.root);                          SN->pDirector\_data.InsertSorted(director1);                      }                      else                      {                          Director1.loc->pShows\_data.InsertSorted(show1);                          SN->pDirector\_data.InsertSorted(Director1.loc->data);                      }                  }              }                pos = 0;              //Actor              getline(fin, line, ',');  // line stores all the actors              if (line != "")           //if line is not empty              {                  if (line.find('~') != 4294967295)     //if tilda is not found then it returns max value of string                      //(line.at(0) == '"') //&& line.at((line.size() - 1)) != '"')                  {                      //Actor\* actor1 = new Actor();                      //actor1->data1 = substr;                        //substr = line.substr(1, (line.size() - 1));                      if (line.find('~') != 4294967295)                      {                          pos = line.find('~');                          substr = line.substr(0, pos);                          line = line.substr(pos + 1, (line.size() - 1));                      }                      else                      {                          substr = line.substr(1, (line.size() - 1));                      }                      Actor1.Search(substr);                      if (Actor1.loc == NULL)                      {                          Actor\* actor1 = new Actor();                          actor1->data1 = substr;                          Actor\_Node\* AN = new Actor\_Node();                          AN->data = actor1;                          AN->pShows\_data.InsertSorted(show1);                          Actor1.root = Actor1.InsertValue(AN, Actor1.root);                          SN->pActor\_data.InsertSorted(actor1);                      }                      else                      {                          Actor1.loc->pShows\_data.InsertSorted(show1);                          SN->pActor\_data.InsertSorted(Actor1.loc->data);                      }                      while (line.find('~') != 4294967295)                      {                          //getline(fin, line, ',');                          pos = line.find('~');                          substr = line.substr(1, pos - 1);                          line = line.substr(pos + 1, (line.size() - 1));                          //if (line.at(line.size() - 1) == '"')                          //{                              //Actor\* actor1 = new Actor();                              //substr = line.substr(0, (line.size() - 1));                          Actor1.Search(substr);                          if (Actor1.loc == NULL)                          {                              Actor\* actor1 = new Actor();                              actor1->data1 = substr;                              Actor\_Node\* AN = new Actor\_Node();                              AN->data = actor1;                              AN->pShows\_data.InsertSorted(show1);                              Actor1.root = Actor1.InsertValue(AN, Actor1.root);                              SN->pActor\_data.InsertSorted(actor1);                          }                          else                          {                              Actor1.loc->pShows\_data.InsertSorted(show1);                              SN->pActor\_data.InsertSorted(Actor1.loc->data);                          }                          //break;                      //}                      }                      line = line.substr(1, (line.size() - 1));                      Actor1.Search(line);                      if (Actor1.loc == NULL)                      {                          Actor\* actor1 = new Actor();                          //substr = line.substr(0, (line.size() - 2));                          actor1->data1 = line;                          Actor\_Node\* AN = new Actor\_Node();                          AN->data = actor1;                          AN->pShows\_data.InsertSorted(show1);                          Actor1.root = Actor1.InsertValue(AN, Actor1.root);                          SN->pActor\_data.InsertSorted(actor1);                      }                      else                      {                          Actor1.loc->pShows\_data.InsertSorted(show1);                          SN->pActor\_data.InsertSorted(Actor1.loc->data);                      }                  }                  else                  {                      Actor1.Search(line);                      if (Actor1.loc == NULL)                      {                          Actor\* actor1 = new Actor();                          //substr = line.substr(0, (line.size() - 2));                          actor1->data1 = line;                          Actor\_Node\* AN = new Actor\_Node();                          AN->data = actor1;                          AN->pShows\_data.InsertSorted(show1);                          Actor1.root = Actor1.InsertValue(AN, Actor1.root);                          SN->pActor\_data.InsertSorted(actor1);                      }                      else                      {                          Actor1.loc->pShows\_data.InsertSorted(show1);                          SN->pActor\_data.InsertSorted(Actor1.loc->data);                      }                  }              }                //Country              getline(fin, line, ',');              if (line != "")              {                  if (line.find('~') != 4294967295)//(line.at(0) == '"')                  {                      //Node\* country1 = new Node();                      if (line.find('~') != 4294967295)                      {                          pos = line.find('~');                          substr = line.substr(0, pos);                          line = line.substr(pos + 1, (line.size() - 1));                      }                      else                      {                          substr = line.substr(1, (line.size() - 1));                      }                        //substr = line.substr(1, (line.size() - 1));                      Country1.Search(substr);                      if (Country1.loc == NULL)                      {                          Node\* country1 = new Node();                          country1->data1 = substr;                          show1->country.InsertSorted(country1);                          Country1.InsertSorted(country1); //New Line Added                      }                      else                      {                          show1->country.InsertSorted(Country1.loc->data);                      }                      while (line.find('~') != 4294967295)                      {                          //getline(fin, line, ',');                          //if (line.at(line.size() - 1) == '"')                          //{                          pos = line.find('~');                          substr = line.substr(1, pos - 1);                          line = line.substr(pos + 1, (line.size() - 1));                          //substr = line.substr(0, line.size() - 1);                          Country1.Search(substr);                          if (Country1.loc == NULL)                          {                              Node\* country1 = new Node();                              country1->data1 = substr;                              show1->country.InsertSorted(country1);                              Country1.InsertSorted(country1); //New Line Added                          }                          else                          {                              show1->country.InsertSorted(Country1.loc->data);                          }                          //break;                      //}                      }                      // "USA,Pakistan,Australia"                      line = line.substr(1, (line.size() - 1));                      Country1.Search(line);                      if (Country1.loc == NULL)                      {                          Node\* country1 = new Node();                          country1->data1 = line;                          show1->country.InsertSorted(country1);                          Country1.InsertSorted(country1); //New Line Added                      }                      else                      {                          show1->country.InsertSorted(Country1.loc->data);                      }                  }                  else                  {                      Country1.Search(line);                      if (Country1.loc == NULL)                      {                          Node\* country1 = new Node();                          country1->data1 = line;                          show1->country.InsertSorted(country1);                          Country1.InsertSorted(country1); //New Line Added                      }                      else                      {                          show1->country.InsertSorted(Country1.loc->data);                      }                  }              }              //date\_added              getline(fin, line, ',');              show1->date\_added = line;              //release year              getline(fin, line, ',');              if (line != "")              {                  show1->release\_year = line; //stoi(line);                  ReleaseYear1.Search(line);                  if (ReleaseYear1.loc != NULL)                  {                      ReleaseYear1.loc->data->pShows\_data.InsertSorted(show1);                  }                  else                  {                      ReleaseYear\* RY1 = new ReleaseYear();                      RY1->data1 = show1->release\_year;                      RY1->pShows\_data.InsertSorted(show1);                      ReleaseYear1.InsertSorted(RY1);                  }              }              //rating              getline(fin, line, ',');              show1->rating = line;              //duration              getline(fin, line, ',');              show1->duration = line;              //Genre              getline(fin, line, ',');              if (line != "")              {                  if (line.find('~') != 4294967295)//(line.at(0) == '"')                  {                      //Node\* genre1 = new Node();                      if (line.find('~') != 4294967295)                      {                          pos = line.find('~');                          substr = line.substr(0, pos);                          line = line.substr(pos + 1, (line.size() - 1));                      }                      else                      {                          substr = line.substr(1, (line.size() - 1));                      }                        //substr = line.substr(1, (line.size() - 1));                      Genre1.Search(substr);                      if (Genre1.loc == NULL)                      {                          Node\* genre1 = new Node();                          genre1->data1 = substr;                          show1->genre.InsertSorted(genre1);                          Genre1.InsertSorted(genre1); //New Line Added                      }                      else                      {                          show1->genre.InsertSorted(Genre1.loc->data);                      }                      while (line.find('~') != 4294967295)                      {                          //getline(fin, line, ',');                          //if (line.at(line.size() - 1) == '"')                          //{                          pos = line.find('~');                          substr = line.substr(1, pos - 1);                          line = line.substr(pos + 1, (line.size() - 1));                          //substr = line.substr(0, (line.size() - 1));                          Genre1.Search(substr);                          if (Genre1.loc == NULL)                          {                              Node\* genre1 = new Node();                              genre1->data1 = substr;                              show1->genre.InsertSorted(genre1);                              Genre1.InsertSorted(genre1); //New Line Added                          }                          else                          {                              show1->genre.InsertSorted(Genre1.loc->data);                          }                          //break;                      //}                      }                      line = line.substr(1, (line.size() - 1));                      Genre1.Search(line);                      if (Genre1.loc == NULL)                      {                          Node\* genre1 = new Node();                          genre1->data1 = line;                          show1->genre.InsertSorted(genre1);                          Genre1.InsertSorted(genre1); //New Line Added                      }                      else                      {                          show1->genre.InsertSorted(Genre1.loc->data);                      }                  }                  else                  {                      Genre1.Search(line);                      if (Genre1.loc == NULL)                      {                          Node\* genre1 = new Node();                          genre1->data1 = line;                          show1->genre.InsertSorted(genre1);                          Genre1.InsertSorted(genre1); //New Line Added                      }                      else                      {                          show1->genre.InsertSorted(Genre1.loc->data);                      }                  }              }              getline(fin, line);              show1->description = line;              Shows1.root = Shows1.InsertValue(SN, Shows1.root);              //Shows2.InsertValue\_showid(SN);          }      }      //this ftn. inserts a new record of show in data structure(BST of Shows)  *void* insert\_new\_record()      {          string new\_showid, new\_type, new\_title, new\_cast;          string new\_director, new\_dateadded, new\_relYear, new\_country;          string new\_rating, new\_duration, new\_genre, new\_desc;          cout << "\nEnter Show ID: ";          cin.ignore();          getline(cin, new\_showid);          //getline(fin, line, ',');          // cout << line << endl;          Shows\* show1 = new Shows();          ShowsNode\* SN = new ShowsNode();          SN->data = show1;          show1->show\_id = new\_showid; //stoi(line);          cout << "\nEnter Type: ";          //cin.ignore();          getline(cin, new\_type);          show1->type = new\_type;          cout << "\nEnter Title: ";          //cin.ignore();          getline(cin, new\_title);          show1->data1 = new\_title;          //Director  *int* dircount;          cout << "How many directors you want to add.\n";          cin >> dircount;          for (*int* i = 0; i < dircount; i++)          {              cout << "\nEnter Name of Director" << i + 1 << ": ";              //cin.ignore();              getline(cin, new\_director);              Director1.Search(new\_director);              if (Director1.loc == NULL)              {                  Director\* director1 = new Director();                  director1->data1 = new\_director;                  Director\_Node\* DN = new Director\_Node();                  DN->data = director1;                  DN->pShows\_data.InsertSorted(show1);                  Director1.root = Director1.InsertValue(DN, Director1.root);                  SN->pDirector\_data.InsertSorted(director1);              }              else              {                  Director1.loc->pShows\_data.InsertSorted(show1);                  SN->pDirector\_data.InsertSorted(Director1.loc->data);              }          }          //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*     //Actor  *int* act\_count;          cout << "How many actors you want to add.\n";          cin >> act\_count;          for (*int* i = 0; i < act\_count; i++)          {              cout << "\nEnter Name of Actor" << i + 1 << ": ";              cin.ignore();              getline(cin, new\_cast);              Actor1.Search(new\_cast);              if (Actor1.loc == NULL)              {                  Actor\* actor1 = new Actor();                  actor1->data1 = new\_cast;                  Actor\_Node\* AN = new Actor\_Node();                  AN->data = actor1;                  AN->pShows\_data.InsertSorted(show1);                  Actor1.root = Actor1.InsertValue(AN, Actor1.root);                  SN->pActor\_data.InsertSorted(actor1);              }              else              {                  Actor1.loc->pShows\_data.InsertSorted(show1);                  SN->pActor\_data.InsertSorted(Actor1.loc->data);              }          }          //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      //Country  *int* countries\_count;          cout << "How many countries you want to add.\n";          cin >> countries\_count;          for (*int* i = 0; i < countries\_count; i++)          {              cout << "\nEnter Name of Country" << i + 1 << ": ";              //cin.ignore();              getline(cin, new\_country);              Country1.Search(new\_country);              if (Country1.loc == NULL)              {                  Node\* country1 = new Node();                  country1->data1 = new\_country;                  show1->country.InsertSorted(country1);                  Country1.InsertSorted(country1); //New Line Added              }              else              {                  show1->country.InsertSorted(Country1.loc->data);              }              getline(cin, new\_dateadded);              show1->date\_added = new\_dateadded;          }          //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      //release\_year          cout << "\nEnter Release Year: ";          //cin.ignore();          getline(cin, new\_relYear);          show1->release\_year = new\_relYear; //stoi(line);          ReleaseYear1.Search(new\_relYear);          if (ReleaseYear1.loc != NULL)          {              ReleaseYear1.loc->data->pShows\_data.InsertSorted(show1);          }          else          {              ReleaseYear\* RY1 = new ReleaseYear();              RY1->data1 = show1->release\_year;              RY1->pShows\_data.InsertSorted(show1);              ReleaseYear1.InsertSorted(RY1);          }          cout << "\nEnter Rating: ";          //cin.ignore();          getline(cin, new\_rating);          show1->rating = new\_rating;          cout << "\nEnter Duration: ";          //cin.ignore();          getline(cin, new\_duration);          show1->duration = new\_duration;          //\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*      //Genre  *int* genre\_count;          cout << "How many genres you want to add.\n";          cin >> genre\_count;          for (*int* i = 0; i < genre\_count; i++)          {              cout << "\nEnter Genre" << i + 1 << ": ";              //cin.ignore();              getline(cin, new\_genre);              Genre1.Search(new\_genre);              if (Genre1.loc == NULL)              {                  Node\* genre1 = new Node();                  genre1->data1 = new\_genre;                  show1->genre.InsertSorted(genre1);                  Genre1.InsertSorted(genre1); //New Line Added              }              else              {                  show1->genre.InsertSorted(Genre1.loc->data);              }          }          cout << "\nEnter Description: ";          //cin.ignore();          getline(cin, new\_desc);          show1->description = new\_desc;          Shows1.root = Shows1.InsertValue(SN, Shows1.root);          //Shows2.InsertValue\_showid(SN);      }      // it prints all netflix dataset with all details of shows  *void* PrintDataSet()      {          Shows1.PrintInOrder(Shows1.root);      }      //printing shows of a particular director  *void* PrintShowDetails\_Of\_Director(string *value*)      {          Director1.Search(*value*);          if (Director1.loc == NULL)          {              cout << "\nDirector not found" << endl;          }          else          {              Director1.loc->pShows\_data.PrintList(true);          }      }      /\*      void Print\_Release\_year(string ry)      {          Shows1.Search\_based\_on\_ReleaseYear(Shows1.root, ry);      }      void Print\_On\_Rating(string rate)      {          Shows1.Search\_based\_on\_Rating(Shows1.root, rate);      }      void PrintCast\_ReleaseYear(string ry)      {          Shows1.PrintCast\_based\_on\_ReleaseYear(Shows1.root, ry);      }      void PrintCast\_Rating(string rating)      {          Shows1.PrintCast\_based\_on\_Rating(Shows1.root, rating);      }      void PrintCast\_Title(string title)      {          Shows1.PrintCast\_based\_on\_Title(Shows1.root, title);      }      void Print\_On\_Title(string title)      {          Shows1.Search\_based\_on\_Title(Shows1.root, title);      }\*/  };  *int* main()  {      All\_Netflix TNMS;          cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;      cout<<"\*\t\t\t\t\t\t\*"<<endl;      cout<<"\*             FIND@NETFLIX\t\t\t\*"<<endl;      cout<<"\*\t\t\t\t\t\t\*"<<endl;      cout<<"\*\t\t\t\t\t\t\*"<<endl;      cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;      cout << "\*\*\*ALL NETFLIX MOVIES AND TV SHOWS\*\*\*";      TNMS.read();      string title, genre, director, rating, releaseYear;  *int* count = 0;  *int* choice;      do {          cout << "\n\nSelect your required function from the following Menu: \n";          cout << "\n                             \*MENU\*\n";          cout << "0. To exit\n1. Print All Movies and TV Shows Details in Netflix Dataset.\n  **Snapshot of Printing All Shows Details**    2. Search and Print based on Title.\n  **Snapshot of Print based on title**    3. Search and Print based on Release year.";          cout << "\n  **Snapshot of Print based on Release Year**    4. Search and Print based on Rating.\n  **Snapshot of Print based on Rating**    5. Search and Print based on Type-Movie.\n  **Snapshot of Print based on Type-Movie**    6. Search and Print based on Type-TvShow.\n  **Snapshot of Print based on Type-TvShow**    7. Sort on the basis of Title.";          cout << "\n    8. Sort on the basis of Release Year.\n  **Snapshot of Print based on Type-Release Year**    9. Print all movies of a particular director.\n  **Snapshot of Print based on Particular Director**    10. Print Director On the basis of Genre.\n  **Snapshot of Print Director based on genre**      11. Print all the shows of the director through a specific show.";          cout << "\n  **Snapshot of Print Shows based on Directors through Shows**    12. Search director who likes to direct a specific genre.\n  **Snapshot of Print Director likes to direct specific genre**    13. Print cast on the basis of show title.\n  **Snapshot of Print Cast of Show**    14. Print cast on the basis of show releasing year.";          cout << "\n  **Snapshot of Print Cast of Show**    15. Print cast on the basis of show rating.\n  **Snapshot of Print Cast of Show**    16. Insert a New MOVIE/TVSHOW record.\n";  **Snapshot of Print Cast of Show**              cin >> choice;          switch (choice)          {          case 0:              exit(0);          case 1:              TNMS.PrintDataSet();              break;          case 2:              cout << "\nEnter Title of the Movie/Tv Show:";              cin.ignore();              getline(cin, title);              TNMS.Shows1.Search\_based\_on\_Title(TNMS.Shows1.root, title, &count);              if (count == 0)                  cout << "\nNot found.";              count = 0;              break;          case 3:              cout << "\nEnter Release Year of the Movie/Tv Show:";              cin.ignore();              getline(cin, releaseYear);              TNMS.Shows1.Search\_based\_on\_ReleaseYear(TNMS.Shows1.root, releaseYear, &count);              if (count == 0)                  cout << "\nNot found.";              count = 0;              break;          case 4:              cout << "\nEnter Rating of the Movie/Tv Show:";              cin.ignore();              getline(cin, rating);              TNMS.Shows1.Search\_based\_on\_Rating(TNMS.Shows1.root, rating, &count);              if (count == 0)                  cout << "\nNot found.";              count = 0;              break;          case 5:              TNMS.Shows1.Search\_based\_on\_Type(TNMS.Shows1.root);              break;          case 6:              TNMS.Shows1.Search\_based\_onTvShow(TNMS.Shows1.root);              break;          case 7:              TNMS.Shows1.SortTitles(TNMS.Shows1.root);              break;          case 8:              cout << "\nEnter Release Year of the Movie/Tv Show:";              cin.ignore();              getline(cin, releaseYear);              TNMS.ReleaseYear1.Sort\_RealeaseYear\_withShows(true);              break;          case 9:              cout << "\nEnter Director of the Movie/Tv Show:";              cin.ignore();              getline(cin, director);              TNMS.PrintShowDetails\_Of\_Director(director);              break;          case 10:              cout << "\nEnter Gnere of the Movie/Tv Show:";              cin.ignore();              getline(cin, genre);              TNMS.Shows1.PrintDirector\_SpecificGenre(TNMS.Shows1.root, genre);              break;          case 11:              cout << "\nEnter Title of the Movie/Tv Show:";              cin.ignore();              getline(cin, title);              TNMS.Director1.PrintShows\_SpecificDirector(TNMS.Director1.root, title);              break;          case 12:              cout << "\nEnter Gnere of the Movie/Tv Show:";              cin.ignore();              getline(cin, genre);              TNMS.Director1.PrintDirectorlikesSpecificGenre(TNMS.Director1.root, genre);              break;          case 13:              cout << "\nEnter Title of the Movie/Tv Show:";              cin.ignore();              getline(cin, title);              TNMS.Shows1.PrintCast\_based\_on\_Title(TNMS.Shows1.root, title, &count);              if (count == 0)                  cout << "\nNot found.";              count = 0;              break;          case 14:              cout << "\nEnter Releasing Year of the Movie/Tv Show:";              cin.ignore();              getline(cin, releaseYear);              TNMS.Shows1.PrintCast\_based\_on\_ReleaseYear(TNMS.Shows1.root, releaseYear, &count);              if (count == 0)                  cout << "\nNot found.";              count = 0;              break;          case 15:              cout << "\nEnter Rating of the Movie/Tv Show :: ";              cin.ignore();              getline(cin, rating);              TNMS.Shows1.PrintCast\_based\_on\_Rating(TNMS.Shows1.root, rating, &count);              if (count == 0)                  cout << "\nNot found.";              count = 0;              break;          case 16:              cout << "\nEnter New Movie/TvShow Record :: \n";              TNMS.insert\_new\_record();              break;            }      } while (1);      return 0;  } |

*\*Note: As the data set contains 6000+ rows and is very large so we do not added all the screenshots of all the values as it is clear from some certain examples that it is displaying correct information.*