Office Rental System

Generated by Doxygen 1.11.0

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 client Class Reference

represents a client management class

`#include <client.h>`

Inheritance diagram for client:

Collaboration diagram for client:

Public Member Functions

- client (int clientId)
  Constructor for the client class.
- ~client ()
  Destroy the client object.
- void addClient (clientData data, bool current=false)
  Adds a new client to the management system.
- void addRentedSpace (officeInformation data)
  Adds information about an office rented by a client.
- clientData getClient (int clientId)
  Retrieves information about a specific client.
- void changeClient (int clientId)
  Changes the information of a specific client.
- void removeClient (int clientId)
  Removes a client from the management system.
- void printClients ()
  Prints information about all clients in the system.

Protected Attributes

- int clientId = 0
- fileHandling file = fileHandling("clients.csv")

4.1.1 Detailed Description

represents a client management class

manages client data and operations using a linked list structure

Definition at line 15 of file client.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 client()

```
client::client (
            int clientId)
```

Constructor for the client class.

Parameters

clientId	ID of the client being managed

Definition at line 13 of file client.cpp.

Here is the call graph for this function:

4.1.2.2 ~client()

```
client::~client ()
```

Destroy the client object.

Definition at line 99 of file client.cpp.

4.1.3 Member Function Documentation

4.1.3.1 addClient()

```
void client::addClient (
            clientData data,
            bool current = false)
```

Adds a new client to the management system.

Parameters

data	Information about the client to be added
current	Indicates if the client is the currently logged-in client

Definition at line 37 of file client.cpp.

Here is the call graph for this function: Here is the caller graph for this function:

4.1.3.2 addRentedSpace()

```
void client::addRentedSpace (
            officeInformation data)
```

Adds information about an office rented by a client.

Parameters

data	Information about the office being rented

Definition at line 88 of file client.cpp.

Here is the call graph for this function:

4.1.3.3 changeClient()

```
void client::changeClient (
            int clientId)
```

Changes the information of a specific client.

Parameters

clientId	ID of the client to change information for

Definition at line 89 of file client.cpp.

Here is the call graph for this function:

4.1.3.4 getClient()

```
clientData client::getClient (
            int clientId)
```

Retrieves information about a specific client.

Parameters

clientId	ID of the client to retrieve information for

Returns

clientData Information about the requested client

Definition at line 45 of file client.cpp.

Here is the caller graph for this function:

4.1.3.5 printClients()

```
void client::printClients ()
```

Prints information about all clients in the system.

Definition at line 78 of file client.cpp.

Here is the call graph for this function:

4.1.3.6 removeClient()

```
void client::removeClient (
            int clientId)
```

Removes a client from the management system.

remove a client from the client list

Parameters

data	Information about the client to be removed
client↩ Id	

Definition at line 58 of file client.cpp.

Here is the call graph for this function:

4.1.4 Member Data Documentation

4.1.4.1 clientId

```
int client::clientId = 0  [protected]
```

Definition at line 17 of file client.h.

4.1.4.2 file

```
fileHandling client::file = fileHandling("clients.csv")  [protected]
```

Definition at line 18 of file client.h.

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

- ADT/client.h
- ADT/client.cpp

4.2 clientData Struct Reference

Struct defining information about a client.

```
#include <structData.h>
```

Collaboration diagram for clientData:

Public Attributes

- int id
- std::string clientName
- std::string clientAddress
- bool isAdmin
- LinkedList< officeInformation > rentedSpaces

4.2.1 Detailed Description

Struct defining information about a client.

This struct contains details about a client, including ID name, address, administrator status, and list of rented office spaces

Definition at line 28 of file structData.h.

4.2.2 Member Data Documentation

4.2.2.1 clientAddress

```
std::string clientData::clientAddress
```

Definition at line 31 of file structData.h.

4.2.2.2 clientName

```
std::string clientData::clientName
```

Definition at line 30 of file structData.h.

4.2.2.3 id

```
int clientData::id
```

Definition at line 29 of file structData.h.

4.2.2.4 isAdmin

```
bool clientData::isAdmin
```

Definition at line 32 of file structData.h.

4.2.2.5 rentedSpaces

```
LinkedList<officeInformation> clientData::rentedSpaces
```

Definition at line 33 of file structData.h.

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

- ADT/structData.h

4.3 clientRent Class Reference

Represents a client rental management class.

`#include <clientRent.h>`

Inheritance diagram for clientRent:

Collaboration diagram for clientRent:

Public Member Functions

- clientRent (int clientID)
  Constructor for the clientRent class.
- bool rentOffice (int officeId)
  Rents an office for the client.
- void showRentedOffices ()
  Shows available offices that can be rented.
- ~clientRent ()
  Destroy the client Rent object.

Public Member Functions inherited from client

- client (int clientId)
  Constructor for the client class.
- ~client ()
  Destroy the client object.
- void addClient (clientData data, bool current=false)
  Adds a new client to the management system.
- void addRentedSpace (officeInformation data)
  Adds information about an office rented by a client.
- clientData getClient (int clientId)
  Retrieves information about a specific client.
- void changeClient (int clientId)
  Changes the information of a specific client.
- void removeClient (int clientId)
  Removes a client from the management system.
- void printClients ()
  Prints information about all clients in the system.

Public Member Functions inherited from LinkedList< clientRentData >

- LinkedList ()
  Constructor to initialize the linked list.
- Node * getHead ()
- ~LinkedList ()
  Destructor to clean up memory allocated for the linked list.
- void add (clientRentData dataStruct)
  Adds a new node with given data to the end of the linked list.
- void remove (clientRentData data)
  Removes the node with given data from the linked list.
- void print ()
  Prints all elements in the linked list.
- int getSize ()
  Returns the current size of the linked list.
- void clean ()
  Cleans up all nodes in the linked list.

Additional Inherited Members

Protected Attributes inherited from client

- int clientId = 0
- fileHandling file = fileHandling("clients.csv")

Protected Attributes inherited from LinkedList< clientRentData >

- Node * head
- Node * tail
- int size

4.3.1 Detailed Description

Represents a client rental management class.

Handles renting offices by clients and checking the rental status of offices

Definition at line 15 of file clientRent.h.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 clientRent()

```
clientRent::clientRent (
            int clientID)
```

Constructor for the clientRent class.

Parameters

clientId	ID of the client using the rental management

Definition at line 13 of file clientRent.cpp.

Here is the call graph for this function:

4.3.2.2 ~clientRent()

```
clientRent::~clientRent ()
```

Destroy the client Rent object.

Definition at line 62 of file clientRent.cpp.

4.3.3 Member Function Documentation

4.3.3.1 rentOffice()

```
bool clientRent::rentOffice (
            int officeId)
```

Rents an office for the client.

Parameters

officeId	ID of the office to rent

Returns

true If the office was successfully rented

false Otherwise

Definition at line 35 of file clientRent.cpp.

Here is the call graph for this function:

4.3.3.2 showRentedOffices()

```
void clientRent::showRentedOffices ()
```

Shows available offices that can be rented.

Definition at line 46 of file clientRent.cpp.

Here is the call graph for this function:

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

- ADT/clientRent.h
- ADT/clientRent.cpp

4.4 clientRentData Struct Reference

```
#include <structData.h>
```

Public Attributes

- int clientId
- int officeId

4.4.1 Detailed Description

Definition at line 36 of file structData.h.

4.4.2 Member Data Documentation

4.4.2.1 clientId

```
int clientRentData::clientId
```

Definition at line 37 of file structData.h.

4.4.2.2 officeId

```
int clientRentData::officeId
```

Definition at line 38 of file structData.h.

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

- ADT/structData.h

4.5 fileHandling Class Reference

Class for handling file operations such as reading from and writing to files.

```
#include <FileHandling.h>
```

Public Member Functions

- fileHandling (std::string filename)
  Constructor to initialize the file handler with a specified filename.
- ~fileHandling ()
  Destructor to clean up resources associated with the file handler.
- template<typename T >
  void writeToFile (const T &last)
  Writes data to the file.
- template<typename... Args>
  void writeToFile (const Args &... args)
  Writes multiple arguments to the file.
- LinkedList< std::string > readFromFile ()
  Reads data from the file into a linked list.

Public Attributes

- int size

4.5.1 Detailed Description

Class for handling file operations such as reading from and writing to files.

Definition at line 15 of file FileHandling.h.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 fileHandling()

```
fileHandling::fileHandling (
            std::string filename)
```

Constructor to initialize the file handler with a specified filename.

Parameters

filename	Name of the file to be handled

Definition at line 9 of file FileHandling.cpp.

4.5.2.2 ~fileHandling()

```
fileHandling::~fileHandling ()
```

Destructor to clean up resources associated with the file handler.

Definition at line 12 of file FileHandling.cpp.

4.5.3 Member Function Documentation

4.5.3.1 readFromFile()

```
LinkedList< std::string > fileHandling::readFromFile ()  [inline]
```

Reads data from the file into a linked list.

Returns

LinkedList<std::string> A linked list containing the data read from the file

Definition at line 93 of file FileHandling.h.

Here is the call graph for this function: Here is the caller graph for this function:

4.5.3.2 writeToFile() [1/2]

```
template<typename...  Args>
void fileHandling::writeToFile (
            const Args &...  args)  [inline]
```

Writes multiple arguments to the file.

Template Parameters

Args	Types of the arguments to write

Parameters

args	Arguments to write to the file

Definition at line 77 of file FileHandling.h.

4.5.3.3 writeToFile() [2/2]

```
template<typename T >
void fileHandling::writeToFile (
            const T & last)  [inline]
```

Writes data to the file.

Template Parameters

Type	of the last argument to write

Parameters

last	Last argument to write to the file

Definition at line 63 of file FileHandling.h.

Here is the caller graph for this function:

4.5.4 Member Data Documentation

4.5.4.1 size

```
int fileHandling::size
```

Definition at line 49 of file FileHandling.h.

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

- includes/FileHandling.h
- includes/FileHandling.cpp

4.6 LinkedList< T > Class Template Reference

```
#include <LinkedList.h>
```

Collaboration diagram for LinkedList< T >:

Classes

- struct Node

Public Member Functions

- LinkedList ()
  Constructor to initialize the linked list.
- Node * getHead ()
- ~LinkedList ()
  Destructor to clean up memory allocated for the linked list.
- void add (T dataStruct)
  Adds a new node with given data to the end of the linked list.
- void remove (T data)
  Removes the node with given data from the linked list.
- void print ()
  Prints all elements in the linked list.
- int getSize ()
  Returns the current size of the linked list.
- void clean ()
  Cleans up all nodes in the linked list.

Protected Attributes

- Node * head
- Node * tail
- int size

4.6.1 Detailed Description

template<typename T>
class LinkedList< T >

Definition at line 5 of file LinkedList.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 LinkedList()

```
template<typename T >
LinkedList< T >::LinkedList ()  [inline]
```

Constructor to initialize the linked list.

Definition at line 20 of file LinkedList.h.

4.6.2.2 ~LinkedList()

```
template<typename T >
LinkedList< T >::~LinkedList ()  [inline]
```

Destructor to clean up memory allocated for the linked list.

Definition at line 33 of file LinkedList.h.

4.6.3 Member Function Documentation

4.6.3.1 add()

```
template<typename T >
void LinkedList< T >::add (
            T dataStruct)  [inline]
```

Adds a new node with given data to the end of the linked list.

Parameters

dataStruct	Data to be stored in the new node

Definition at line 49 of file LinkedList.h.

Here is the caller graph for this function:

4.6.3.2 clean()

```
template<typename T >
void LinkedList< T >::clean ()  [inline]
```

Cleans up all nodes in the linked list.

Deletes all nodes to free memory and resets the head to nullptr

Definition at line 114 of file LinkedList.h.

4.6.3.3 getHead()

```
template<typename T >
Node * LinkedList< T >::getHead ()  [inline]
```

Definition at line 25 of file LinkedList.h.

Here is the caller graph for this function:

4.6.3.4 getSize()

```
template<typename T >
int LinkedList< T >::getSize ()  [inline]
```

Returns the current size of the linked list.

Returns

int Size of the linked list

Definition at line 104 of file LinkedList.h.

Here is the caller graph for this function:

4.6.3.5 print()

```
template<typename T >
void LinkedList< T >::print ()  [inline]
```

Prints all elements in the linked list.

Definition at line 91 of file LinkedList.h.

4.6.3.6 remove()

```
template<typename T >
void LinkedList< T >::remove (
            T data)  [inline]
```

Removes the node with given data from the linked list.

Parameters

data	Data of the node to be removed

Definition at line 68 of file LinkedList.h.

4.6.4 Member Data Documentation

4.6.4.1 head

```
template<typename T >
Node* LinkedList< T >::head  [protected]
```

Definition at line 11 of file LinkedList.h.

4.6.4.2 size

```
template<typename T >
int LinkedList< T >::size  [protected]
```

Definition at line 13 of file LinkedList.h.

4.6.4.3 tail

```
template<typename T >
Node* LinkedList< T >::tail  [protected]
```

Definition at line 12 of file LinkedList.h.

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

- includes/LinkedList.h

4.7 LinkedList< T >::Node Struct Reference

```
#include <LinkedList.h>
```

Collaboration diagram for LinkedList< T >::Node:

Public Attributes

- T data
- Node * next

4.7.1 Detailed Description

template<typename T>
struct LinkedList< T >::Node

Definition at line 7 of file LinkedList.h.

4.7.2 Member Data Documentation

4.7.2.1 data

```
template<typename T >
T LinkedList< T >::Node::data
```

Definition at line 8 of file LinkedList.h.

4.7.2.2 next

```
template<typename T >
Node* LinkedList< T >::Node::next
```

Definition at line 9 of file LinkedList.h.

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

- includes/LinkedList.h

4.8 office Class Reference

Represents an office management class.

`#include <office.h>`

Inheritance diagram for office:

Collaboration diagram for office:

Public Member Functions

- office (int clientID)
  Constructor for the office class.
- void addOffice (officeInformation data)
  Adds a new office to the management system.
- void rentOffice (officeInformation data, int clientID)
  Rents an office to a specified client.
- void endRental (officeInformation data)
  Ends the rental of an office, making it available again.
- void printOffices ()
  prints info about all offices in the system
- officeInformation getOffice (int officeId)
  retrieves detailed info about a specific office
- ~office ()
  Destroy the office object.

Public Member Functions inherited from LinkedList< officeInformation >

- LinkedList ()

  Constructor to initialize the linked list.
- Node * getHead ()
- ~LinkedList ()

  Destructor to clean up memory allocated for the linked list.
- void add (officeInformation dataStruct)

  Adds a new node with given data to the end of the linked list.
- void remove (officeInformation data)

  Removes the node with given data from the linked list.
- void print ()

  Prints all elements in the linked list.
- int getSize ()

  Returns the current size of the linked list.
- void clean ()

  Cleans up all nodes in the linked list.

Additional Inherited Members

Protected Attributes inherited from LinkedList< officeInformation >

- Node * head
- Node * tail
- int size

4.8.1 Detailed Description

Represents an office management class.

class manages office data including rental operations, which used a linked list structure.

Definition at line 12 of file office.h.

4.8.2 Constructor & Destructor Documentation

4.8.2.1 office()

```
office::office (
            int clientID)
```

Constructor for the office class.

Parameters

clientID	ID of the client using the office management system.

Definition at line 12 of file office.cpp.

Here is the call graph for this function:

4.8.2.2 ~office()

```
office::~office ()
```

Destroy the office object.

Definition at line 79 of file office.cpp.

4.8.3 Member Function Documentation

4.8.3.1 addOffice()

```
void office::addOffice (
            officeInformation data)
```

Adds a new office to the management system.

Parameters

data	information about the office to be added

Definition at line 31 of file office.cpp.

Here is the call graph for this function: Here is the caller graph for this function:

4.8.3.2 endRental()

```
void office::endRental (
            officeInformation data)
```

Ends the rental of an office, making it available again.

Parameters

data	Information about the office whose rental is ending

Definition at line 35 of file office.cpp.

Here is the call graph for this function:

4.8.3.3 getOffice()

```
officeInformation office::getOffice (
            int officeId)
```

retrieves detailed info about a specific office

Parameters

office↩ Id	ID of the office to retrieve information for

Returns

officeInformation Information about the requested office

Definition at line 46 of file office.cpp.

Here is the caller graph for this function:

4.8.3.4 printOffices()

```
void office::printOffices ()
```

prints info about all offices in the system

Definition at line 67 of file office.cpp.

Here is the caller graph for this function:

4.8.3.5 rentOffice()

```
void office::rentOffice (
            officeInformation data,
            int clientID)
```

Rents an office to a specified client.

Parameters

data	Information about the office to be rented
clientID	ID of the client renting the office.

Definition at line 56 of file office.cpp.

Here is the call graph for this function:

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

- ADT/office.h
- ADT/office.cpp

4.9 officeInformation Struct Reference

Struct defining information about an office.

```
#include <structData.h>
```

Public Attributes

- int id
- std::string officeName
- std::string officeAddress
- int officePrice
- std::string officeSize
- bool isRented

4.9.1 Detailed Description

Struct defining information about an office.

contains details about an office, including the ID, name, address, rental price, size, and rental status

Definition at line 13 of file structData.h.

4.9.2 Member Data Documentation

4.9.2.1 id

```
int officeInformation::id
```

Definition at line 14 of file structData.h.

4.9.2.2 isRented

```
bool officeInformation::isRented
```

Definition at line 19 of file structData.h.

4.9.2.3 officeAddress

```
std::string officeInformation::officeAddress
```

Definition at line 16 of file structData.h.

4.9.2.4 officeName

```
std::string officeInformation::officeName
```

Definition at line 15 of file structData.h.

4.9.2.5 officePrice

```
int officeInformation::officePrice
```

Definition at line 17 of file structData.h.

4.9.2.6 officeSize

```
std::string officeInformation::officeSize
```

Definition at line 18 of file structData.h.

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

- ADT/structData.h

4.10 Queue< T > Class Template Reference

```
#include <Queue.h>
```

Public Member Functions

- Queue ()
- ~Queue ()
- void enqueue (T dataStruct)
- void dequeue ()
- void print ()
- int getSize ()
- bool isEmpty ()
- T front ()
- T back ()

4.10.1 Detailed Description

template<typename T>
class Queue< T >

Definition at line 4 of file Queue.h.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Queue()

```
template<typename T >
Queue< T >::Queue ()  [inline]
```

Definition at line 15 of file Queue.h.

4.10.2.2 ~Queue()

```
template<typename T >
Queue< T >::~Queue ()  [inline]
```

Definition at line 20 of file Queue.h.

4.10.3 Member Function Documentation

4.10.3.1 back()

```
template<typename T >
T Queue< T >::back ()  [inline]
```

Definition at line 71 of file Queue.h.

4.10.3.2 dequeue()

```
template<typename T >
void Queue< T >::dequeue ()  [inline]
```

Definition at line 42 of file Queue.h.

4.10.3.3 enqueue()

```
template<typename T >
void Queue< T >::enqueue (
            T dataStruct)  [inline]
```

Definition at line 29 of file Queue.h.

4.10.3.4 front()

```
template<typename T >
T Queue< T >::front ()  [inline]
```

Definition at line 65 of file Queue.h.

4.10.3.5 getSize()

```
template<typename T >
int Queue< T >::getSize ()  [inline]
```

Definition at line 59 of file Queue.h.

4.10.3.6 isEmpty()

```
template<typename T >
bool Queue< T >::isEmpty ()  [inline]
```

Definition at line 62 of file Queue.h.

4.10.3.7 print()

```
template<typename T >
void Queue< T >::print ()  [inline]
```

Definition at line 51 of file Queue.h.

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

- TEMP/Queue.h

4.11 Stack< T > Class Template Reference

```
#include <Stack.h>
```

Public Member Functions

- Stack ()
- ~Stack ()
- void push (T dataStruct)
- void pop ()
- void print ()
- int getSize ()

4.11.1 Detailed Description

template<typename T>
class Stack< T >

Definition at line 4 of file Stack.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 Stack()

```
template<typename T >
Stack< T >::Stack ()  [inline]
```

Definition at line 14 of file Stack.h.

4.11.2.2 ~Stack()

```
template<typename T >
Stack< T >::~Stack ()  [inline]
```

Definition at line 18 of file Stack.h.

4.11.3 Member Function Documentation

4.11.3.1 getSize()

```
template<typename T >
int Stack< T >::getSize ()  [inline]
```

Definition at line 51 of file Stack.h.

4.11.3.2 pop()

```
template<typename T >
void Stack< T >::pop ()  [inline]
```

Definition at line 34 of file Stack.h.

4.11.3.3 print()

```
template<typename T >
void Stack< T >::print ()  [inline]
```

Definition at line 43 of file Stack.h.

4.11.3.4 push()

```
template<typename T >
void Stack< T >::push (
            T dataStruct)  [inline]
```

Definition at line 27 of file Stack.h.

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

- TEMP/Stack.h

Chapter 5

File Documentation

5.1 ADT/client.cpp File Reference

```
#include "./client.h"
#include <array>
#include <iostream>
#include <vector>
#include "../includes/FileHandling.h"
#include "../includes/LinkedList.h"
#include "../includes/utils.h"
#include "./office.h"
```

Include dependency graph for client.cpp:

Macros

- #define CLIENT_CPP

5.1.1 Macro Definition Documentation

5.1.1.1 CLIENT_CPP

```
#define CLIENT_CPP
```

Definition at line 2 of file client.cpp.

5.2 client.cpp

Go to the documentation of this file.

```
00001 #ifndef CLIENT_CPP
00002 #define CLIENT_CPP
00003 #include "./client.h"
00004
00005 #include <array>
00006 #include <iostream>
00007 #include <vector>
00008
00009 #include "../includes/FileHandling.h"
```

```
00010 #include "../includes/LinkedList.h"
00011 #include "../includes/utils.h"
00012 #include "./office.h"
00013 client::client(int clientID) {
00014     if (clientID == -1) return;
00015     clientId = clientID;
00016     LinkedList<std::string> data = file.readFromFile();
00017     auto current = data.getHead();
00018     const std::size_t size = 4;
00019     bool isUser = false;
00020     while (current != nullptr) {
00021         std::array<std::string, size> output;
00022         splitData(current->data, ',', output);
00023         current = current->next;
00024         if (clientID == std::stoi(output[0])) {
00025             loggedClient.id = std::stoi(output[0]);
00026             loggedClient.clientName = output[1];
00027             loggedClient.clientAddress = output[2];
00028             loggedClient.isAdmin = output[3] == "1";
00029         }
00030         add({std::stoi(output[0]), output[1], output[2], output[3] == "1"});
00031     }
00032     if (loggedClient.clientName == "") {
00033         std::cout << "Client not found" << std::endl;
00034     }
00035 }
00036
00037 void client::addClient(clientData data, bool current) {
00038     std::cout << data.isAdmin << std::endl;
00039     add(data);
00040     file.writeToFile(data.id, data.clientName, data.clientAddress, data.isAdmin);
00041     if (current) {
00042         clientId = data.id;
00043     };
00044 }
00045 clientData client::getClient(int clientId) {
00046     Node* current = this->head;
00047     while (current != nullptr) {
00048         if (current->data.id == clientId) return current->data;
00049         current = current->next;
00050     }
00051     return {};
00052 }
00058 void client::removeClient(int clientId) {
00059     Node* current = head;
00060     while (current != nullptr) {
00061         if (current->data.id == clientId) {
00062             remove(current->data);
00063             break;
00064         }
00065         current = current->next;
00066     }
00067 }
00068 // void LinkedList<officeInformation>::print() {
00069 //     Node* current = head;
00070 //     while (current != nullptr) {
00071 //         std::cout << "Office ID: " << current->data.officeID << std::endl;
00072 //         std::cout << "Office Size: " << current->data.officeSize << std::endl;
00073 //         std::cout << "Office Price: " << current->data.officePrice << std::endl;
00074 //         std::cout << "Is Office Rented? " << current->data.isRented << std::endl;
00075 //         current = current->next;
00076 //     }
00077 // }
00078 void client::printClients() {
00079     Node* current = head;
00080     while (current != nullptr) {
00081         std::cout << "Client ID: " << current->data.id << std::endl;
00082         std::cout << "Client Name: " << current->data.clientName << std::endl;
00083         std::cout << "Client Phone: " << current->data.clientAddress << std::endl;
00084         std::cout << "Client Rented Space: " << current->data.rentedSpaces.getSize() << std::endl;
00085         current = current->next;
00086     }
00087 }
00088 void client::addRentedSpace(officeInformation data) { head->data.rentedSpaces.add(data); };
00089 void client::changeClient(int clientId) {
00090     Node* current = head;
00091     while (current != nullptr) {
00092         if (current->data.id == clientId) {
00093             current->data = getClient(clientId);
00094             break;
00095         }
00096         current = current->next;
00097     }
00098 }
00099 client::~client() {
00100     std::cout << "Client Deleted" << std::endl;
00101     // if (head != nullptr) {
```

```
00102     //     Node* current = head;
00103     //     while (current != nullptr) {
00104     //         Node* temp = current;
00105     //         current = current->next;
00106     //         delete temp;
00107     //     }
00108     // }
00109 }
00110
00111 #endif
```

5.3 ADT/client.h File Reference

```
#include <string>
#include "../includes/LinkedList.h"
#include "../includes/fileHandling.h"
#include "./structData.h"
```

Include dependency graph for client.h: This graph shows which files directly or indirectly include this file:

Classes

- class client

  represents a client management class

5.4 client.h

Go to the documentation of this file.

```
00001 //* customer parent
00002 #ifndef CLIENT_H
00003 #define CLIENT_H
00004 #include <string>
00005
00006 #include "../includes/LinkedList.h"
00007 #include "../includes/fileHandling.h"
00008 #include "./structData.h"
00009
00015 class client : LinkedList<clientData> {
00016    protected:
00017     int clientId = 0;  // currentlly logged in client
00018     fileHandling file = fileHandling("clients.csv");
00019
00020    private:
00021     clientData loggedClient;
00022     LinkedList<clientData> clientList;
00023
00024    public:
00030     client(int clientId);
00031
00036     ~client();
00037
00044     void addClient(clientData data, bool current = false);
00045
00051     void addRentedSpace(officeInformation data);
00052
00059     clientData getClient(int clientId);
00060
00066     void changeClient(int clientId);
00067
00073     void removeClient(int clientId);
00074
00079     void printClients();
00080 };
00081 #endif
```

5.5 ADT/clientRent.cpp File Reference

```
#include "./clientRent.h"
#include <iostream>
#include "../includes/FileHandling.h"
#include "../includes/utils.h"
#include "./client.h"
#include "./office.h"
```

Include dependency graph for clientRent.cpp:

Macros

- #define OFFICERENTAL_CPP

5.5.1 Macro Definition Documentation

5.5.1.1 OFFICERENTAL_CPP

```
#define OFFICERENTAL_CPP
```

Definition at line 3 of file clientRent.cpp.

5.6 clientRent.cpp

Go to the documentation of this file.

```
00001 //* shows a list of ALL offices.
00002 #ifndef OFFICERENTAL_CPP
00003 #define OFFICERENTAL_CPP
00004 #include "./clientRent.h"
00005
00006 #include <iostream>
00007
00008 #include "../includes/FileHandling.h"
00009 #include "../includes/utils.h"
00010 #include "./client.h"
00011 #include "./office.h"
00012
00013 clientRent::clientRent(int clientID) : client(clientID) {
00014     clientId = clientID;
00015     LinkedList<std::string> data = file.readFromFile();
00016     auto current = data.getHead();
00017     const size_t size = 2;
00018     while (current != nullptr) {
00019         std::array<std::string, size> output;
00020         splitData(current->data, ',', output);
00021         clientRentData office;
00022         // check if output is not empty
00023         if (output[0].empty() || output[1].empty()) {
00024             current = current->next;
00025             continue;
00026         }
00027         office.clientId = std::stoi(output[0]);
00028         office.officeId = std::stoi(output[1]);
00029         if (clientId == office.clientId) {
00030             LinkedList<clientRentData>::add(office);
00031         }
00032         current = current->next;
00033     }
00034 };
00035 bool clientRent::rentOffice(int officeId) {
00036     LinkedList<clientRentData>::Node* current = LinkedList<clientRentData>::head;
00037     while (current != nullptr) {
00038         if (current->data.clientId != officeId) {
00039             current = current->next;
```

```
00040             continue;
00041         }
00042     }
00043     file.writeToFile(clientId, officeId);
00044     return true;
00045 }
00046 void clientRent::showRentedOffices() {
00047     auto currentOffice = office(clientId).getHead();
00048     LinkedList<clientRentData>::Node* current = LinkedList<clientRentData>::getHead();
00049     while (current != nullptr) {
00050         while (currentOffice != nullptr) {
00051             if (current->data.officeId == currentOffice->data.id) {
00052                 std::cout « "office Name: " « currentOffice->data.officeName « std::endl;
00053                 std::cout « "office Address: " « currentOffice->data.officeAddress « std::endl;
00054                 std::cout « "office Price: " « currentOffice->data.officePrice « std::endl;
00055                 std::cout « "office Size: " « currentOffice->data.officeSize « std::endl;
00056             }
00057             currentOffice = currentOffice->next;
00058         }
00059         current = current->next;
00060     }
00061 }
00062 clientRent::~clientRent() {
00063     std::cout « "Client Rent Destructor Called" « std::endl;
00064 }
00065
00066 #endif
```

5.7 ADT/clientRent.h File Reference

```
#include <string>
#include "../includes/LinkedList.h"
#include "./client.h"
#include "./office.h"
```

Include dependency graph for clientRent.h: This graph shows which files directly or indirectly include this file:

Classes

- class clientRent

  Represents a client rental management class.

5.8 clientRent.h

Go to the documentation of this file.

```
00001 //* child class of client. Responsible for renting an office, checking if an office is rented or not
      as well.
00002 #ifndef OFFICERENTAL_H
00003 #define OFFICERENTAL_H
00004 #include <string>
00005
00006 #include "../includes/LinkedList.h"
00007 #include "./client.h"
00008 #include "./office.h"
00009
00015 class clientRent : public client, public LinkedList<clientRentData> {
00016    private:
00017     int clientId;
00018     fileHandling file = fileHandling("rentedOffices.csv");
00019
00020    public:
00026     clientRent(int clientID);
00027
00035     bool rentOffice(int officeId);
00036
00041     void showRentedOffices();
00042
00047     ~clientRent();
00048 };
00049 #endif
```

5.9 ADT/office.cpp File Reference

```
#include "office.h"
#include <array>
#include <iostream>
#include "../includes/LinkedList.h"
#include "../includes/fileHandling.h"
#include "../includes/utils.h"
```

Include dependency graph for office.cpp:

Macros

- #define OFFICE_CPP

5.9.1 Macro Definition Documentation

5.9.1.1 OFFICE_CPP

```
#define OFFICE_CPP
```

Definition at line 3 of file office.cpp.

5.10 office.cpp

Go to the documentation of this file.

```
00001
00002 #ifndef OFFICE_CPP
00003 #define OFFICE_CPP
00004 #include "office.h"
00005
00006 #include <array>
00007 #include <iostream>
00008
00009 #include "../includes/LinkedList.h"
00010 #include "../includes/fileHandling.h"
00011 #include "../includes/utils.h"
00012 office::office(int clientID) : LinkedList() {
00013     LinkedList<std::string> data = file.readFromFile();
00014     const std::size_t size = 6;
00015     std::array<std::string, size> output;
00016     auto current = data.getHead();
00017     while (current != nullptr) {
00018         std::array<std::string, size> output;
00019         splitData(current->data, ',', output);
00020         officeInformation office;
00021         office.id = std::stoi(output[0]);
00022         office.officeName = output[1];
00023         office.officeAddress = output[2];
00024         office.officePrice = std::stoi(output[3]);
00025         office.officeSize = output[4];
00026         office.isRented = output[5] == "1";
00027         add(office);
00028         current = current->next;
00029     }
00030 };
00031 void office::addOffice(officeInformation data) {
00032     add(data);
00033     file.writeToFile(data.id, data.officeName, data.officeAddress, data.officePrice, data.officeSize,
      data.isRented);
00034 };
00035 void office::endRental(officeInformation data) {
00036     LinkedList<officeInformation>::Node* current = head;
00037     while (current != nullptr) {
00038         if (current->data.id != data.id) {
```

```
00039             current = current->next;
00040             continue;
00041         }
00042         current->data.isRented = false;
00043         file.writeToFile(data.id, data.officeName, data.officeAddress, data.officePrice,
      data.officeSize, data.isRented);
00044     }
00045 }
00046 officeInformation office::getOffice(int officeId) {
00047     LinkedList<officeInformation>::Node* current = head;
00048     while (current != nullptr) {
00049         if (current->data.id == officeId) {
00050             return current->data;
00051         }
00052         current = current->next;
00053     }
00054     return officeInformation();
00055 }
00056 void office::rentOffice(officeInformation data, int clientID) {
00057     LinkedList<officeInformation>::Node* current = head;
00058     while (current != nullptr) {
00059         if (current->data.id != data.id) {
00060             current = current->next;
00061             continue;
00062         }
00063         current->data.isRented = true;
00064         file.writeToFile(data.id, clientID);
00065     }
00066 }
00067 void office::printOffices() {
00068     LinkedList<officeInformation>::Node* current = head;
00069     while (current != nullptr) {
00070         std::cout « "Office ID: " « current->data.id « std::endl;
00071         std::cout « "Office Name: " « current->data.officeName « std::endl;
00072         std::cout « "Office Address: " « current->data.officeAddress « std::endl;
00073         std::cout « "Office Price: " « current->data.officePrice « std::endl;
00074         std::cout « "Office Size: " « current->data.officeSize « std::endl;
00075         std::cout « "Office is Rented: " « current->data.isRented « std::endl;
00076         current = current->next;
00077     }
00078 }
00079 office::~office() {
00080     std::cout « "Office Deleted" « std::endl;
00081     if (head != nullptr) return;
00082     delete head;
00083 }
00084 #endif
```

5.11 ADT/office.h File Reference

```
#include "../includes/LinkedList.h"
#include "../includes/fileHandling.h"
#include "./structData.h"
```

Include dependency graph for office.h: This graph shows which files directly or indirectly include this file:

Classes

- class office

  Represents an office management class.

5.12 office.h

Go to the documentation of this file.

```
00001 #ifndef OFFICE_H
00002 #define OFFICE_H
00003 #include "../includes/LinkedList.h"
00004 #include "../includes/fileHandling.h"
00005 #include "./structData.h"
00012 class office : public LinkedList<officeInformation> {
```

```
00013    private:
00018     struct Node {
00019         officeInformation data;
00020         Node* next;
00021     };
00022     fileHandling file = fileHandling("offices.csv");
00023     office();  // Private constructor
00024
00025    public:
00031     office(int clientID);
00032
00038     void addOffice(officeInformation data);
00039
00046     void rentOffice(officeInformation data, int clientID);
00047
00053     void endRental(officeInformation data);
00054
00059     void printOffices();
00060
00067     officeInformation getOffice(int officeId);
00072     ~office();
00073 };
00074 #endif
```

5.13 ADT/structData.h File Reference

```
#include <string>
#include "../includes/LinkedList.h"
```

Include dependency graph for structData.h: This graph shows which files directly or indirectly include this file:

Classes

- struct officeInformation

  Struct defining information about an office.
- struct clientData

  Struct defining information about a client.
- struct clientRentData

5.14 structData.h

Go to the documentation of this file.

```
00001 #ifndef STRUCTDATA_H
00002 #define STRUCTDATA_H
00003 #include <string>
00004 // Data types for the program. Just to have a unified place to store  all data types plus intellisene
      support
00005 #include "../includes/LinkedList.h"
00006
00013 struct officeInformation {
00014     int id;
00015     std::string officeName;
00016     std::string officeAddress;
00017     int officePrice;
00018     std::string officeSize;
00019     bool isRented;
00020 };
00021
00028 struct clientData {
00029     int id;
00030     std::string clientName;
00031     std::string clientAddress;
00032     bool isAdmin;
00033     LinkedList<officeInformation> rentedSpaces;
00034 };
00035
00036 struct clientRentData {
00037     int clientId;
00038     int officeId;
00039 };
00040
00041 #endif
```

5.15 includes/FileHandling.cpp File Reference

```
#include "./FileHandling.h"
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
```

Include dependency graph for FileHandling.cpp:

5.16 FileHandling.cpp

Go to the documentation of this file.

```
00001 #ifndef FILEHANDLING_CPP
00002 #define FILEHANDLING_CPP
00003 #include "./FileHandling.h"
00004
00005 #include <fstream>
00006 #include <iostream>
00007 #include <string>
00008 #include <vector>
00009 fileHandling::fileHandling(std::string filename) {
00010     this->filename = filename;
00011 }
00012 fileHandling::~fileHandling() {
00013     return;
00014 }
00015
00016 #endif
```

5.17 includes/FileHandling.h File Reference

```
#include <fstream>
#include <iostream>
#include <string>
#include <vector>
#include "./LinkedList.h"
```

Include dependency graph for FileHandling.h: This graph shows which files directly or indirectly include this file:

Classes

- class fileHandling

  Class for handling file operations such as reading from and writing to files.

5.18 FileHandling.h

Go to the documentation of this file.

```
00001 #ifndef FILEHANDLING_H
00002 #define FILEHANDLING_H
00003
00004 #include <fstream>
00005 #include <iostream>
00006 #include <string>
00007 #include <vector>
00008
00009 #include "./LinkedList.h"  // Include the header file for the LinkedList class
00010
00015 class fileHandling {
```

```
00016    private:
00017     std::string filename;
00018     template <typename T>
00019     void writeToFileHelper(std::ofstream& file, const T& last) {
00020         file « last « std::endl;
00021     }
00022
00031     template <typename T, typename... Args>
00032     void writeToFileHelper(std::ofstream& file, const T& first, const Args&... args) {
00033         file « first;
00034         if constexpr (sizeof...(args) > 0) {
00035             file « ", ";
00036             writeToFileHelper(file, args...);
00037         } else {
00038             file « std::endl;
00039         }
00040     }
00041
00042    public:
00048     fileHandling(std::string filename);
00049     int size;
00054     ~fileHandling();
00055
00062     template <typename T>
00063     void writeToFile(const T& last) {
00064         std::ofstream file;
00065         file.open(filename, std::ios::app);
00066         file « last « std::endl;  // Writes the last (or only) item followed by a newline
00067         file.close();
00068     }
00069
00076     template <typename... Args>
00077     void writeToFile(const Args&... args) {
00078         std::ofstream file(filename, std::ios::app);  // Open file once
00079         if (!file.is_open()) {
00080             std::cerr « "Error opening file: " « filename « std::endl;
00081
00082             return;
00083         }
00084         writeToFileHelper(file, args...);  // Start the recursive writing with the file stream
00085         file.close();                      // Close file after all writing is done
00086     }
00087
00093     LinkedList<std::string> readFromFile() {
00094         LinkedList<std::string> data;
00095         std::ifstream file;
00096         file.open(filename);
00097         if (!file.is_open()) {
00098             std::cerr « "Error opening file" « filename « std::endl;
00099             return data;
00100         }
00101         std::string line;
00102         while (std::getline(file, line)) {
00103             data.add(line);  // Assuming LinkedList has a push_back method similar to std::vector
00104         }
00105         file.close();
00106         return data;
00107     }
00108 };
00109
00110 #endif
```

5.19 includes/handlePrint.h File Reference

`#include <iostream>`
Include dependency graph for handlePrint.h:

Functions

- void print ()

  Prints a newline character to terminate output.

- template<typename T , typename... Args>
  void print (const T &first, const Args &... args)

  Prints multiple arguments separated by commas.

5.19.1 Function Documentation

5.19.1.1 print() [1/2]

```
void print ()
```

Prints a newline character to terminate output.

Definition at line 8 of file handlePrint.h.

Here is the caller graph for this function:

5.19.1.2 print() [2/2]

```
template<typename T , typename...  Args>
void print (
             const T & first,
             const Args &...  args)
```

Prints multiple arguments separated by commas.

Parameters

first	The first argument to print
args	The remaining arguments to print (optional)

Returns

template <typename T, typename... Args>

Definition at line 21 of file handlePrint.h.

Here is the call graph for this function:

5.20 handlePrint.h

Go to the documentation of this file.

```
00001 #include <iostream>
00002
00007 // Base case for the recursion
00008 void print() {
00009     std::cout « std::endl;  // End the line after all elements are printed
00010 }
00011
00019 // Variadic template function to handle at least one parameter
00020 template <typename T, typename... Args>
00021 void print(const T& first, const Args&... args) {
00022     std::cout « first;
00023     if constexpr (sizeof...(args) > 0) {
00024         std::cout « ", ";
00025         print(args...);  // Recursive call with the rest of the parameters
00026     } else {
00027         std::cout « std::endl;  // End the line after the last element
00028     }
00029 }
```

5.21 includes/LinkedList.h File Reference

`#include <iostream>`
Include dependency graph for LinkedList.h: This graph shows which files directly or indirectly include this file:

Classes

- class LinkedList< T >
- struct LinkedList< T >::Node

5.22 LinkedList.h

Go to the documentation of this file.

```
00001 #ifndef LINKEDLIST_H
00002 #define LINKEDLIST_H
00003 #include <iostream>
00004 template <typename T>
00005 class LinkedList {
00006    protected:
00007     struct Node {
00008         T data;
00009         Node* next;
00010     };
00011     Node* head;
00012     Node* tail;
00013     int size;
00014
00015    public:
00020     LinkedList() {
00021         head = nullptr;
00022         tail = nullptr;
00023         size = 0;
00024     }
00025     Node* getHead() {
00026         return head;
00027     }
00028
00033     ~LinkedList() {
00034         std::cout << "Linked list Deleted" << std::endl;
00035         // Node* current = head;
00036         // while (current != nullptr) {
00037         //     Node* next = current->next;  // Save next node
00038         //     delete current;              // Delete the current node
00039         //     current = next;              // Move to the next node
00040         // }
00041         // head = nullptr;
00042     }
00043
00049     void add(T dataStruct) {
00050         Node* newNode = new Node;
00051         newNode->data = dataStruct;
00052         newNode->next = nullptr;
00053         if (head == nullptr) {
00054             head = newNode;
00055             tail = newNode;
00056         } else {
00057             tail->next = newNode;
00058             tail = newNode;
00059         }
00060         size++;
00061     }
00062
00068     void remove(T data) {
00069         Node* current = head;
00070         Node* previous = nullptr;
00071         while (current != nullptr) {
00072             if (current->data.id == data.id) {
00073                 if (previous == nullptr) {
00074                     head = current->next;
00075                 } else {
00076                     previous->next = current->next;
00077                 }
00078                 delete current;
00079                 size--;
```

```
00080                 return;
00081             }
00082             previous = current;
00083             current = current->next;
00084         }
00085     }
00086
00091     void print() {
00092         Node* current = head;
00093         while (current != nullptr) {
00094             std::cout << current->data << std::endl;
00095             current = current->next;
00096         }
00097     }
00098
00104     int getSize() {
00105         return size;
00106     }
00107
00114     void clean() {
00115         Node* current = head;
00116         while (current != nullptr) {
00117             Node* next = current->next;  // Save next node
00118             delete current;              // Delete the current node
00119             current = next;              // Move to the next node
00120         }
00121         head = nullptr;
00122     }
00123 };
00124
00125 #endif
```

5.23 includes/utils.h File Reference

```
#include <array>
#include <iostream>
#include <sstream>
#include <string>
#include "./LinkedList.h"
```

Include dependency graph for utils.h: This graph shows which files directly or indirectly include this file:

Functions

- template<size_t N>
  void splitData (const std::string &input, char delimiter, std::array< std::string, N > &output)

  Splits a string into parts based on a delimiter.
- double getDouble (std::string prompt="")

  Prompts the user to enter a double value.
- int displayMenu ()

  Displays a menu for an office rental system.

5.23.1 Function Documentation

5.23.1.1 displayMenu()

```
int displayMenu ()  [inline]
```

Displays a menu for an office rental system.

Returns

int The user's selected option from the menu

Definition at line 57 of file utils.h.

Here is the caller graph for this function:

5.23.1.2 getDouble()

```
double getDouble (
            std::string prompt = "")  [inline]
```

Prompts the user to enter a double value.

Parameters

prompt	Optional prompt message to display to the user

Returns

double The double value entered by the user

Definition at line 35 of file utils.h.

Here is the caller graph for this function:

5.23.1.3 splitData()

```
template<size_t N>
void splitData (
            const std::string & input,
            char delimiter,
            std::array< std::string, N > & output)
```

Splits a string into parts based on a delimiter.

Parameters

data	The string to be split
delimiter	The character to split the string by

Returns

std::vector<std::string> Vector of strings containing the split parts

Definition at line 19 of file utils.h.

Here is the caller graph for this function:

5.24 utils.h

Go to the documentation of this file.

```
00001 #ifndef UTILS_H
00002 #define UTILS_H
00003
00004 #include <array>
00005 #include <iostream>
00006 #include <sstream>
00007 #include <string>
00008
00009 #include "./LinkedList.h"
00017 // Definition of splitData function
00018 template <size_t N>
00019 void splitData(const std::string& input, char delimiter, std::array<std::string, N>& output) {
00020     std::istringstream stream(input);
00021     std::string token;
00022     size_t index = 0;
00023
00024     while (std::getline(stream, token, delimiter) && index < N) {
00025         output[index++] = token;
00026     }
00027 }
00034 // gets a number from the user
00035 inline double getDouble(std::string prompt = "") {
00036     std::string num;
00037     char* p;
00038     do {
00039         std::cout << prompt;
00040         std::cin >> num;
00041         double convertedNum = strtod(num.c_str(), &p);
00042         if (*p) {
00043             std::cout << "Invalid input" << std::endl;
00044         } else {
00045             std::cin.ignore();
00046             return convertedNum;
00047         }
00048     } while (true);
00049     return 0;
00050 }
00051
00057 inline int displayMenu() {
00058     std::cout << "Office Space Rental System\n";
00059     std::cout << "[1] Add New Office\n";
00060     std::cout << "[2] Rent an Office\n";
00061     std::cout << "[3] Return an Office\n";
00062     std::cout << "[4] Show Office Details\n";
00063     std::cout << "[5] Display All Offices\n";
00064     std::cout << "[6] Add New Client\n";
00065     std::cout << "[7] Show Client Details\n";
00066     std::cout << "[8] Exit\n";
00067     std::cout << "Choose an option: ";
00068     return 0;
00069 }
00070
00077 // inline std::string generateRandomNumber() {
00078 //     srand(time(0));  // Seed the random number generator
00079 //     std::string random_number_str;
00080 //     int digits = 10;  // Number of digits
00081 //     for (int i = 0; i < digits; ++i) {
00082 //         int random_number = rand() % 10;                    // Generate a random number between 0
      and 9
00083 //         random_number_str += std::to_string(random_number);  // Append the digit to the string
00084 //     }
00085 //     return random_number_str;
00086 // }
00087
00088 #endif  // UTILS_H
```

5.25 main.cpp File Reference

para san to

```
#include <iostream>
#include <string>
#include "./ADT/clientRent.h"
```

```
#include "./ADT/office.h"
#include "./ADT/structData.h"
#include "./includes/fileHandling.h"
#include "./includes/utils.h"
```

Include dependency graph for main.cpp:

Functions

- int main ()

5.25.1 Detailed Description

para san to

Author

group name here

Version

0.5

Date

2024-07-16 when weas this created

Copyright

Copyright (c) 2024

Definition in file main.cpp.

5.25.2 Function Documentation

5.25.2.1 main()

```
int main ()
```

Definition at line 23 of file main.cpp.

Here is the call graph for this function:

5.26 main.cpp

Go to the documentation of this file.

```
00001
00013 #include <iostream>
00014 #include <string>
00015
00016 #include "./ADT/clientRent.h"
00017 #include "./ADT/office.h"
00018 #include "./ADT/structData.h"
00019 #include "./includes/fileHandling.h"
00020 #include "./includes/utils.h"
00021 using namespace std;
00022
00023 int main() {
00024     clientRent clientList = clientRent(-1);
00025     int clientId;
00026     char hasAccount;
00027     cout « "Welcome to Office Space Rental System\n";
00028     cout « "Do you have an account? (y/n): ";
00029     cin.get(hasAccount);
00030     hasAccount = static_cast<char>(toupper(hasAccount));
00031     switch (hasAccount) {
00032         case 'Y': {
00033             clientId = (int)getDouble("Enter your Client ID: ");
00034             cin.ignore();
00035             clientList = clientRent(clientId);
00036             break;
00037         }
00038         case 'N': {
00039             clientData currentClient;
00040             currentClient.id = (int)getDouble("Enter new Client ID: ");
00041             clientId = currentClient.id;
00042             currentClient.isAdmin = false;
00043             cout « currentClient.id « endl;
00044             cout « "Enter new Client Name: ";
00045             getline(cin, currentClient.clientName);
00046             cout « "Enter new Client Address: ";
00047             getline(cin, currentClient.clientAddress);
00048             clientList.addClient(currentClient, true);
00049             cout « "New client added and logged in successfully!\n";
00050             clientId = currentClient.id;
00051             break;
00052         }
00053     }
00054     office officeList(clientId);
00055     bool isRunning = true;
00056     int choice;
00057     while (isRunning) {
00058         displayMenu();
00059         choice = (int)getDouble("Enter your choice: ");
00060         // convert to if else statements
00061         switch (choice) {
00062             //* working and tested.
00063             case 1: {
00064                 officeInformation newOffice;
00065                 cout « "Enter Office ID: ";
00066                 cin » newOffice.id;
00067                 cin.ignore();
00068                 cout « "Enter Office Name: ";
00069                 getline(cin, newOffice.officeName);
00070                 cout « "Enter Office Address: ";
00071                 getline(cin, newOffice.officeAddress);
00072                 cout « "Enter Office Price: ";
00073                 cin » newOffice.officePrice;
00074                 cout « "Enter Office Size: ";
00075                 cin » newOffice.officeSize;
00076                 newOffice.isRented = false;
00077                 officeList.addOffice(newOffice);
00078                 cout « "New Office added successfully!\n";
00079                 break;
00080             }
00081             //* not yet tested
00082             case 2: {
00083                 int officeId;
00084                 cout « "Enter Office ID to rent: ";
00085                 cin » officeId;
00086                 officeInformation office = officeList.getOffice(officeId);
00087                 if (office.id != 0 && !office.isRented) {
00088                     office.isRented = true;
00089                     // officeList.rentOffice(office, clientId);
00090                     cout « "Office rented sucessfully!\n";
00091                 } else {
00092                     cout « "Office is not available for rent.\n";
00093                 }
```

```
00094                 break;
00095             }
00096             //* working and tested
00097             case 3: {
00098                 int officeId;
00099                 cout « "Enter office ID to return: ";
00100                 cin » officeId;
00101 
00102                 // officeInformation office = officeList.getOffice(officeId);
00103                 // if (office.id != 0 && office.isRented) {
00104                 //     office.isRented = false;
00105                 //     // officeList.endRental(office);
00106                 //     cout « "Office returned successfully!\n";
00107                 // } else {
00108                 //     cout « "Office not currently rented.\n";
00109                 // }
00110                 break;
00111             }
00112             //* not yet tested
00113             case 4: {
00114                 int officeId;
00115                 cout « "Enter Office ID to show details: ";
00116                 cin » officeId;
00117 
00118                 // officeInformation officeData = officeList.getOffice(officeId);
00119                 // if (officeData.id != 0) {
00120                 //     cout « "Office ID: " « officeData.id « endl;
00121                 //     cout « "Office Name: " « officeData.officeName « endl;
00122                 //     cout « "Office Address: " « officeData.officeAddress « endl;
00123                 //     cout « "Office Price: " « officeData.officePrice « endl;
00124                 //     cout « "Office Size: " « officeData.officeSize « endl;
00125                 //     cout « "Is Office Rented? " « (officeData.isRented ? "Yes" : "No") « endl;
00126                 // } else {
00127                 //     cout « "Office is not found.\n";
00128                 // }
00129                 break;
00130             }
00131             //* not yet tested. incomplete
00132             case 5: {
00133                 cout « "Displaying all offices: \n";
00134                 officeList.printOffices();
00135                 break;
00136             }
00137             //* working and tested.
00138             case 7: {
00139                 cout « "Enter Client ID to show details: ";
00140                 cin » clientId;
00141                 clientData client = clientList.getClient(clientId);
00142                 if (client.id) {
00143                     cout « "Client ID: " « client.id « endl;
00144                     cout « "Client Name: " « client.clientName « endl;
00145                     cout « "Client Address: " « client.clientAddress « endl;
00146                 } else {
00147                     cout « "Client not found.\n";
00148                 }
00149                 break;
00150             }
00151             case 8: {
00152                 cout « "Exiting program...\n";
00153                 isRunning = false;
00154                 return 0;
00155             }
00156             default:
00157                 cout « "Invalid choice. Please try again.\n";
00158         }
00159     }
00160 
00161     return 0;
00162 }
```

5.27 TEMP/Queue.h File Reference

Classes

- class Queue< T >

5.28 Queue.h

Go to the documentation of this file.

```
00001 #ifndef QUEUE_H
00002 #define QUEUE_H
00003 template <typename T>
00004 class Queue {
00005    private:
00006     struct Node {
00007         T data;
00008         Node* next;
00009     };
00010     Node* head;
00011     Node* tail;
00012     int size;
00013
00014    public:
00015     Queue() {
00016         head = nullptr;
00017         tail = nullptr;
00018         size = 0;
00019     }
00020     ~Queue() {
00021         Node* current = head;
00022         Node* next;
00023         while (current != nullptr) {
00024             next = current->next;
00025             delete current;
00026             current = next;
00027         }
00028     }
00029     void enqueue(T dataStruct) {
00030         Node* newNode = new Node;
00031         newNode->data = dataStruct;
00032         newNode->next = nullptr;
00033         if (head == nullptr) {
00034             head = newNode;
00035             tail = newNode;
00036         } else {
00037             tail->next = newNode;
00038             tail = newNode;
00039         }
00040         size++;
00041     }
00042     void dequeue() {
00043         if (head == nullptr) {
00044             return;
00045         }
00046         Node* temp = head;
00047         head = head->next;
00048         delete temp;
00049         size--;
00050     }
00051     void print() {
00052         Node* current = head;
00053         while (current != nullptr) {
00054             std::cout « current->data « " ";
00055             current = current->next;
00056         }
00057         std::cout « std::endl;
00058     }
00059     int getSize() {
00060         return size;
00061     }
00062     bool isEmpty() {
00063         return size == 0;
00064     }
00065     T front() {
00066         if (head == nullptr) {
00067             return -1;
00068         }
00069         return head->data;
00070     }
00071     T back() {
00072         if (tail == nullptr) {
00073             return -1;
00074         }
00075         return tail->data;
00076     }
00077 };
00078 #endif
```

5.29 TEMP/Stack.h File Reference

Classes

- class Stack< T >

5.30 Stack.h

Go to the documentation of this file.

```
00001 #ifndef STACK_H
00002 #define STACK_H
00003 template <typename T>
00004 class Stack {
00005    private:
00006     struct Node {
00007         T data;
00008         Node* next;
00009     };
00010     Node* head;
00011     int size;
00012
00013    public:
00014     Stack() {
00015         head = nullptr;
00016         size = 0;
00017     }
00018     ~Stack() {
00019         Node* current = head;
00020         Node* next;
00021         while (current != nullptr) {
00022             next = current->next;
00023             delete current;
00024             current = next;
00025         }
00026     }
00027     void push(T dataStruct) {
00028         Node* newNode = new Node;
00029         newNode->data = dataStruct;
00030         newNode->next = head;
00031         head = newNode;
00032         size++;
00033     }
00034     void pop() {
00035         if (head == nullptr) {
00036             return;
00037         }
00038         Node* temp = head;
00039         head = head->next;
00040         delete temp;
00041         size--;
00042     }
00043     void print() {
00044         Node* current = head;
00045         while (current != nullptr) {
00046             std::cout « current->data « " ";
00047             current = current->next;
00048         }
00049         std::cout « std::endl;
00050     }
00051     int getSize() {
00052         return size;
00053     }
00054 };
00055 #endif
```

Index