**The structure used to sort the data, uses Merchandise to store the data and Node as the struct to hold the data of the tree**

class BinTree

{

public:

//Constructors

BinTree();

~BinTree();

//Getters

bool retrieve(const Merchandise &target, Merchandise\* &location) const;

bool isEmpty() const;

//Setters

bool insert(Merchandise\* data);

void makeEmpty();

//Operator Overloads

friend ostream& operator<<(ostream &print, const BinTree &bin);

BinTree& operator=(const BinTree &);

private:

struct Node

{

Merchandise\* data;

Node\* left;

Node\* right;

};

Node\* root;

//Private and Helper Functions

void inOrderPrint(Node\* root, ostream &print) const;

void emptyTree(Node\* &root);

bool findData(Node\* root, const Merchandise &target, Merchandise\* &location, Node\* &nodeLocation) const;

Node\* copyUtil(Node \*);

};

#endif

**Child Class of Merchandise and Parent class of Comedy, Drama, and Classic** **common attributes of movies: the director, the title and the year it was released. It also contains methods for receiving and displaying those attributes.**

class Movie: public Merchandise

{

public:

//Constructors

Movie();

Movie(istream&);

virtual ~Movie();

//Getters

string getDirector() const;

string getTitle() const;

int getYear() const;

virtual ostream& display(ostream&) const;

virtual void displayHeader() const;

virtual void displayForHistory() const;

//Operator Overloads

virtual bool operator==(const Merchandise&) const = 0;

virtual bool operator<(const Merchandise&) const = 0;

protected:

string director;

string title;

int releasedYear;

virtual void setData(istream&);

};

#endif

**Child Class of Movie. Uses unique attributes and own display and operator functions**

class Classic: public Movie

{

public:

//Constructors

Classic();

Classic(const int&, const int&, const string&);

Classic(istream&);

~Classic();

//Getters

string getActorName() const;

int getMonth() const;

virtual ostream& display(ostream&) const;

virtual void displayHeader() const;

virtual void displayForHistory() const;

//Operator Overloads

virtual bool operator==(const Merchandise&) const;

virtual bool operator<(const Merchandise&) const;

private:

string actorName;

int releasedMonth;

virtual void setData(istream&);

};

#endif

**This class is designed to be the child class of a Movie and uses several of its inherited functions. However, because parameters in which this class is considered to be "sorted," it defines its own comparison for less than.**

class Comedy: public Movie

{

public:

//Constructors

Comedy();

Comedy(istream&);

Comedy(const string&, const int&);

~Comedy();

//Getters

virtual void displayHeader() const;

virtual void displayForHistory() const;

//Operator Overloads

virtual bool operator==(const Merchandise&) const;

virtual bool operator<(const Merchandise&) const;

};

#endif

**Same as Comedy**

class Drama: public Movie

{

public:

//Constructors

Drama();

Drama(istream&);

Drama(const string&, const string&);

~Drama();

//Getters

virtual void displayHeader() const;

virtual void displayForHistory() const;

//Operator Overloads

virtual bool operator==(const Merchandise&) const;

virtual bool operator<(const Merchandise&) const;

};

#endif

**Represents Customer object. Displays their transaction history, name and ID. Uses Transaction function to add transactions and access transaction history.**

class Customer

{

friend ostream &operator<<(ostream &, const Customer &);

private:

int id;

string firstName;

string lastName;

vector<Transaction \*> transactionHistory;

public:

Customer(int, string, string);

Customer();

~Customer();

string getName() const;

int getID() const;

void displayHistory() const;

void addTransaction(Transaction \*);

void deleteUtil();

};

#endif

**This class is designed as a parent class for Movie and Superclass for Comedy, Drama, Classic in the store. It tracks the stock of each item and for Movies, it keeps track of the format.**

class Merchandise

{

public:

//Constructors

Merchandise();

virtual ~Merchandise();

//Getters

int getStock() const;

virtual ostream& display(ostream&) const;

virtual void displayHeader() const = 0;

virtual void displayForHistory() const;

string getFormat() const;

inline int getMaxStock() { return maxStock; };

//Setters

bool increaseStock();

bool decreaseStock();

bool setStock(const int&);

void setMaxStock(const int&);

void setFormat(const string&);

//Operator Overloads

virtual bool operator==(const Merchandise&) const = 0;

virtual bool operator<(const Merchandise&) const = 0;

friend ostream& operator<<(ostream&, const Merchandise&);

protected:

int stock;

int maxStock;

string format;

virtual void setData(istream&) = 0;

};

#endif

**Represents store object. Essentially this class is what runs the output of the entire operation. It accesses the data created in transaction, tracks Customers and their individual information, allows for borrowing and returning Merchandise items and stores data into the HashMap**

class Store

{

BinTree\* mediaTreeTable[TABLE\_SIZE];

HASH\_MAP<int, Customer> customerTable;

string storename;

void printCustomerHistory(int&);

void borrowUtil(int&, char&, char&, istringstream&);

void returnUtil(int&, char&, char&, istringstream&);

void formatString(string &);

public:

Store(string&);

~Store();

void setMerchandise(ifstream&);

void setCustomers(ifstream&);

void setCommands(ifstream&);

void allInventory() const noexcept;

void printCustomers() const;

inline string getStoreName() const noexcept;

};

#endif

**Basically just keeps track of transactions that have been done. Borrow and Return are handled in the Store class rather than being their own subclasses of transactions**

class Transaction

{

public:

Transaction(Merchandise \*, const char &);

~Transaction ();

void display() const;

private:

char type;

Merchandise \*merchandise;

};