

# VideoStore

Generated by Doxygen 1.8.17



<b>1 Bug List</b>	<b>1</b>
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Class Documentation</b>	<b>7</b>
4.1 Customer Struct Reference	7
4.1.1 Detailed Description	8
4.1.2 Constructor & Destructor Documentation	8
4.1.2.1 Customer() [1/2]	8
4.1.2.2 Customer() [2/2]	8
4.1.3 Member Data Documentation	8
4.1.3.1 cust	8
4.1.3.2 next	8
4.2 CustomerInfo Class Reference	9
4.2.1 Detailed Description	9
4.2.2 Constructor & Destructor Documentation	9
4.2.2.1 CustomerInfo()	10
4.2.3 Member Function Documentation	10
4.2.3.1 displayInfo()	10
4.2.3.2 getAccStatus()	10
4.2.3.3 getFName()	10
4.2.3.4 getID()	10
4.2.3.5 getLName()	11
4.2.3.6 getNumberOfRented()	11
4.2.3.7 getRentedList()	11
4.2.3.8 setAccStatus()	11
4.2.3.9 setFName()	12
4.2.3.10 setID()	12
4.2.3.11 setLName()	12
4.2.3.12 setNumberOfRented()	12
4.3 customerList Class Reference	13
4.3.1 Detailed Description	13
4.3.2 Constructor & Destructor Documentation	13
4.3.2.1 customerList()	13
4.3.3 Member Function Documentation	14
4.3.3.1 addCustomer()	14
4.3.3.2 displaylist()	14
4.3.3.3 getHead()	14
4.3.3.4 operator[]()	14
4.3.3.5 searchCustomer()	15

4.4 Rented Struct Reference	15
4.4.1 Detailed Description	16
4.4.2 Constructor & Destructor Documentation	16
4.4.2.1 Rented()	16
4.4.3 Member Data Documentation	16
4.4.3.1 customerID	17
4.4.3.2 dueDate	17
4.4.3.3 rentedON	17
4.5 Revenue Struct Reference	17
4.5.1 Detailed Description	18
4.5.2 Constructor & Destructor Documentation	18
4.5.2.1 Revenue() [1/2]	18
4.5.2.2 Revenue() [2/2]	18
4.5.3 Member Data Documentation	18
4.5.3.1 next	19
4.6 Time Struct Reference	19
4.6.1 Detailed Description	20
4.6.2 Constructor & Destructor Documentation	20
4.6.2.1 Time() [1/3]	20
4.6.2.2 Time() [2/3]	20
4.6.2.3 Time() [3/3]	20
4.6.3 Member Function Documentation	21
4.6.3.1 checkDate()	21
4.6.3.2 getDaysTill()	21
4.6.3.3 getNextWeek()	22
4.6.3.4 operator<()	22
4.6.3.5 operator<=()	22
4.6.3.6 operator=()	23
4.6.3.7 operator>()	23
4.6.3.8 operator>=()	23
4.6.3.9 print()	25
4.6.3.10 printDate()	25
4.6.4 Member Data Documentation	25
4.6.4.1 tsecs	25
4.7 Video Struct Reference	26
4.7.1 Detailed Description	26
4.7.2 Constructor & Destructor Documentation	26
4.7.2.1 Video() [1/2]	27
4.7.2.2 Video() [2/2]	27
4.7.3 Member Data Documentation	27
4.7.3.1 next	27
4.7.3.2 vid	27

4.8 VideoInfo Class Reference	27
4.8.1 Detailed Description	28
4.8.2 Constructor & Destructor Documentation	28
4.8.2.1 VideoInfo() [1/2]	29
4.8.2.2 VideoInfo() [2/2]	29
4.8.3 Member Function Documentation	29
4.8.3.1 displayDetails()	29
4.8.3.2 getAvailableCopies()	29
4.8.3.3 getMovieDirector()	30
4.8.3.4 getProtagonist()	30
4.8.3.5 getRentedCopies()	30
4.8.3.6 getTotalCopies()	30
4.8.3.7 getVideoTitle()	31
4.8.3.8 numberOfCopies()	31
4.8.3.9 operator=()	31
4.8.3.10 setAvailableCopies()	31
4.8.3.11 setMovieDirector()	32
4.8.3.12 setProtagonist()	32
4.8.3.13 setTotalCopies()	32
4.8.3.14 setVideoTitle()	32
4.9 videoList Class Reference	33
4.9.1 Detailed Description	34
4.9.2 Constructor & Destructor Documentation	34
4.9.2.1 videoList()	34
4.9.3 Member Function Documentation	34
4.9.3.1 addVideo()	34
4.9.3.2 getHead()	34
4.9.3.3 operator[]()	35
4.9.3.4 printCheckedInMovies()	35
4.9.3.5 printCheckedOut()	35
4.9.3.6 printMovies()	35
4.9.3.7 removeVideo()	36
4.9.3.8 searchParticular()	36
4.9.3.9 searchVideo()	36
4.10 videoStore Class Reference	37
4.10.1 Detailed Description	38
4.10.2 Constructor & Destructor Documentation	38
4.10.2.1 videoStore()	38
4.10.3 Member Function Documentation	38
4.10.3.1 addCustomers()	38
4.10.3.2 addMovie()	38
4.10.3.3 addRevenue()	38

4.10.3.4 checkAvailability()	39
4.10.3.5 deleteCasette()	39
4.10.3.6 getCurrentTime()	39
4.10.3.7 getCustList()	40
4.10.3.8 getOutStandingAmount()	40
4.10.3.9 getRevenueGenerated()	40
4.10.3.10 getVidList()	40
4.10.3.11 rentToCustomer()	40
4.10.3.12 returnMovie()	41
4.10.3.13 showDetails()	41
4.10.3.14 updateTime()	41
<b>5 File Documentation</b>	<b>43</b>
5.1 include/CustomerInfo.h File Reference	43
5.1.1 Detailed Description	44
5.2 include/customerList.h File Reference	45
5.2.1 Detailed Description	46
5.3 include/Time.h File Reference	47
5.3.1 Detailed Description	48
5.3.2 Function Documentation	48
5.3.2.1 convertToUppercase()	48
5.3.2.2 IsLeapYear()	48
5.4 include/Utilities.h File Reference	49
5.4.1 Detailed Description	50
5.4.2 Enumeration Type Documentation	50
5.4.2.1 STATUS	50
5.5 include/VideoInfo.h File Reference	51
5.5.1 Detailed Description	52
5.6 include/videoList.h File Reference	52
5.6.1 Detailed Description	54
5.7 include/videoStore.h File Reference	54
5.7.1 Detailed Description	56
5.8 src/CustomerInfo.cpp File Reference	57
5.8.1 Detailed Description	57
5.9 src/customerList.cpp File Reference	58
5.9.1 Detailed Description	59
5.10 src/main.cpp File Reference	60
5.10.1 Detailed Description	61
5.11 src/Time.cpp File Reference	62
5.11.1 Detailed Description	63
5.11.2 Function Documentation	63
5.11.2.1 convertToUppercase()	63
5.11.2.2 IsLeapYear()	63

---

5.12 src/VideoInfo.cpp File Reference . . . . .	64
5.12.1 Detailed Description . . . . .	64
5.13 src/videoList.cpp File Reference . . . . .	65
5.13.1 Detailed Description . . . . .	65
5.14 src/videoStore.cpp File Reference . . . . .	66
5.14.1 Detailed Description . . . . .	67
<b>Index</b>	<b>69</b>





# Chapter 1

## Bug List

File [main.cpp](#)

No known bugs



## Chapter 2

# Class Index

### 2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#">Customer</a>	Represents a node in the linked list for customerInfo. Contains the data of the customer and holds the pointer to the next node in the list . . . . .	7
<a href="#">CustomerInfo</a>	Represents a customer of the store object. Stores basic information about the customer such as their name. Assigns them a random but unique ID . . . . .	9
<a href="#">customerList</a>	Maintains a linked list for customerInfo objects.Used later in <a href="#">videoStore</a> to store the list of customers of the shop . . . . .	13
<a href="#">Rented</a>	A struct to hold info about when and who rented a particular movie. An array of <a href="#">Rented</a> is used inside <a href="#">VideoInfo</a> to keep track of which copies of a particular movie have been rented out . . .	15
<a href="#">Revenue</a>	Holds the revenue generated on a particular day/Time. The timestamps are represented by the <a href="#">Time</a> object and the amount generated is stored as well. Also represents a node in the linked list that holds <a href="#">Revenue</a> objects . . . . .	17
<a href="#">Time</a>	A struct to deal with time . . . . .	19
<a href="#">Video</a>	Represents a node for the linked list that will store <a href="#">VideoInfo</a> objects. Used to construct a linked list of <a href="#">VideoInfo</a> objects . . . . .	26
<a href="#">VideoInfo</a>	Represents a movie present in the store . . . . .	27
<a href="#">videoList</a>	Maintains the linked list where each node is a <a href="#">Video</a> object. A list to hold all the movies. This is done via linked list data structure of <a href="#">VideoInfo</a> objects. Implements necessary relevant methods as well . . . . .	33
<a href="#">videoStore</a>	Main class that represents our video store. Holds the methods to implement the core functionality of the store . . . . .	37



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all documented files with brief descriptions:

include/ <a href="#">CustomerInfo.h</a>	
<a href="#">CustomerInfo</a> class is defined here . . . . .	43
include/ <a href="#">customerList.h</a>	
Implements the class simulating the database for <a href="#">videoStore</a> in the form of a linked list . . . . .	45
include/ <a href="#">Time.h</a>	
Header interface for the struct <a href="#">Time</a> . . . . .	47
include/ <a href="#">Utilities.h</a>	
Contains definitions for some useful functions . . . . .	49
include/ <a href="#">VideoInfo.h</a>	
Contains the definition of the class <a href="#">VideoInfo</a> which is abstraction of a movie . . . . .	51
include/ <a href="#">videoList.h</a> . . . . .	52
include/ <a href="#">videoStore.h</a> . . . . .	54
src/ <a href="#">CustomerInfo.cpp</a>	
<a href="#">CustomerInfo</a> class is implemented . . . . .	57
src/ <a href="#">customerList.cpp</a>	
Defines and implements the <a href="#">customerList</a> class . . . . .	58
src/ <a href="#">main.cpp</a>	
The file contains the main function, the starting point of the program. Employs all the classes and structs to simulate a video store with a minimal functionality . . . . .	60
src/ <a href="#">Time.cpp</a>	
Contains the implementation of the struct class <a href="#">Time</a> . . . . .	62
src/ <a href="#">VideoInfo.cpp</a>	
Implementation of <a href="#">VideoInfo</a> class . . . . .	64
src/ <a href="#">videoList.cpp</a> . . . . .	65
src/ <a href="#">videoStore.cpp</a> . . . . .	66



## Chapter 4

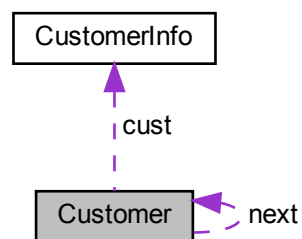
# Class Documentation

### 4.1 Customer Struct Reference

Represents a node in the linked last for customerInfo. Contains the data of the customer and holds the pointer to the next node in the list.

```
#include <customerList.h>
```

Collaboration diagram for Customer:



#### Public Member Functions

- **Customer** ()  
*Construct a new **Customer** node.*
- **Customer** (**CustomerInfo** &customr)  
*parameterized constructor for **Customer** object.*

#### Public Attributes

- **Customer** \* next
- **CustomerInfo** cust

### 4.1.1 Detailed Description

Represents a node in the linked last for customerInfo. Contains the data of the customer and holds the pointer to the next node in the list.

### 4.1.2 Constructor & Destructor Documentation

#### 4.1.2.1 Customer() [1/2]

```
Customer::Customer ( ) [inline]
```

Construct a new [Customer](#) node.

#### 4.1.2.2 Customer() [2/2]

```
Customer::Customer (
    CustomerInfo & customr ) [inline]
```

parameterized constructor for [Customer](#) object.

Parameters

<i>customr</i>	
----------------	--

### 4.1.3 Member Data Documentation

#### 4.1.3.1 cust

```
CustomerInfo Customer::cust
```

Holds the data of the customer.

#### 4.1.3.2 next

```
Customer* Customer::next
```

Points to the next node in the linked list.

The documentation for this struct was generated from the following file:

- [include/customerList.h](#)



## 4.2 CustomerInfo Class Reference

Represents a customer of the store object. Stores basic information about the customer such as their name. Assigns them a random but unique ID.

```
#include <CustomerInfo.h>
```

### Public Member Functions

- [CustomerInfo](#) ()  
*Construct a new [CustomerInfo](#) object.*
- int [getNumberOfRented](#) ()  
*Get the Number Of [Rented](#) movies.*
- void [setNumberOfRented](#) (int x)  
*Set the Number Of [Rented](#) movies. Increments or decrements the number of rented movies based on param x. If x==1 increments, else decrements.*
- string [getFName](#) ()  
*Returns first name of customer.*
- void [setFName](#) (string fName)  
*Sets first name of the customer.*
- string [getLName](#) ()  
*Returns last name of the customer.*
- void [setLName](#) (string lName)  
*Sets last name of the customer.*
- int [getID](#) ()  
*Returns ID Of the customer.*
- void [setID](#) (int ID)  
*Sets ID of the customer.*
- [STATUS](#) [getAccStatus](#) ()  
*Get the Acc Status of customer.*
- void [setAccStatus](#) ([STATUS](#) accStatus)  
*Set the Acc Status of customer.*
- [videoList](#) & [getRentedList](#) ()  
*Get the List of the rented movies.*
- void [displayInfo](#) ()  
*Displays the information of the customer.*

### 4.2.1 Detailed Description

Represents a customer of the store object. Stores basic information about the customer such as their name. Assigns them a random but unique ID.

### 4.2.2 Constructor & Destructor Documentation

#### 4.2.2.1 CustomerInfo()

```
CustomerInfo::CustomerInfo ( )
```

Construct a new [CustomerInfo](#) object.

### 4.2.3 Member Function Documentation

#### 4.2.3.1 displayInfo()

```
void CustomerInfo::displayInfo ( )
```

Displays the information of the customer.

#### 4.2.3.2 getAccStatus()

```
STATUS CustomerInfo::getAccStatus ( )
```

Get the Acc Status of customer.

Returns

STATUS

#### 4.2.3.3 getFName()

```
string CustomerInfo::getFName ( )
```

Returns first name of customer.

Returns

string

#### 4.2.3.4 getID()

```
int CustomerInfo::getID ( )
```

Returns ID Of the customer.

Returns

int

#### 4.2.3.5 getLName()

```
string CustomerInfo::getLName ( )
```

Returns last name of the customer.

Returns

string

#### 4.2.3.6 getNumberOfRented()

```
int CustomerInfo::getNumberOfRented ( )
```

Get the Number Of [Rented](#) movies.

Returns

int

#### 4.2.3.7 getRentedList()

```
videoList & CustomerInfo::getRentedList ( )
```

Get the List of the rented movies.

Returns

[videoList](#)&

#### 4.2.3.8 setAccStatus()

```
void CustomerInfo::setAccStatus (
    STATUS accStatus )
```

Set the Acc Status of customer.

Parameters

<i>accStatus</i>	
------------------	--

#### 4.2.3.9 setFName()

```
void CustomerInfo::setFName (
    string fName )
```

Sets first name of the customer.

Parameters

<i>fName</i>	
--------------	--

#### 4.2.3.10 setID()

```
void CustomerInfo::setID (
    int ID )
```

Sets ID of the customer.

Parameters

<i>ID</i>	
-----------	--

#### 4.2.3.11 setLName()

```
void CustomerInfo::setLName (
    string lName )
```

Sets last name of the customer.

Parameters

<i>lName</i>	
--------------	--

#### 4.2.3.12 setNumberOfRented()

```
void CustomerInfo::setNumberOfRented (
    int x )
```

Set the Number Of [Rented](#) movies. Increments or decrements the number of rented movies based on param x. If x==1 increments, else decrements.

Parameters

<i>x</i>	
----------	--

The documentation for this class was generated from the following files:

- include/[CustomerInfo.h](#)
- src/[CustomerInfo.cpp](#)

## 4.3 customerList Class Reference

Maintains a linked list for customerInfo objects.Used later in [videoStore](#) to store the list of customers of the shop.

```
#include <customerList.h>
```

### Public Member Functions

- [customerList](#) ()  
*Construct a new [customerList](#) object. Both head and tail initialized to null to represent an empty list.*
- [Customer](#) \* [getHead](#) ()  
*Get the Head of the list. Useful for methods outside the [customerList](#) class that need to access and traverse the [customerList](#) linked list.*
- void [addCustomer](#) ([Customer](#) \*node)  
*Adds a [Customer](#) node to the list. That is, a new customer is added to the list via the method.*
- int [searchCustomer](#) (int ID)  
*Searches for a particular customer in the list, and returns the index, at which they are present in the list. Returns -1 otherwise indicating that the customer is not present in the list.*
- void [displaylist](#) ()  
*Displays the list of the customers with their info such as name and ID.*
- [CustomerInfo](#) & [operator](#)[ ] (int ind)  
*Overloads [] operator to access any [CustomerInfo](#) object from the list via array indexing notation. Easier to use, and gets used extensively inside the [videoStore](#) methods. Function is guaranteed to return an object always (since it is preceeded by a call to check if the object is in the list), therefore a reference is being used plus I was lazy.*

### 4.3.1 Detailed Description

Maintains a linked list for customerInfo objects.Used later in [videoStore](#) to store the list of customers of the shop.

### 4.3.2 Constructor & Destructor Documentation

#### 4.3.2.1 customerList()

```
customerList::customerList ( )
```

Construct a new [customerList](#) object. Both head and tail initialized to null to represent an empty list.

### 4.3.3 Member Function Documentation

#### 4.3.3.1 addCustomer()

```
void customerList::addCustomer (
    Customer * node )
```

Adds a [Customer](#) node to the list. That is, a new customer is added to the list via the method.

Parameters

<i>node</i>	new <a href="#">Customer</a> node to add to the list.
-------------	---

#### 4.3.3.2 displaylist()

```
void customerList::displaylist ( )
```

Displays the list of the customers with their info such as name and ID.

#### 4.3.3.3 getHead()

```
Customer * customerList::getHead ( )
```

Get the Head of the list. Useful for methods outside the [customerList](#) class that need to access and traverse the [customerList](#) linked list.

Returns

Customer\*

#### 4.3.3.4 operator[]()

```
CustomerInfo & customerList::operator[] (
    int ind )
```

Overloads [] operator to access any [CustomerInfo](#) object from the list via array indexing notation. Easier to use, and gets used extensively inside the [videoStore](#) methods. Function is guaranteed to return an object always (since it is preceded by a call to check if the object is in the list), therefore a reference is being used plus I was lazy.

Parameters

<i>ind</i>	Index of the <a href="#">Customer</a> object in the list whose <a href="#">CustomerInfo</a> object is required.
------------	---

Returns

[CustomerInfo](#)&

#### 4.3.3.5 searchCustomer()

```
int customerList::searchCustomer (
    int ID )
```

Searches for a particular customer in the list, and returns the index, at which they are present in the list. Returns -1 otherwise indicating that the customer is not present in the list.

Parameters

<i>ID</i>	ID of the customer that we are looking for in the list.
-----------	---

Returns

int

The documentation for this class was generated from the following files:

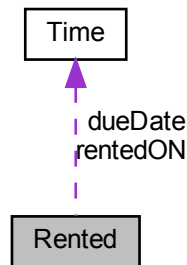
- include/[customerList.h](#)
- src/[customerList.cpp](#)

## 4.4 Rented Struct Reference

A struct to hold info about when and who rented a particular movie. An array of [Rented](#) is used inside [VideoInfo](#) to keep track of which copies of a particular movie have been rented out.

```
#include <VideoInfo.h>
```

Collaboration diagram for Rented:



## Public Member Functions

- [Rented \(\)](#)

Construct a new empty [Rented](#) object. A null `customerID` indicates the copy is not rented since all IDs are greater than 1.

## Public Attributes

- [Time](#) `rentedON`
- [Time](#) `dueDate`
- `int` `customerID`

### 4.4.1 Detailed Description

A struct to hold info about when and who rented a particular movie. An array of [Rented](#) is used inside [VideoInfo](#) to keep track of which copies of a particular movie have been rented out.

### 4.4.2 Constructor & Destructor Documentation

#### 4.4.2.1 Rented()

```
Rented::Rented ( ) [inline]
```

Construct a new empty [Rented](#) object. A null `customerID` indicates the copy is not rented since all IDs are greater than 1.

### 4.4.3 Member Data Documentation



#### 4.4.3.1 customerID

```
int Rented::customerID
```

ID of the customer who rented out the copy.

#### 4.4.3.2 dueDate

```
Time Rented::dueDate
```

The time at which the movie is due. It is one week away from the time at which it was rented on.

#### 4.4.3.3 rentedON

```
Time Rented::rentedON
```

The time at which the movie was rented.

The documentation for this struct was generated from the following file:

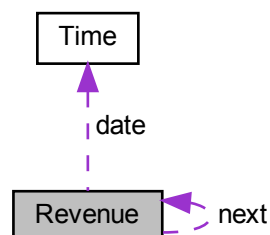
- `include/VideoInfo.h`

## 4.5 Revenue Struct Reference

Holds the revenue generated on a particular day/Time. The timestamps are represented by the [Time](#) object and the amount generated is stored as well. Also represents a node in the linked list that holds [Revenue](#) objects.

```
#include <videoStore.h>
```

Collaboration diagram for Revenue:



## Public Member Functions

- [Revenue](#) ()  
*Construct a new empty [Revenue](#) object.*
- [Revenue](#) ([Time](#) tdate, int amnt)  
*Parameterized constructor for [Revenue](#) object.*

## Public Attributes

- [Time](#) date
- int amount
- [Revenue](#) \* next

### 4.5.1 Detailed Description

Holds the revenue generated on a particular day/Time. The timestamps are represented by the [Time](#) object and the amount generated is stored as well. Also represents a node in the linked list that holds [Revenue](#) objects.

### 4.5.2 Constructor & Destructor Documentation

#### 4.5.2.1 [Revenue](#)() [1/2]

```
Revenue::Revenue ( ) [inline]
```

Construct a new empty [Revenue](#) object.

#### 4.5.2.2 [Revenue](#)() [2/2]

```
Revenue::Revenue (
    Time tdate,
    int amnt ) [inline]
```

Parameterized constructor for [Revenue](#) object.

Parameters

<i>tdate</i>	<a href="#">Time</a> object to hold the timestamp.
<i>amnt</i>	Amount generated at that time.

### 4.5.3 Member Data Documentation

### 4.5.3.1 next

`Revenue* Revenue::next`

Points to the next `Revenue` object in the linked list. The linked list will be implemented and used inside `videoStore`.

The documentation for this struct was generated from the following file:

- `include/videoStore.h`

## 4.6 Time Struct Reference

A struct to deal with time.

```
#include <Time.h>
```

### Public Member Functions

- `Time ()`  
*Construct a new `Time` object.*
- `Time (int t)`  
*Construct a new `Time` object.*
- `Time (short year, short month, short day, short hour=0, short minute=0, short second=0)`  
*Construct a new `Time` object.*
- `void printDate ()`  
*Prints only the date and not the time of the this `Time` object. Useful later, when only the date is required without the time. See `videoStore` method `videoStore::getRevenueGenerated()`*
- `void print ()`  
*Prints the date and time in the standard UTC format. `ctime()` adds a '\n' character which is being removed using `std::string::erase()`, since we don't want an endl, as time needs to be printed inline at certain occasions.*
- `bool checkDate (const Time &test)`  
*Checks whether the two dates are equal. Compares date of this object with the date of the param "test" and returns a bool accordingly.*
- `Time getNextWeek ()`  
*Get the time which is exactly 7 days from this object's time.*
- `int getDaysTill (const Time &t)`  
*Get how many days between this object and the passed param "t" time. Useful in calculating how many days overdue is the movie and thus is used to calculate fine.*
- `void operator= (const Time &t)`  
*Assignment operator for `Time` objects. Copies the total number of seconds from `Time` object parameter to this object.*
- `bool operator< (const Time &t)`  
*Comparison "<" operator for time objects. Useful later in checking whether a given time is ahead of the time being compared or before it.*
- `bool operator> (const Time &t)`  
*Comparison ">" operator for time objects. Useful later in checking whether a given time is ahead of the time being compared or before it.*
- `bool operator<= (const Time &t)`  
*Comparison "<=" operator for time objects. Useful later in checking whether a given time is ahead of the time being compared or before it.*
- `bool operator>= (const Time &t)`  
*Comparison ">=" operator for time objects. Useful later in checking whether a given time is ahead of the time being compared or before it.*

## Public Attributes

- time\_t [tsecs](#)

### 4.6.1 Detailed Description

A struct to deal with time.

The struct contains a time\_t variable which is the primary member variable. All calculations are performed on it to get the time displayed in standard format. Overloaded operators are defined as well to handle comparisons of times.

### 4.6.2 Constructor & Destructor Documentation

#### 4.6.2.1 Time() [1/3]

```
Time::Time ( )
```

Construct a new [Time](#) object.

Initializes time\_t to zero.

#### 4.6.2.2 Time() [2/3]

```
Time::Time (
    int t )
```

Construct a new [Time](#) object.

Sets the time object to a specific time.

Parameters

<i>t</i>	number of seconds since 1970
----------	------------------------------

#### 4.6.2.3 Time() [3/3]

```
Time::Time (
    short year,
    short month,
    short day,
    short hour = 0,
```

```
short minute = 0,  
short second = 0 )
```

Construct a new [Time](#) object.

Sets the time specified by proper date and time format.

Parameters

<i>year</i>	Year
<i>month</i>	Month
<i>day</i>	Day
<i>hour</i>	Hour
<i>minute</i>	Minute
<i>second</i>	Second

## 4.6.3 Member Function Documentation

### 4.6.3.1 `checkDate()`

```
bool Time::checkDate (  
    const Time & test )
```

Checks whether the two dates are equal. Compares date of this object with the date of the param "test" and returns a bool accordingly.

Parameters

<i>test</i>	<a href="#">Time</a> object Date to test if its equal to the current date.
-------------	--

Returns

bool

### 4.6.3.2 `getDaysTill()`

```
int Time::getDaysTill (  
    const Time & t )
```

Get how many days between this object and the passed param "t" time. Useful in calculating how many days overdue is the movie and thus is used to calculate fine.

Parameters

<i>t</i>	<a href="#">Time</a> object till which we need to find the total number of days.
----------	--

Returns

int

#### 4.6.3.3 getNextWeek()

```
Time Time::getNextWeek ( )
```

Get the time which is exactly 7 days from this object's time.

Returns

[Time](#)

#### 4.6.3.4 operator<()

```
bool Time::operator< (
    const Time & t )
```

Comparison "<" operator for time objects. Useful later in checking whether a given time is ahead of the time being compared or before it.

Parameters

<i>t</i>	
----------	--

Returns

true

false

#### 4.6.3.5 operator<=()

```
bool Time::operator<= (
    const Time & t )
```

Comparison "<=" operator for time objects. Useful later in checking whether a given time is ahead of the time being compared or before it.

Parameters

<i>t</i>	
----------	--

Returns

true  
false

#### 4.6.3.6 operator=()

```
void Time::operator= (
    const Time & t )
```

Assignment operator for [Time](#) objects. Copies the total number of seconds from [Time](#) object parameter to this object.

Parameters

<i>t</i>	
----------	--

#### 4.6.3.7 operator>()

```
bool Time::operator> (
    const Time & t )
```

Comparison ">" operator for time objects. Useful later in checking whether a given time is ahead of the time being compared or before it.

Parameters

<i>t</i>	
----------	--

Returns

true  
false

#### 4.6.3.8 operator>=()

```
bool Time::operator>= (
    const Time & t )
```

Comparison ">=" operator for time objects. Useful later in checking whether a given time is ahead of the time being compared or before it.



Parameters

<i>t</i>	
----------	--

Returns

true  
false

#### 4.6.3.9 print()

```
void Time::print ( )
```

Prints the date and time in the standard UTC format. `ctime()` adds a '\n' character which is being removed using `std::string::erase()`, since we don't want an newline, as time needs to be printed inline at certain occasions.

#### 4.6.3.10 printDate()

```
void Time::printDate ( )
```

Prints only the date and not the time of the this [Time](#) object. Useful later, when only the date is required without the time. See [videoStore](#) method [videoStore::getRevenueGenerated\(\)](#)

### 4.6.4 Member Data Documentation

#### 4.6.4.1 tsecs

```
time_t Time::tsecs
```

Primary variable for the struct, which stores the time in the form of total number of seconds that have passed since 1970 till the time that is being stored.

The documentation for this struct was generated from the following files:

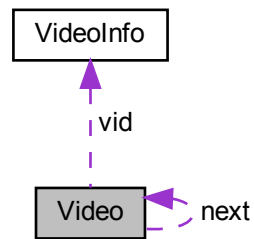
- [include/Time.h](#)
- [src/Time.cpp](#)

## 4.7 Video Struct Reference

Represents a node for the linked list that will store [VideoInfo](#) objects. Used to construct a linked list of [VideoInfo](#) objects.

```
#include <videoList.h>
```

Collaboration diagram for Video:



### Public Member Functions

- [Video](#) ()  
*Construct a new [Video](#) object. The next pointer is set to NULL since it is not pointing to anything initially.*
- [Video](#) ([VideoInfo](#) &mov)  
*Parameterized constructor for [Video](#) object.*

### Public Attributes

- [VideoInfo](#) vid
- [Video](#) \* next

#### 4.7.1 Detailed Description

Represents a node for the linked list that will store [VideoInfo](#) objects. Used to construct a linked list of [VideoInfo](#) objects.

#### 4.7.2 Constructor & Destructor Documentation

#### 4.7.2.1 Video() [1/2]

```
Video::Video ( ) [inline]
```

Construct a new [Video](#) object. The next pointer is set to NULL since it is not pointing to anything initially.

#### 4.7.2.2 Video() [2/2]

```
Video::Video (
    VideoInfo & mov ) [inline]
```

Parameterized constructor for [Video](#) object.

Parameters

<i>mov</i>	
------------	--

### 4.7.3 Member Data Documentation

#### 4.7.3.1 next

```
Video* Video::next
```

Points to the next node in the linked list.

#### 4.7.3.2 vid

```
VideoInfo Video::vid
```

Stores the data of the node.

The documentation for this struct was generated from the following file:

- [include/videoList.h](#)

## 4.8 VideoInfo Class Reference

Represents a movie present in the store.

```
#include <VideoInfo.h>
```

## Public Member Functions

- [VideoInfo](#) ()  
*Construct a new empty [VideoInfo](#) object.*
- [VideoInfo](#) (string vt, string pt, string md, int tc)  
*Parameterized constructor for [VideoInfo](#) object.*
- vector< [Rented](#) > & [getRentedCopies](#) ()  
*Get the rentedCopies vector.*
- string [getVideoTitle](#) ()  
*Get the [Video](#) Title.*
- void [setVideoTitle](#) (string VideoTitle)  
*Set the VideoTitle of the movie.*
- string [getProtagonist](#) ()  
*Get the Protagonist actor name of the movie.*
- void [setProtagonist](#) (string protagonist)  
*Set the Protagonist actor name.*
- string [getMovieDirector](#) ()  
*Get the Movie Director name.*
- void [setMovieDirector](#) (string movieDirector)  
*Set the Movie Director name.*
- int [getTotalCopies](#) ()  
*Get the Total Copies available of movie.*
- void [setTotalCopies](#) (int totalCopies)  
*Set the Total Copies of the movie.*
- int [getAvailableCopies](#) ()  
*Get the Available Copies of the movie.*
- void [setAvailableCopies](#) (int copies)  
*Set the Available Copies of the movie.*
- void [displayDetails](#) ()  
*Display details of the movie to the console.*
- void [numberOfCopies](#) ()  
*Displays the available and total number of copies of movie to the console.*
- void [operator=](#) (const [VideoInfo](#) &vi)  
*Assignment operator for [VideoInfo](#) object. Copies all the member variables of [VideoInfo](#) object vi to this object.*

### 4.8.1 Detailed Description

Represents a movie present in the store.

Stores necessary information about the movie itself, such as the title, director name etc. and also stores the information about the total number of copies and available number of copies.

### 4.8.2 Constructor & Destructor Documentation

#### 4.8.2.1 VideoInfo() [1/2]

```
VideoInfo::VideoInfo ( )
```

Construct a new empty [VideoInfo](#) object.

#### 4.8.2.2 VideoInfo() [2/2]

```
VideoInfo::VideoInfo (
    string vt,
    string pt,
    string md,
    int tc )
```

Parameterized constructor for [VideoInfo](#) object.

Parameters

<i>vt</i>	<a href="#">Video</a> title
<i>pt</i>	Protagonist name
<i>md</i>	Movie director name
<i>tc</i>	Total copies of the movie

### 4.8.3 Member Function Documentation

#### 4.8.3.1 displayDetails()

```
void VideoInfo::displayDetails ( )
```

Display details of the movie to the console.

#### 4.8.3.2 getAvailableCopies()

```
int VideoInfo::getAvailableCopies ( )
```

Get the Available Copies of the movie.

Returns

`int`

#### 4.8.3.3 getMovieDirector()

```
string VideoInfo::getMovieDirector ( )
```

Get the Movie Director name.

Returns

string

#### 4.8.3.4 getProtagonist()

```
string VideoInfo::getProtagonist ( )
```

Get the Protagonist actor name of the movie.

Returns

string

#### 4.8.3.5 getRentedCopies()

```
vector< Rented > & VideoInfo::getRentedCopies ( )
```

Get the rentedCopies vector.

Returns

vector<Rented>&

#### 4.8.3.6 getTotalCopies()

```
int VideoInfo::getTotalCopies ( )
```

Get the Total Copies available of movie.

Returns

int

#### 4.8.3.7 `getVideoTitle()`

```
string VideoInfo::getVideoTitle ( )
```

Get the [Video](#) Title.

Returns

string

#### 4.8.3.8 `numberOfCopies()`

```
void VideoInfo::numberOfCopies ( )
```

Displays the available and total number of copies of movie to the console.

#### 4.8.3.9 `operator=()`

```
void VideoInfo::operator= (
    const VideoInfo & vi )
```

Assignment operator for [VideoInfo](#) object. Copies all the member variables of [VideoInfo](#) object vi to this object.

Parameters

<i>vi</i>	<a href="#">VideoInfo</a> object to be copied
-----------	---

#### 4.8.3.10 `setAvailableCopies()`

```
void VideoInfo::setAvailableCopies (
    int copies )
```

Set the Available Copies of the movie.

Parameters

<i>copies</i>	
---------------	--

#### 4.8.3.11 setMovieDirector()

```
void VideoInfo::setMovieDirector (
    string movieDirector )
```

Set the Movie Director name.

Parameters

<i>movieDirector</i>	
----------------------	--

#### 4.8.3.12 setProtagonist()

```
void VideoInfo::setProtagonist (
    string protagonist )
```

Set the Protagonist actor name.

Parameters

<i>protagonist</i>	
--------------------	--

#### 4.8.3.13 setTotalCopies()

```
void VideoInfo::setTotalCopies (
    int totalCopies )
```

Set the Total Copies of the movie.

Parameters

<i>totalCopies</i>	
--------------------	--

#### 4.8.3.14 setVideoTitle()

```
void VideoInfo::setVideoTitle (
    string VideoTitle )
```

Set the VideoTitle of the movie.



Parameters

VideoTitle	
------------	--

The documentation for this class was generated from the following files:

- [include/VideoInfo.h](#)
- [src/VideoInfo.cpp](#)

## 4.9 videoList Class Reference

Maintains the linked list where each node is a [Video](#) object. A list to hold all the movies. This is done via linked list data structure of [VideoInfo](#) objects. Implements necessary relevant methods as well.

```
#include <videoList.h>
```

### Public Member Functions

- [videoList](#) ()  
*Construct a new empty [videoList](#) object. The head is set to NULL implying there is no object in the list.*
- [Video \\* getHead](#) ()  
*Get the head node of the list. Useful to access the list outside of the [videoList](#) class. Access only given to head and the list can be traversed through once head is known.*
- void [addVideo](#) ([Video \\*node](#))  
*Add a movie/Video node to the linked list.*
- int [searchVideo](#) (string title)  
*Search for a specific title in the list and return its index. Since there's only one movie stored with a particular title, the index, i.e. at what point in the list is the relevant node stored should be unique and is returned.*
- void [removeVideo](#) (string videotitle)  
*Remove a particular movie from the list using its title. Traverses through the list to find that particular node and removes that.*
- [VideoInfo & operator\[\]](#) (int ind)  
*Returns the [videoInfo](#) reference from the list. Overloading operator[] to use it just like the arrays, i.e. accessing a particular element from list using its index. Useful later in the [videoStore](#) to easily access the list elements instead of traversing through it again and again using while loop syntax. Function is guaranteed to return an object always (since it is preceded by a call to check if the object is in the list), therefore a reference is being used plus I was lazy.*
- void [printMovies](#) (bool titlesOnly)  
*Prints all the movies in the list. The param titlesOnly controls whether all the details are displayed or just the titles of the movies.*
- void [printCheckedInMovies](#) ()  
*Prints all the movies that have been checked in, i.e. all the movies in the store whose availableCopies is more than 1. One of the core functionalities of the [videoStore](#).*
- void [printCheckedOut](#) ()  
*Prints all the movies that have been checked out. That is, all those movies whose availableCopies is less than the total number of copies at the store.*
- void [searchParticular](#) (string inpt, char param)  
*Searches the list for the movies from a particular director or actor. The parameter param controls whether to look for movies from a particular actor or a particular director.*

## 4.9.1 Detailed Description

Maintains the linked list where each node is a [Video](#) object. A list to hold all the movies. This is done via linked list data structure of [VideoInfo](#) objects. Implements necessary relevant methods as well.

## 4.9.2 Constructor & Destructor Documentation

### 4.9.2.1 videoList()

```
videoList::videoList ( )
```

Construct a new empty [videoList](#) object. The head is set to NULL implying there is no object in the list.

## 4.9.3 Member Function Documentation

### 4.9.3.1 addVideo()

```
void videoList::addVideo (
    Video * node )
```

Add a movie/Video node to the linked list.

Parameters

<i>node</i>	
-------------	--

### 4.9.3.2 getHead()

```
Video * videoList::getHead ( )
```

Get the head node of the list. Useful to access the list outside of the [videoList](#) class. Access only given to head and the list can be traversed through once head is known.

Returns

[Video](#)\*

#### 4.9.3.3 operator[]()

```
VideoInfo & videoList::operator[] (
    int ind )
```

Returns the videoInfo reference from the list. Overloading operator[] to use it just like the arrays, i.e. accessing a particular element from list using its index. Useful later in the [videoStore](#) to easily access the list elements instead of traversing through it again and again using while loop syntax. Function is guaranteed to return an object always (since it is preceded by a call to check if the object is in the list), therefore a reference is being used plus I was lazy.

Parameters

<i>ind</i>	Index of the object required in the list.
------------	---

Returns

[VideoInfo](#)&

#### 4.9.3.4 printCheckedInMovies()

```
void videoList::printCheckedInMovies ( )
```

Prints all the movies that have been checked in, i.e. all the movies in the store whose availableCopies is more than 1. One of the core functionalities of the [videoStore](#).

#### 4.9.3.5 printCheckedOut()

```
void videoList::printCheckedOut ( )
```

Prints all the movies that have been checked out. That is, all those movies whose availableCopies is less than the total number of copies at the store.

#### 4.9.3.6 printMovies()

```
void videoList::printMovies (
    bool titlesOnly )
```

Prints all the movies in the list. The param titlesOnly controls whether all the details are displayed or just the titles of the movies.

Parameters

<i>titlesOnly</i>	Set to true if only the titles of the movies are required, otherwise false.
-------------------	---

#### 4.9.3.7 removeVideo()

```
void videoList::removeVideo (
    string videotitle )
```

Remove a particular movie from the list using its title. Traverses through the list to find that particular node and removes that.

Parameters

<i>videotitle</i>	
-------------------	--

#### 4.9.3.8 searchParticular()

```
void videoList::searchParticular (
    string inpt,
    char param )
```

Searches the list for the movies from a particular director or actor. The parameter param controls whether to look for movies from a particular actor or a particular director.

Parameters

<i>inpt</i>	Stores the director or the protagonist name depending upon the param.
<i>param</i>	'P' for protagonist and 'D' to look for directors.

#### 4.9.3.9 searchVideo()

```
int videoList::searchVideo (
    string title )
```

Search for a specific title in the list and return its index. Since there's only one movie stored with a particular title, the index, i.e. at what point in the list is the relevant node stored should be unique and is returned.

Parameters

<i>title</i>	Title of the movie that is being searched.
--------------	--

Returns

int Index of the movie in the list.

The documentation for this class was generated from the following files:

- [include/videoList.h](#)
- [src/videoList.cpp](#)

## 4.10 videoStore Class Reference

Main class that represents our video store. Holds the methods to implement the core functionality of the store.

```
#include <videoStore.h>
```

### Public Member Functions

- [videoStore \(\)](#)  
*Construct a new empty [videoStore](#) object.*
- [customerList & getCustList \(\)](#)  
*Returns the list of the customers.*
- [videoList & getVidList \(\)](#)  
*Returns the list of the movies at the store.*
- [Time getCurrentTime \(\)](#)  
*Returns the current time at the store.*
- void [addRevenue \(Revenue \\*node\)](#)  
*Adds a [Revenue](#) node to the list of the revenues.*
- void [updateTime \(\)](#)  
*Updates the time at a particular rate to create a simulation. Used later in the main loop to keep updating the time with every iteration of the loop.*
- void [addMovie \(\)](#)  
*Adds a new movie to the store. Takes in input from user, checks if the database already have the movie to maintain a unique database with no repetitions. Adds a movie if it doesn't exist before.*
- void [addCustomers \(\)](#)  
*Adds a new customer to the store database of customers. A new [CustomerInfo](#) object is added to the list of customers held by [videoStore](#). Each customer is assigned a new randomly generated but unique ID.*
- void [showDetails \(string title\)](#)  
*Shows the details of a particular movie, searches via the title of the movie.*
- bool [checkAvailability \(string inptitle\)](#)  
*Checks whether a particular movie, searched via the title is available at the store or not. The database is searched for the movie, and if not found, returns false. If the movie is available at the store, but all the copies are rented out, displays the details of all checked out copies and returns false, otherwise returns true.*
- void [rentToCustomer \(int id, string title\)](#)  
*Rents a particular movie to a particular customer. Checks if a movie is available for renting out by searching the database, if it is, checks if the user's account is overdue, if it isn't rents the user the movie, via their ID. Updates all the relevant information in the database as well such as checked in and out times, and who and when rented it and the revenue generated by it.*
- void [returnMovie \(int id, string title\)](#)  
*Checks in the movie that the user returns, ID and title of movie are checked in database, and when the movie is returned, fine is calculated if any and the availablecopies of the movie is updated. Other relevant variables are also updated.*
- int [getOutStandingAmount \(\)](#)  
*Get the Total outstanding amount by the current time. Goes through the list of all the movies at the store, then for each movie searches if any of its copy is overdue, if it is, calculates the fine for that and adds that to total amount. Returns the amount then.*
- void [getRevenueGenerated \(\)](#)  
*Calculate and print the total revenue generated within a given time frame given by user. Core functionality of the [videoStore](#).*
- void [deleteCasette \(string title\)](#)  
*Deletes a movie casette from the store. Removes a movie completely from the store's movie list if there is no copy anymore in the store.*

### 4.10.1 Detailed Description

Main class that represents our video store. Holds the methods to implement the core functionality of the store.

### 4.10.2 Constructor & Destructor Documentation

#### 4.10.2.1 videoStore()

```
videoStore::videoStore ( )
```

Construct a new empty [videoStore](#) object.

### 4.10.3 Member Function Documentation

#### 4.10.3.1 addCustomers()

```
void videoStore::addCustomers ( )
```

Adds a new customer to the store database of customers. A new [CustomerInfo](#) object is added to the list of customers held by [videoStore](#). Each customer is assigned a new randomly generated but unique ID.

#### 4.10.3.2 addMovie()

```
void videoStore::addMovie ( )
```

Adds a new movie to the store. Takes in input from user, checks if the database already have the movie to maintain a unique database with no repetitions. Adds a movie if it doesn't exist before.

#### 4.10.3.3 addRevenue()

```
void videoStore::addRevenue (
    Revenue * node )
```

Adds a [Revenue](#) node to the list of the revenues.

Parameters

<i>node</i>	A revenue node to add to the list.
-------------	------------------------------------

#### 4.10.3.4 checkAvailability()

```
bool videoStore::checkAvailability (
    string inptitle )
```

Checks whether a particular movie, searched via the title is available at the store or not. The database is searched for the movie, and if not found, returns false. If the movie is available at the store, but all the copies are rented out, displays the details of all checked out copies and returns false, otherwise returns true.

Parameters

<i>inptitle</i>	Title of the movie to look for.
-----------------	---------------------------------

Returns

bool

#### 4.10.3.5 deleteCasette()

```
void videoStore::deleteCasette (
    string title )
```

Deletes a movie cassette from the store. Removes a movie completely from the store's movie list if there is no copy anymore in the store.

Parameters

<i>title</i>	Title of movie to delete the cassette of.
--------------	---

#### 4.10.3.6 getCurrentTime()

```
Time videoStore::getCurrentTime ( )
```

Returns the current time at the store.

Returns

Time

#### 4.10.3.7 getCustList()

```
customerList & videoStore::getCustList ( )
```

Returns the list of the customers.

Returns

```
customerList&
```

#### 4.10.3.8 getOutStandingAmount()

```
int videoStore::getOutStandingAmount ( )
```

Get the Total outstanding amount by the current time. Goes through the list of all the movies at the store, then for each movie searches if any of its copy is overdue, if it is, calculates the fine for that and adds that to total amount. Returns the amount then.

Returns

```
int
```

#### 4.10.3.9 getRevenueGenerated()

```
void videoStore::getRevenueGenerated ( )
```

Calculate and print the total revenue generated within a given time frame given by user. Core functionality of the [videoStore](#).

#### 4.10.3.10 getVidList()

```
videoList & videoStore::getVidList ( )
```

Returns the list of the movies at the store.

Returns

```
videoList&
```

#### 4.10.3.11 rentToCustomer()

```
void videoStore::rentToCustomer (
    int id,
    string title )
```

Rents a particular movie to a particular customer. Checks if a movie is available for renting out by searching the database, if it is, checks if the user's account is overdue, if it isn't rents the user the movie, via their ID. Updates all the relevant information in the database as well such as checked in and out times, and who and when rented it and the revenue generated by it.



Parameters

<i>id</i>	ID of the user who wants to rent the movie.
<i>title</i>	Title of the movie being requested.

#### 4.10.3.12 returnMovie()

```
void videoStore::returnMovie (
    int id,
    string title )
```

Checks in the movie that the user returns, ID and title of movie are checked in database, and when the movie is returned, fine is calculated if any and the availablecopies of the movie is updated. Other relevant variables are also updated.

Parameters

<i>id</i>	ID of the user who wants to return the movie.
<i>title</i>	title of the movie being returned.

#### 4.10.3.13 showDetails()

```
void videoStore::showDetails (
    string title )
```

Shows the details of a particular movie, searches via the title of the movie.

Parameters

<i>title</i>	title of movie to display the details.
--------------	--

#### 4.10.3.14 updateTime()

```
void videoStore::updateTime ( )
```

Updates the time at a particular rate to create a simulation. Used later in the main loop to keep updating the time with every iteration of the loop.

The documentation for this class was generated from the following files:

- include/[videoStore.h](#)
- src/[videoStore.cpp](#)



## Chapter 5

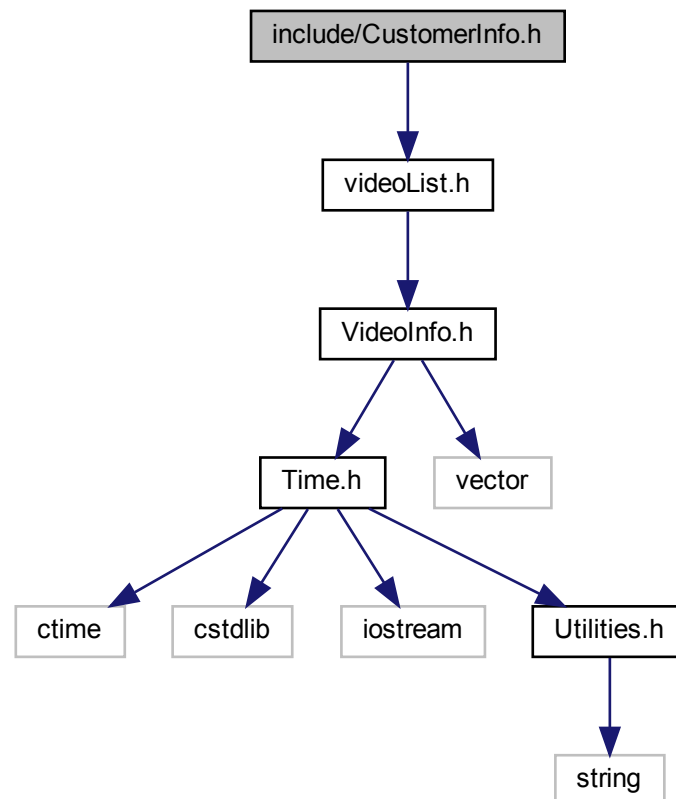
# File Documentation

### 5.1 include/CustomerInfo.h File Reference

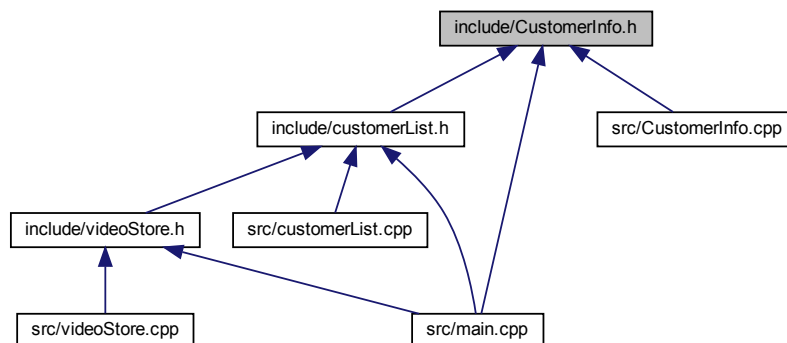
[CustomerInfo](#) class is defined here.

```
#include "videoList.h"
```

Include dependency graph for CustomerInfo.h:



This graph shows which files directly or indirectly include this file:



## Classes

- class [CustomerInfo](#)

*Represents a customer of the store object. Stores basic information about the customer such as their name. Assigns them a random but unique ID.*

### 5.1.1 Detailed Description

[CustomerInfo](#) class is defined here.

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk) )

Version

0.1

Date

2022-01-08

Copyright

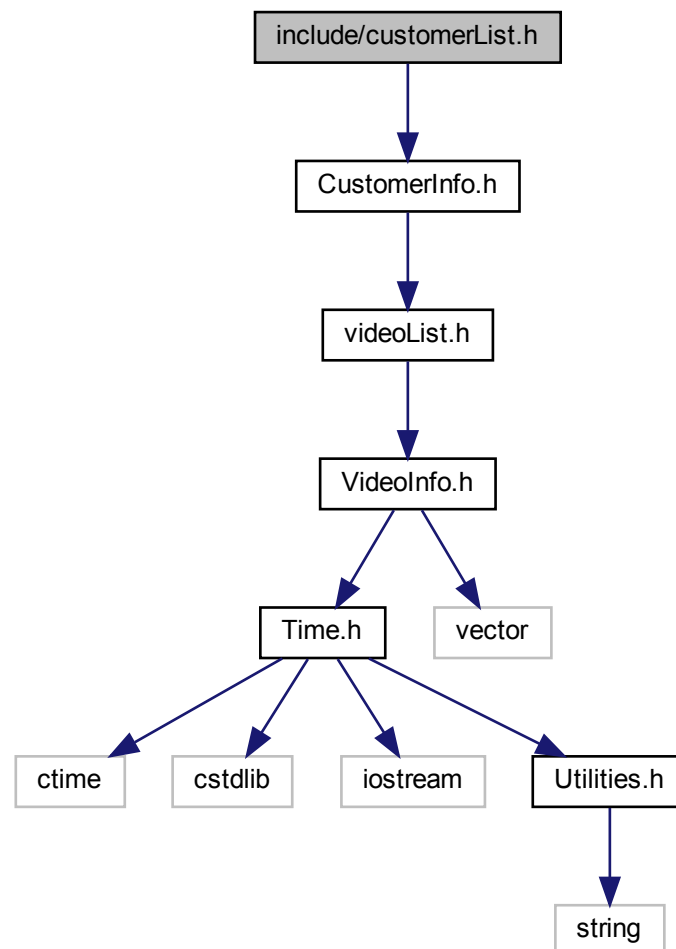
Copyright (c) 2022

## 5.2 include/customerList.h File Reference

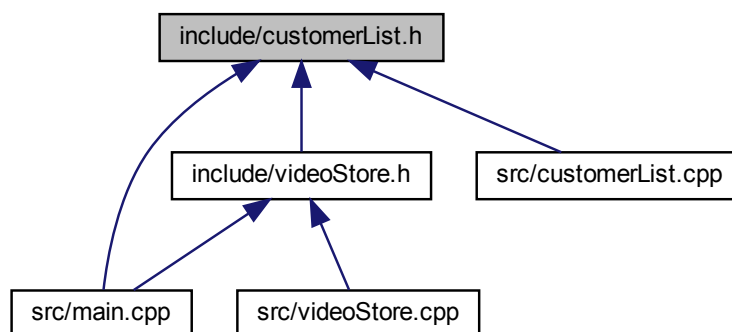
Implements the class simulating the database for [videoStore](#) in the form of a linked list.

```
#include "CustomerInfo.h"
```

Include dependency graph for customerList.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [Customer](#)

*Represents a node in the linked last for customerInfo. Contains the data of the customer and holds the pointer to the next node in the list.*

- class [customerList](#)

*Maintains a linked list for customerInfo objects.Used later in [videoStore](#) to store the list of customers of the shop.*

### 5.2.1 Detailed Description

Implements the class simulating the database for [videoStore](#) in the form of a linked list.

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk))

Version

0.1

Date

2022-01-08

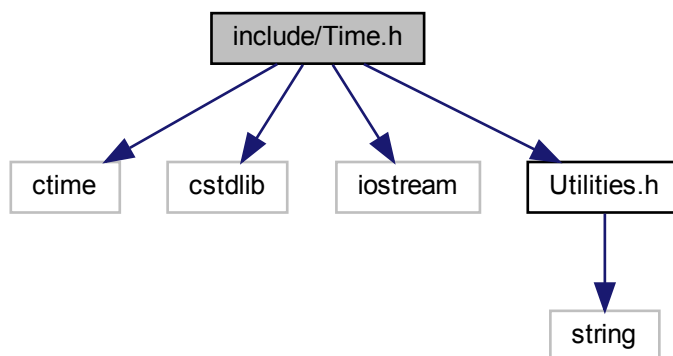
Copyright

Copyright (c) 2022

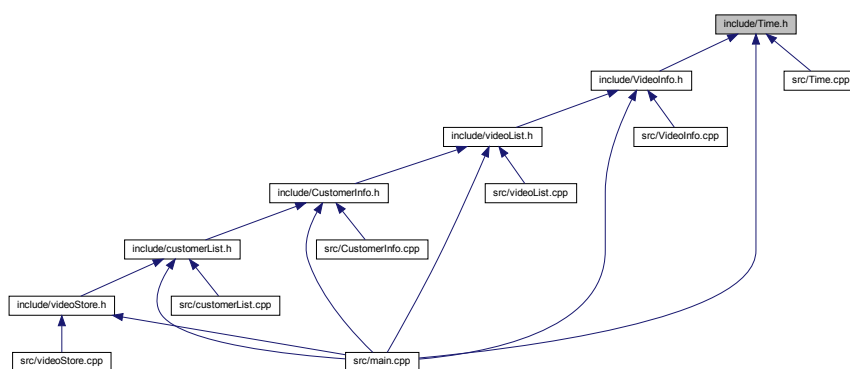
## 5.3 include/Time.h File Reference

Header interface for the struct [Time](#).

```
#include <ctime>
#include <cstdlib>
#include <iostream>
#include "Utilities.h"
Include dependency graph for Time.h:
```



This graph shows which files directly or indirectly include this file:



## Classes

- struct [Time](#)

*A struct to deal with time.*

## Functions

- bool `IsLeapYear` (short year)  
*A utility function that returns whether a given year is a leap year or not.*
- void `convertToUppercase` (std::string &str)  
*A utility function to change a given string such that all its individual letters get uppercased.*

### 5.3.1 Detailed Description

Header interface for the struct `Time`.

Author

Abdul Rafay ( `24100173@lums.edu.pk` )

Version

0.1

Date

2022-01-08

Copyright

Copyright (c) 2022

### 5.3.2 Function Documentation

#### 5.3.2.1 `convertToUppercase()`

```
void convertToUppercase (
    std::string & str )
```

A utility function to change a given string such that all its individual letters get uppercased.

Parameters

<code>str</code>	
------------------	--

#### 5.3.2.2 `IsLeapYear()`

```
bool IsLeapYear (
    short year )
```



A utility function that returns whether a given year is a leap year or not.

Parameters

<i>year</i>	
-------------	--

Returns

true

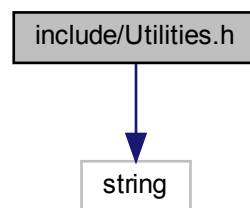
false

## 5.4 include/Utilities.h File Reference

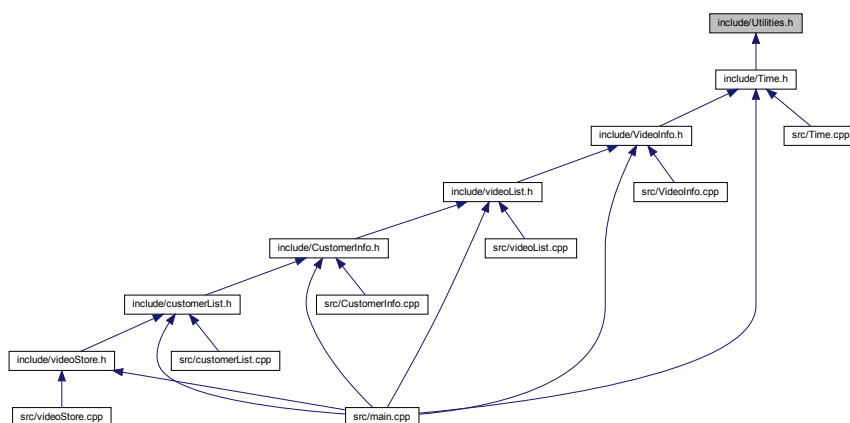
Contains definitions for some useful functions.

```
#include <string>
```

Include dependency graph for Utilities.h:



This graph shows which files directly or indirectly include this file:



## Macros

- `#define _UTILITIES_H`
- `#define CLEARCMD "clear"`

## Enumerations

- `enum STATUS { CLEAR, RENTED }`

*To be used in [CustomerInfo](#) class to represent the account status of the customer.*

## Variables

- `const int PRICE_PER_MOVIE = 1000`
- `const int FINE_PER_DAY = 50`
- `const int SECONDS_PER_MINUTE = 60`
- `const int SECONDS_PER_HOUR = 3600`
- `const int SECONDS_PER_DAY = 86400`
- `const int SECS_IN_WEEK = 60*60*24*7`
- `const int DaysOfMonth [12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}`
- `const std::string Months [12] = {"Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sept", "Oct", "Nov", "Dec"}`

### 5.4.1 Detailed Description

Contains definitions for some useful functions.

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk) )

Version

0.1

Date

2022-01-08

Copyright

Copyright (c) 2022

### 5.4.2 Enumeration Type Documentation

#### 5.4.2.1 STATUS

`enum STATUS`

To be used in [CustomerInfo](#) class to represent the account status of the customer.

Enumerator

CLEAR	Represents that the account status of user is clear and they have nothing rented or overdue.
RENTED	Represents that the user have rented something.

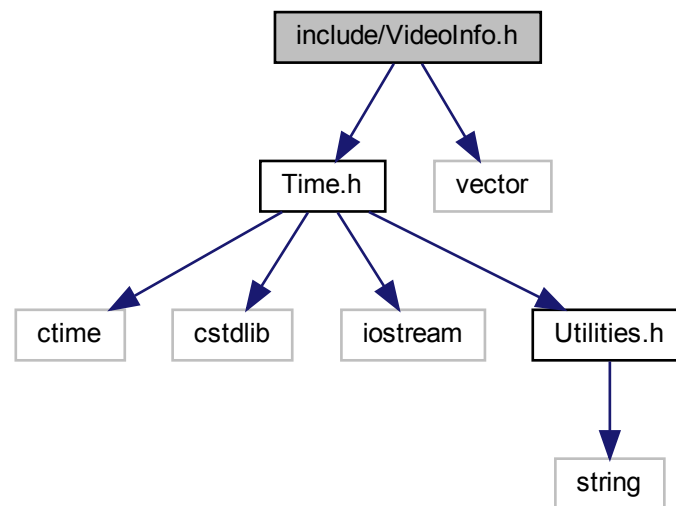
## 5.5 include/VideoInfo.h File Reference

Contains the definition of the class [VideoInfo](#) which is abstraction of a movie.

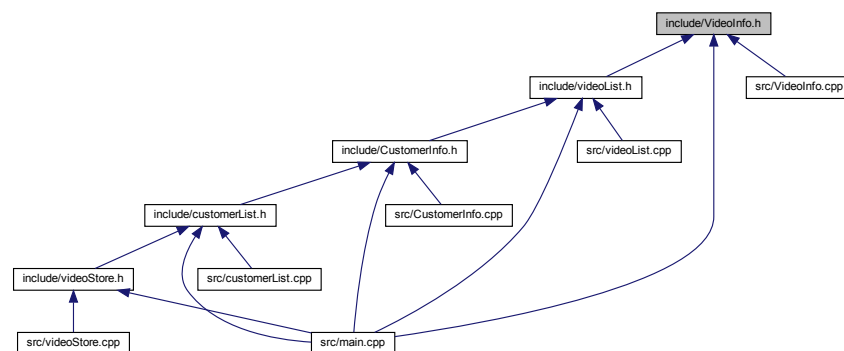
```
#include "Time.h"
```

```
#include <vector>
```

Include dependency graph for VideoInfo.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [Rented](#)

*A struct to hold info about when and who rented a particular movie. An array of [Rented](#) is used inside [VideoInfo](#) to keep track of which copies of a particular movie have been rented out.*

- class [VideoInfo](#)

*Represents a movie present in the store.*

### 5.5.1 Detailed Description

Contains the definition of the class [VideoInfo](#) which is abstraction of a movie.

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk))

Version

0.1

Date

2022-01-08

Copyright

Copyright (c) 2022

## 5.6 include/videoList.h File Reference

```
#include "VideoInfo.h"
```



### 5.6.1 Detailed Description

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk))

Version

0.1

Date

2022-01-08

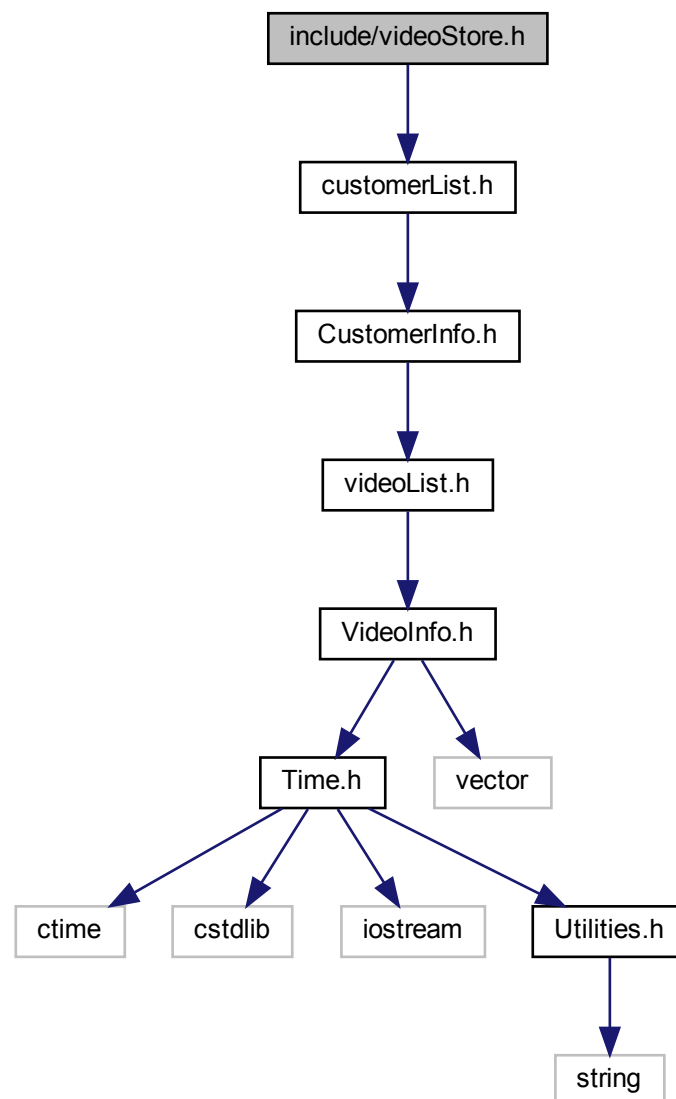
Copyright

Copyright (c) 2022

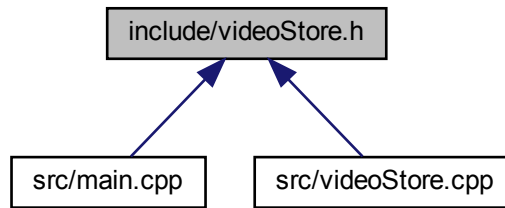
## 5.7 include/videoStore.h File Reference

```
#include "customerList.h"
```

Include dependency graph for videoStore.h:



This graph shows which files directly or indirectly include this file:



## Classes

- struct [Revenue](#)

*Holds the revenue generated on a particular day/Time. The timestamps are represented by the [Time](#) object and the amount generated is stored as well. Also represents a node in the linked list that holds [Revenue](#) objects.*

- class [videoStore](#)

*Main class that represents our video store. Holds the methods to implement the core functionality of the store.*

### 5.7.1 Detailed Description

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk) )

Version

0.1

Date

2022-01-08

Copyright

Copyright (c) 2022

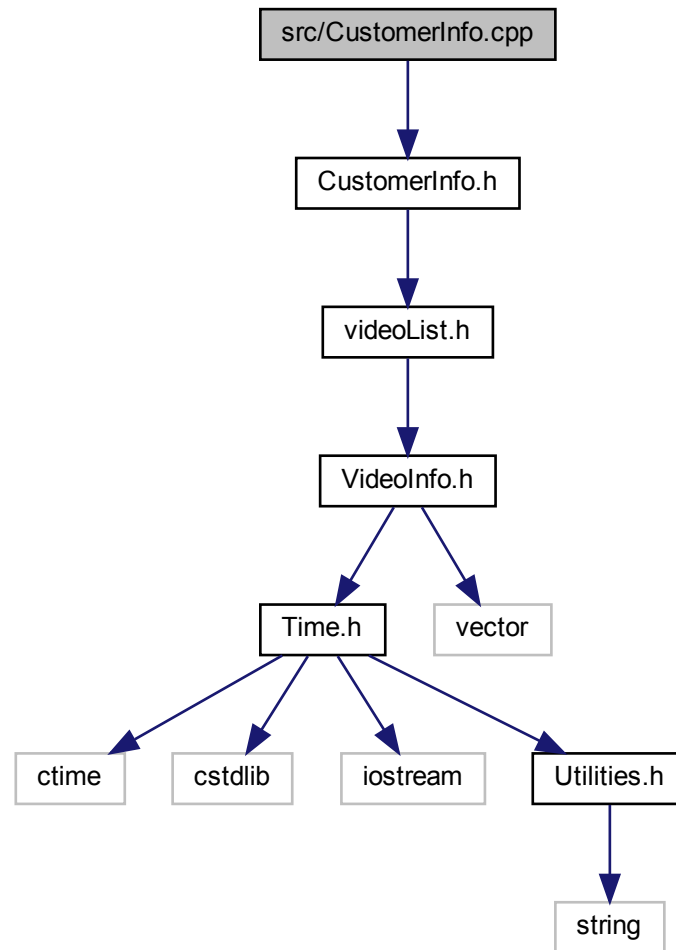


## 5.8 src/CustomerInfo.cpp File Reference

`CustomerInfo` class is implemented.

```
#include "CustomerInfo.h"
```

Include dependency graph for CustomerInfo.cpp:



### 5.8.1 Detailed Description

`CustomerInfo` class is implemented.

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk) )

Version

0.1

Date

2022-01-08

Copyright

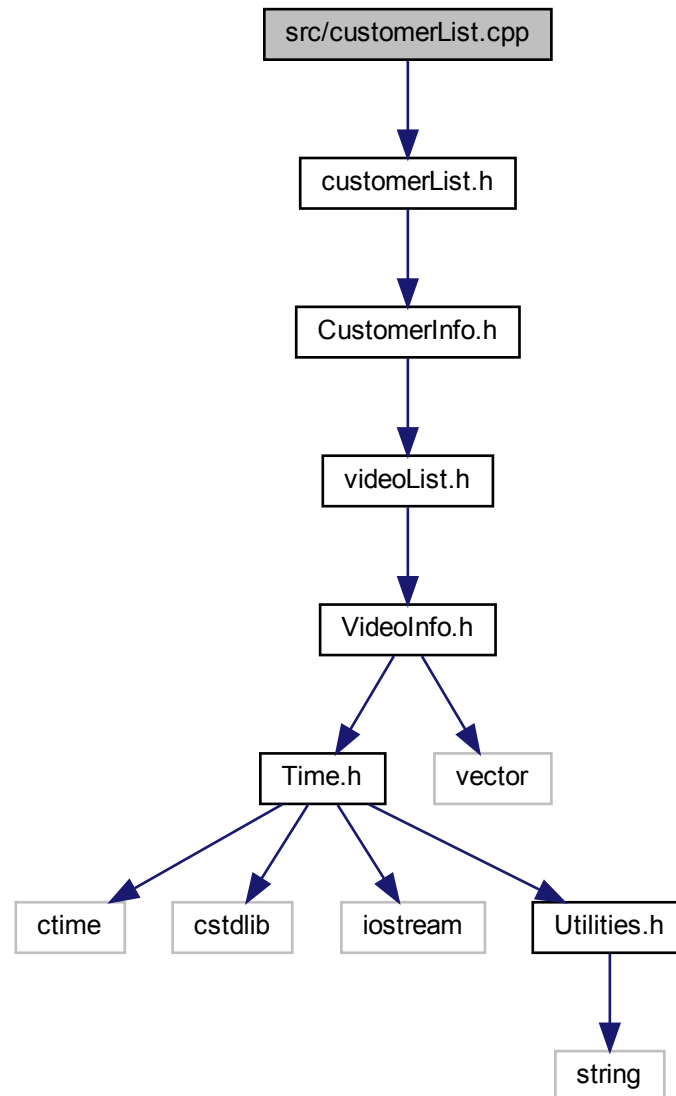
Copyright (c) 2022

## 5.9 src/customerList.cpp File Reference

Defines and implements the [customerList](#) class.

```
#include "customerList.h"
```

Include dependency graph for customerList.cpp:



### 5.9.1 Detailed Description

Defines and implements the `customerList` class.

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk) )

Version

0.1

Date

2022-01-08

Copyright

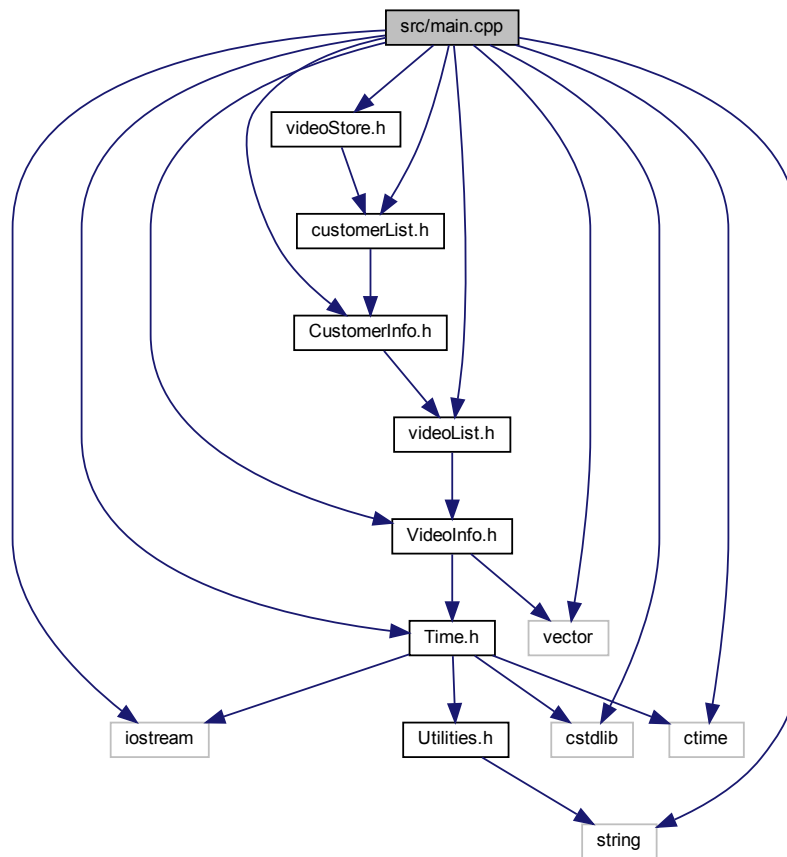
Copyright (c) 2022

## 5.10 src/main.cpp File Reference

The file contains the main function, the starting point of the program. Employs all the classes and structs to simulate a video store with a minimal functionality.

```
#include <iostream>
#include <string>
#include <cstdlib>
#include <ctime>
#include <vector>
#include "Time.h"
#include "VideoInfo.h"
#include "videoList.h"
#include "CustomerInfo.h"
#include "customerList.h"
#include "videoStore.h"
```

Include dependency graph for main.cpp:



## Functions

- `int main ()`

### 5.10.1 Detailed Description

The file contains the main function, the starting point of the program. Employs all the classes and structs to simulate a video store with a minimal functionality.

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk))

Version

0.1

Date

2021-12-12

Copyright

Copyright (c) 2021

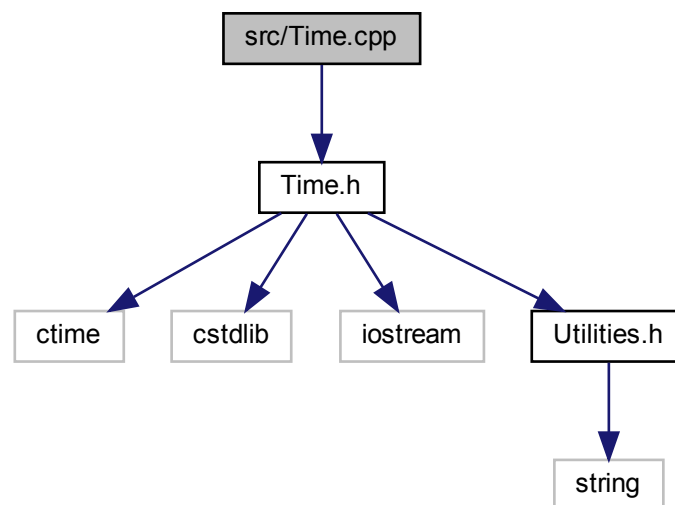
**Bug** No known bugs

## 5.11 src/Time.cpp File Reference

Contains the implementation of the struct class [Time](#).

```
#include "Time.h"
```

Include dependency graph for Time.cpp:



### Functions

- bool [IsLeapYear](#) (short year)  
*A utility function that returns whether a given year is a leap year or not.*
- void [convertToUppercase](#) (std::string &str)  
*A utility function to change a given string such that all its individual letters get uppercased.*

### 5.11.1 Detailed Description

Contains the implementation of the struct class [Time](#).

Author

Abdul Rafay

Version

0.1

Date

2022-01-08

Copyright

Copyright (c) 2022

### 5.11.2 Function Documentation

#### 5.11.2.1 convertToUppercase()

```
void convertToUppercase (
    std::string & str )
```

A utility function to change a given string such that all its individual letters get uppercased.

Parameters

<i>str</i>	
------------	--

#### 5.11.2.2 IsLeapYear()

```
bool IsLeapYear (
    short year )
```

A utility function that returns whether a given year is a leap year or not.

Parameters

<i>year</i>	
-------------	--

Returns

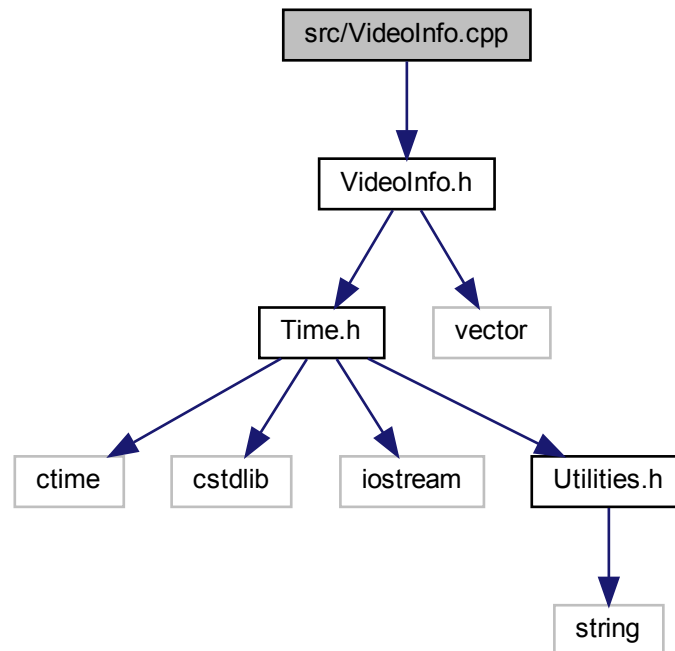
true  
false

## 5.12 src/VideoInfo.cpp File Reference

Implementation of [VideoInfo](#) class.

```
#include "VideoInfo.h"
```

Include dependency graph for VideoInfo.cpp:



### 5.12.1 Detailed Description

Implementation of [VideoInfo](#) class.

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk) )

Version

0.1



Date

2022-01-08

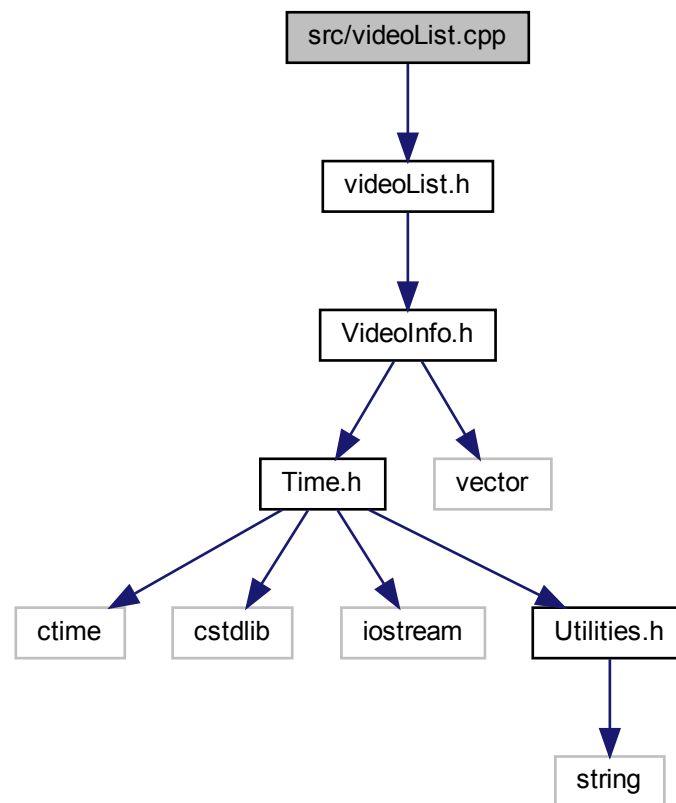
Copyright

Copyright (c) 2022

## 5.13 src/videoList.cpp File Reference

```
#include "videoList.h"
```

Include dependency graph for videoList.cpp:



### 5.13.1 Detailed Description

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk) )

Version

0.1

Date

2022-01-08

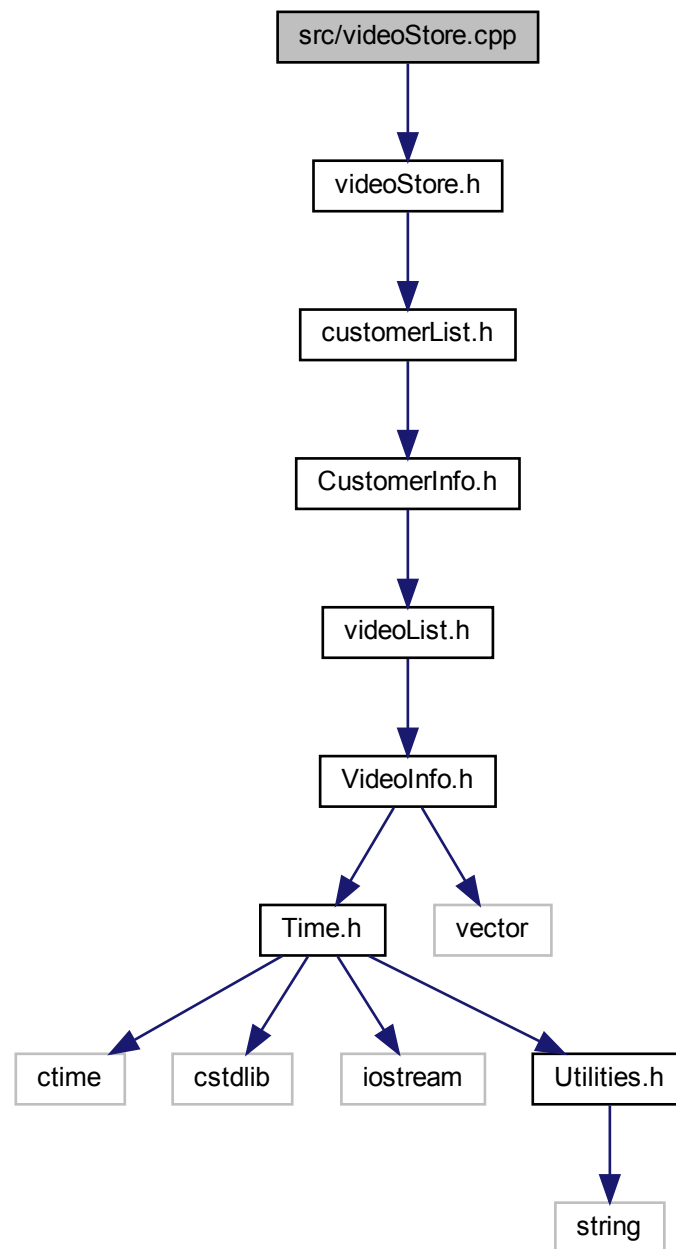
Copyright

Copyright (c) 2022

## 5.14 src/videoStore.cpp File Reference

```
#include "videoStore.h"
```

Include dependency graph for videoStore.cpp:



### 5.14.1 Detailed Description

Author

Abdul Rafay ( [24100173@lums.edu.pk](mailto:24100173@lums.edu.pk) )

Version

0.1

Date

2022-01-08

Copyright

Copyright (c) 2022

# Index

- addCustomer
  - customerList, [14](#)
- addCustomers
  - videoStore, [38](#)
- addMovie
  - videoStore, [38](#)
- addRevenue
  - videoStore, [38](#)
- addVideo
  - videoList, [34](#)
- checkAvailability
  - videoStore, [39](#)
- checkDate
  - Time, [21](#)
- CLEAR
  - Utilities.h, [51](#)
- convertToUppercase
  - Time.cpp, [63](#)
  - Time.h, [48](#)
- cust
  - Customer, [8](#)
- Customer, [7](#)
  - cust, [8](#)
  - Customer, [8](#)
  - next, [8](#)
- customerID
  - Rented, [16](#)
- CustomerInfo, [9](#)
  - CustomerInfo, [9](#)
  - displayInfo, [10](#)
  - getAccStatus, [10](#)
  - getFName, [10](#)
  - getID, [10](#)
  - getLName, [10](#)
  - getNumberOfRented, [11](#)
  - getRentedList, [11](#)
  - setAccStatus, [11](#)
  - setFName, [11](#)
  - setID, [12](#)
  - setLName, [12](#)
  - setNumberOfRented, [12](#)
- customerList, [13](#)
  - addCustomer, [14](#)
  - customerList, [13](#)
  - displaylist, [14](#)
  - getHead, [14](#)
  - operator[], [14](#)
  - searchCustomer, [15](#)
- deleteCasette
  - videoStore, [39](#)
- displayDetails
  - VideoInfo, [29](#)
- displayInfo
  - CustomerInfo, [10](#)
- displaylist
  - customerList, [14](#)
- dueDate
  - Rented, [17](#)
- getAccStatus
  - CustomerInfo, [10](#)
- getAvailableCopies
  - VideoInfo, [29](#)
- getCurrentTime
  - videoStore, [39](#)
- getCustList
  - videoStore, [39](#)
- getDaysTill
  - Time, [21](#)
- getFName
  - CustomerInfo, [10](#)
- getHead
  - customerList, [14](#)
  - videoList, [34](#)
- getID
  - CustomerInfo, [10](#)
- getLName
  - CustomerInfo, [10](#)
- getMovieDirector
  - VideoInfo, [29](#)
- getNextWeek
  - Time, [22](#)
- getNumberOfRented
  - CustomerInfo, [11](#)
- getOutStandingAmount
  - videoStore, [40](#)
- getProtagonist
  - VideoInfo, [30](#)
- getRentedCopies
  - VideoInfo, [30](#)
- getRentedList
  - CustomerInfo, [11](#)
- getRevenueGenerated
  - videoStore, [40](#)
- getTotalCopies
  - VideoInfo, [30](#)
- getVideoTitle
  - VideoInfo, [30](#)

- getVidList
  - videoStore, 40
- include/CustomerInfo.h, 43
- include/customerList.h, 45
- include/Time.h, 47
- include/Utilities.h, 49
- include/VideoInfo.h, 51
- include/videoList.h, 52
- include/videoStore.h, 54
- IsLeapYear
  - Time.cpp, 63
  - Time.h, 48
- next
  - Customer, 8
  - Revenue, 18
  - Video, 27
- numberOfCopies
  - VideoInfo, 31
- operator<
  - Time, 22
- operator<=
  - Time, 22
- operator>
  - Time, 23
- operator>=
  - Time, 23
- operator=
  - Time, 23
  - VideoInfo, 31
- operator[]
  - customerList, 14
  - videoList, 34
- print
  - Time, 25
- printCheckedInMovies
  - videoList, 35
- printCheckedOut
  - videoList, 35
- printDate
  - Time, 25
- printMovies
  - videoList, 35
- removeVideo
  - videoList, 36
- RENTED
  - Utilities.h, 51
- Rented, 15
  - customerID, 16
  - dueDate, 17
  - Rented, 16
  - rentedON, 17
- rentedON
  - Rented, 17
- rentToCustomer
  - videoStore, 40
- returnMovie
  - videoStore, 41
- Revenue, 17
  - next, 18
  - Revenue, 18
- searchCustomer
  - customerList, 15
- searchParticular
  - videoList, 36
- searchVideo
  - videoList, 36
- setAccStatus
  - CustomerInfo, 11
- setAvailableCopies
  - VideoInfo, 31
- setFName
  - CustomerInfo, 11
- setID
  - CustomerInfo, 12
- setLName
  - CustomerInfo, 12
- setMovieDirector
  - VideoInfo, 31
- setNumberOfRented
  - CustomerInfo, 12
- setProtagonist
  - VideoInfo, 32
- setTotalCopies
  - VideoInfo, 32
- setVideoTitle
  - VideoInfo, 32
- showDetails
  - videoStore, 41
- src/CustomerInfo.cpp, 57
- src/customerList.cpp, 58
- src/main.cpp, 60
- src/Time.cpp, 62
- src/VideoInfo.cpp, 64
- src/videoList.cpp, 65
- src/videoStore.cpp, 66
- STATUS
  - Utilities.h, 50
- Time, 19
  - checkDate, 21
  - getDaysTill, 21
  - getNextWeek, 22
  - operator<, 22
  - operator<=, 22
  - operator>, 23
  - operator>=, 23
  - operator=, 23
  - print, 25
  - printDate, 25
  - Time, 20
  - tsecs, 25
- Time.cpp

- convertToUppercase, [63](#)
  - IsLeapYear, [63](#)
- Time.h
  - convertToUppercase, [48](#)
  - IsLeapYear, [48](#)
- tsecs
  - Time, [25](#)
- updateTime
  - videoStore, [41](#)
- Utilities.h
  - CLEAR, [51](#)
  - RENTED, [51](#)
  - STATUS, [50](#)
- vid
  - Video, [27](#)
- Video, [26](#)
  - next, [27](#)
  - vid, [27](#)
  - Video, [26, 27](#)
- VideoInfo, [27](#)
  - displayDetails, [29](#)
  - getAvailableCopies, [29](#)
  - getMovieDirector, [29](#)
  - getProtagonist, [30](#)
  - getRentedCopies, [30](#)
  - getTotalCopies, [30](#)
  - getVideoTitle, [30](#)
  - numberOfCopies, [31](#)
  - operator=, [31](#)
  - setAvailableCopies, [31](#)
  - setMovieDirector, [31](#)
  - setProtagonist, [32](#)
  - setTotalCopies, [32](#)
  - setVideoTitle, [32](#)
  - VideoInfo, [28, 29](#)
- videoList, [33](#)
  - addVideo, [34](#)
  - getHead, [34](#)
  - operator[], [34](#)
  - printCheckedInMovies, [35](#)
  - printCheckedOut, [35](#)
  - printMovies, [35](#)
  - removeVideo, [36](#)
  - searchParticular, [36](#)
  - searchVideo, [36](#)
  - videoList, [34](#)
- videoStore, [37](#)
  - addCustomers, [38](#)
  - addMovie, [38](#)
  - addRevenue, [38](#)
  - checkAvailability, [39](#)
  - deleteCasette, [39](#)
  - getCurrentTime, [39](#)
  - getCustList, [39](#)
  - getOutStandingAmount, [40](#)
  - getRevenueGenerated, [40](#)
  - getVidList, [40](#)
  - rentToCustomer, [40](#)
  - returnMovie, [41](#)
  - showDetails, [41](#)
  - updateTime, [41](#)
  - videoStore, [38](#)