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

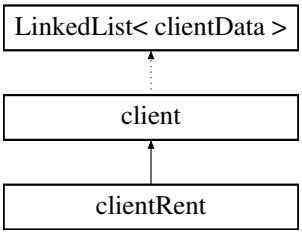
Class Documentation

4.1 client Class Reference

represents a client management class

`#include <client.h>`

Inheritance diagram for client:



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 data)
  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 **12** of file **client.cpp**.

4.1.2.2 ~client()

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

Destroy the client object.

Definition at line **94** 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 **32** of file **client.cpp**.

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 **83** of file **client.cpp**.

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 **84** of file **client.cpp**.

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 (p. 10) Information about the requested client

Definition at line **40** of file **client.cpp**.

4.1.3.5 printClients()

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

Prints information about all clients in the system.

Definition at line **73** of file **client.cpp**.

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 **53** of file **client.cpp**.

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>
```

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:

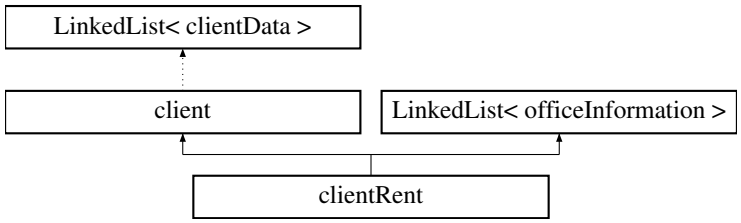
- ADT/ **structData.h**

4.3 clientRent Class Reference

Represents a client rental management class.

`#include <officeRental.h>`

Inheritance diagram for clientRent:



Public Member Functions

- **clientRent** (int clientId)
  *Constructor for the **clientRent*** (p. *12*) *class.*
- bool **rentOffice** (int officeId)
  Rents an office for the client.
- void **ShowAvailableOffices** ()
  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 data)
  Removes a client from the management system.
- void **printClients** ()
  Prints information about all clients in the system.

Public Member Functions inherited from LinkedList< officeInformation >

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

Additional Inherited Members

Protected Attributes inherited from client

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

Protected Attributes inherited from LinkedList< officeInformation >

- 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 **13** of file **officeRental.h**.

4.3.2 Constructor & Destructor Documentation

4.3.2.1 clientRent()

```
clientRent::clientRent (
            int clientId)
```

Constructor for the **clientRent** (p. 12) class.

Parameters

client↩ Id	ID of the client using the rental management

Definition at line **12** of file **officerental.cpp**.

4.3.2.2 ∼clientRent()

```
clientRent::∼clientRent ()
```

Destroy the client Rent object.

Definition at line **58** of file **officerental.cpp**.

4.3.3 Member Function Documentation

4.3.3.1 rentOffice()

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

Rents an office for the client.

Parameters

office↩ Id	ID of the office to rent

Returns

true If the office was successfully rented

false Otherwise

Definition at line **28** of file **officerental.cpp**.

4.3.3.2 ShowAvailableOffices()

```
void clientRent::ShowAvailableOffices ()
```

Shows available offices that can be rented.

Definition at line **44** of file **officerental.cpp**.

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

- ADT/ **officeRental.h**
- ADT/ **officerental.cpp**

4.4 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)
- std::vector< std::string > **readFromFile** ()

4.4.1 Detailed Description

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

Definition at line **13** of file **FileHandling.h**.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 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**.

4.4.2.2 ~fileHandling()

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

Destructor to clean up resources associated with the file handler.

Definition at line **12** of file **FileHandling.cpp**.

4.4.3 Member Function Documentation

4.4.3.1 readFromFile()

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

Definition at line **71** of file **FileHandling.h**.

4.4.3.2 writeToFile() [1/2]

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

Definition at line **61** of file **FileHandling.h**.

4.4.3.3 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 **53** of file **FileHandling.h**.

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

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

4.5 LinkedList< T > Class Template Reference

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

Classes

- struct **Node**

Public Member Functions

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

Protected Attributes

- **Node** * **head**
- **Node** * **tail**
- int **size**

4.5.1 Detailed Description

template<typename T>
class LinkedList< T >

Definition at line **5** of file **LinkedList.h**.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 LinkedList()

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

Constructor to initialize the linked list.

Definition at line **20** of file **LinkedList.h**.

4.5.2.2 ~LinkedList()

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

Destructor to clean up memory allocated for the linked list.

Definition at line **30** of file **LinkedList.h**.

4.5.3 Member Function Documentation

4.5.3.1 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 **48** of file **LinkedList.h**.

4.5.3.2 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 **103** of file **LinkedList.h**.

4.5.3.3 print()

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

Prints all elements in the linked list.

Definition at line **90** of file **LinkedList.h**.

4.5.3.4 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 **67** of file **LinkedList.h**.

4.5.4 Member Data Documentation

4.5.4.1 head

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

Definition at line **11** of file **LinkedList.h**.

4.5.4.2 size

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

Definition at line **13** of file **LinkedList.h**.

4.5.4.3 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**

4.6 LinkedList< T >::Node Struct Reference

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

Public Attributes

- T **data**
- **Node** * **next**

4.6.1 Detailed Description

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

Definition at line **7** of file **LinkedList.h**.

4.6.2 Member Data Documentation

4.6.2.1 data

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

Definition at line **8** of file **LinkedList.h**.

4.6.2.2 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:

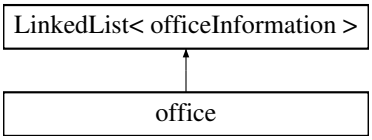
- includes/ **LinkedList.h**

4.7 office Class Reference

Represents an office management class.

`#include <office.h>`

Inheritance 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
- **LinkedList**< **officeInformation** > **getRentedOffices** ()
  retrieves a list of offices currently rented out
- ~**office** ()
  Destroy the office object.

Public Member Functions inherited from LinkedList< officeInformation >

- **LinkedList** ()

  Constructor to initialize the linked list.
- ~**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.

Additional Inherited Members

Protected Attributes inherited from LinkedList< officeInformation >

- Node * **head**
- Node * **tail**
- int **size**

4.7.1 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**.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 office()

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

Constructor for the office class.

Parameters

clientID	ID of the client using the office management system.

Definition at line **13** of file **office.cpp**.

4.7.2.2 ~office()

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

Destroy the office object.

Definition at line **75** of file **office.cpp**.

4.7.3 Member Function Documentation

4.7.3.1 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 **28** of file **office.cpp**.

4.7.3.2 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 **32** of file **office.cpp**.

4.7.3.3 getOffice()

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

retrieves detailed info about a specific office

Parameters

officeId	ID of the office to retrieve information for

Returns

officeInformation (p. 23) Information about the requested office

Definition at line **43** of file **office.cpp**.

4.7.3.4 getRentedOffices()

```
LinkedList< officeInformation > office::getRentedOffices ()
```

retrieves a list of offices currently rented out

Returns

LinkedList<**officeInformation**> (p. 16) List of rented offices

Definition at line **53** of file **office.cpp**.

4.7.3.5 printOffices()

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

prints info about all offices in the system

4.7.3.6 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 **64** of file **office.cpp**.

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

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

4.8 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**

4.8.1 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**.

4.8.2 Member Data Documentation

4.8.2.1 id

```
int officeInformation::id
```

Definition at line **14** of file **structData.h**.

4.8.2.2 isRented

```
bool officeInformation::isRented
```

Definition at line **19** of file **structData.h**.

4.8.2.3 officeAddress

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

Definition at line **16** of file **structData.h**.

4.8.2.4 officeName

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

Definition at line **15** of file **structData.h**.

4.8.2.5 officePrice

```
int officeInformation::officePrice
```

Definition at line **17** of file **structData.h**.

4.8.2.6 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**

4.9 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** ()

4.9.1 Detailed Description

template<typename T>
class Queue< T >

Definition at line **4** of file **Queue.h**.

4.9.2 Constructor & Destructor Documentation

4.9.2.1 Queue()

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

Definition at line **15** of file **Queue.h**.

4.9.2.2 ~Queue()

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

Definition at line **20** of file **Queue.h**.

4.9.3 Member Function Documentation

4.9.3.1 back()

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

Definition at line **71** of file **Queue.h**.

4.9.3.2 dequeue()

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

Definition at line **42** of file **Queue.h**.

4.9.3.3 enqueue()

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

Definition at line **29** of file **Queue.h**.

4.9.3.4 front()

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

Definition at line **65** of file **Queue.h**.

4.9.3.5 getSize()

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

Definition at line **59** of file **Queue.h**.

4.9.3.6 isEmpty()

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

Definition at line **62** of file **Queue.h**.

4.9.3.7 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**

4.10 Stack< T > Class Template Reference

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

Public Member Functions

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

4.10.1 Detailed Description

template<typename T>
class Stack< T >

Definition at line **4** of file **Stack.h**.

4.10.2 Constructor & Destructor Documentation

4.10.2.1 Stack()

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

Definition at line **14** of file **Stack.h**.

4.10.2.2 ~Stack()

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

Definition at line **18** of file **Stack.h**.

4.10.3 Member Function Documentation

4.10.3.1 getSize()

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

Definition at line **51** of file **Stack.h**.

4.10.3.2 pop()

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

Definition at line **34** of file **Stack.h**.

4.10.3.3 print()

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

Definition at line **43** of file **Stack.h**.

4.10.3.4 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**

4.11 User Struct Reference

Struct defining basic user information.

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

Public Attributes

- std::string **username**
- std::string **password**
- int **role**

4.11.1 Detailed Description

Struct defining basic user information.

Contains basic information about a user, such as username, password, and role

Definition at line **42** of file **structData.h**.

4.11.2 Member Data Documentation

4.11.2.1 password

```
std::string User::password
```

Definition at line **44** of file **structData.h**.

4.11.2.2 role

```
int User::role
```

Definition at line **45** of file **structData.h**.

4.11.2.3 username

```
std::string User::username
```

Definition at line **43** of file **structData.h**.

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

- ADT/ **structData.h**

Chapter 5

File Documentation

5.1 ADT/client.cpp File Reference

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

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 <iostream>
00006 #include <vector>
00007
00008 #include "../includes/FileHandling.h"
00009 #include "../includes/LinkedList.h"
00010 #include "../includes/utils.h"
00011 #include "./office.h"
```

```
00012 client::client(int clientID) : LinkedList() {
00013     if (clientID == -1) return;
00014     clientId = clientID;
00015     std::vector<std::string> data = file.readFromFile();
00016     for (std::string line : data) {
00017         std::vector<std::string> clientData = splitData(line, ',');
00018         if (clientID == std::stoi(clientData[0])) {
00019             loggedClient.id = std::stoi(clientData[0]);
00020             loggedClient.clientName = clientData[1];
00021             loggedClient.clientAddress = clientData[2];
00022             loggedClient.isAdmin = std::stoi(clientData[3]) == 1;
00023             loggedClient.rentedSpaces = rented = office(clientID).getRentedOffices();
00024         }
00025         clientList.add({std::stoi(clientData[0]), clientData[1], clientData[2],
      std::stoi(clientData[3]) == 1});
00026     };
00027     if (loggedClient.clientName == "") {
00028         std::cout « "Client not found" « std::endl;
00029     }
00030 };
00031
00032 void client::addClient(clientData data, bool current) {
00033     std::cout « data.isAdmin « std::endl;
00034     add(data);
00035     file.writeToFile(data.id, data.clientName, data.clientAddress, data.isAdmin);
00036     if (current) {
00037         clientId = data.id;
00038     };
00039 }
00040 clientData client::getClient(int clientId) {
00041     Node* current = head;
00042     while (current != nullptr) {
00043         if (current->data.id == clientId) return current->data;
00044         current = current->next;
00045     }
00046     return {};
00047 }
00053 void client::removeClient(int clientId) {
00054     Node* current = head;
00055     while (current != nullptr) {
00056         if (current->data.id == clientId) {
00057             clientList.remove(current->data);
00058             break;
00059         }
00060         current = current->next;
00061     }
00062 }
00063 // void LinkedList<officeInformation>::print() {
00064 //     Node* current = head;
00065 //     while (current != nullptr) {
00066 //         std::cout « "Office ID: " « current->data.officeID « std::endl;
00067 //         std::cout « "Office Size: " « current->data.officeSize « std::endl;
00068 //         std::cout « "Office Price: " « current->data.officePrice « std::endl;
00069 //         std::cout « "Is Office Rented? " « current->data.isRented « std::endl;
00070 //         current = current->next;
00071 //     }
00072 // }
00073 void client::printClients() {
00074     Node* current = head;
00075     while (current != nullptr) {
00076         std::cout « "Client ID: " « current->data.id « std::endl;
00077         std::cout « "Client Name: " « current->data.clientName « std::endl;
00078         std::cout « "Client Phone: " « current->data.clientAddress « std::endl;
00079         std::cout « "Client Rented Space: " « rented.getSize() « std::endl;
00080         current = current->next;
00081     }
00082 }
00083 void client::addRentedSpace(officeInformation data) { rented.add(data); };
00084 void client::changeClient(int clientId) {
00085     Node* current = head;
00086     while (current != nullptr) {
00087         if (current->data.id == clientId) {
00088             current->data = getClient(clientId);
00089             break;
00090         }
00091         current = current->next;
00092     }
00093 }
00094 client::~client() {
00095     std::cout « "Client Deleted" « std::endl;
00096     if (head != nullptr) return;
00097     delete head;
00098 }
00099 #endif
```

5.3 ADT/client.h File Reference

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

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     Node *head;
00024     LinkedList<officeInformation> rented = LinkedList<officeInformation>();
00025
00026    public:
00032     client(int clientId);
00033
00038     ~client();
00039
00046     void addClient(clientData data, bool current = false);
00047
00053     void addRentedSpace(officeInformation data);
00054
00061     clientData getClient(int clientId);
00062
00068     void changeClient(int clientId);
00069
00075     void removeClient(int data);
00076
00081     void printClients();
00082 };
00083 #endif
```

5.5 ADT/office.cpp File Reference

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

Macros

- #define **OFFICE_CPP**

5.5.1 Macro Definition Documentation

5.5.1.1 OFFICE_CPP

```
#define OFFICE_CPP
```

Definition at line **3** of file **office.cpp**.

5.6 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 <iostream>
00007 #include <vector>
00008
00009 #include "../includes/LinkedList.h"
00010 #include "../includes/fileHandling.h"
00011 #include "../includes/utils.h"
00012
00013 office::office(int clientID) : LinkedList() {
00014     std::vector<std::string> data = file.readFromFile();
00015     for (std::string line : data) {
00016         std::vector<std::string> officeData = splitData(line, ',');
00017         officeInformation office;
00018         office.id = std::stoi(officeData[0]);
00019         office.officeName = officeData[1];
00020         office.officeAddress = officeData[2];
00021         office.officePrice = std::stoi(officeData[3]);
00022         office.officeSize = officeData[4];
00023         office.isRented = officeData[5] == "1";
00024         if (office.id != clientID) continue;
00025         add(office);
00026     }
00027 };
00028 void office::addOffice(officeInformation data) {
00029     add(data);
00030     file.writeToFile(data.id, data.officeName, data.officeAddress, data.officePrice, data.officeSize,
      data.isRented);
00031 };
00032 void office::endRental(officeInformation data) {
00033     LinkedList<officeInformation>::Node* current = head;
00034     while (current != nullptr) {
00035         if (current->data.id != data.id) {
00036             current = current->next;
00037             continue;
00038         }
00039         current->data.isRented = false;
00040         file.writeToFile(data.id, data.officeName, data.officeAddress, data.officePrice,
      data.officeSize, data.isRented);
00041     }
00042 }
00043 officeInformation office::getOffice(int officeId) {
00044     LinkedList<officeInformation>::Node* current = head;
00045     while (current != nullptr) {
00046         if (current->data.id == officeId) {
00047             return current->data;
00048         }
00049         current = current->next;
00050     }
00051     return officeInformation();
00052 }
00053 LinkedList<officeInformation> office::getRentedOffices() {
00054     LinkedList<officeInformation> rentedOfficesList;
00055     LinkedList<officeInformation>::Node* current = head;
00056     while (current != nullptr) {
```

```
00057         if (current->data.isRented) {
00058             rentedOfficesList.add(current->data);
00059         }
00060         current = current->next;
00061     }
00062     return rentedOfficesList;
00063 };
00064 void office::rentOffice(officeInformation data, int clientID) {
00065     LinkedList<officeInformation>::Node* current = head;
00066     while (current != nullptr) {
00067         if (current->data.id != data.id) {
00068             current = current->next;
00069             continue;
00070         }
00071         current->data.isRented = true;
00072         file.writeToFile(data.id, clientID);
00073     }
00074 }
00075 office::~office() {
00076     std::cout « "Office Deleted" « std::endl;
00077     if (head != nullptr) return;
00078     delete head;
00079 }
00080 #endif
```

5.7 ADT/office.h File Reference

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

Classes

- class **office**

  Represents an office management class.

5.8 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
00024    public:
00030     office(int clientID);
00031
00037     void addOffice(officeInformation data);
00038
00045     void rentOffice(officeInformation data, int clientID);
00046
00052     void endRental(officeInformation data);
00053
00058     void printOffices();
00059
00066     officeInformation getOffice(int officeId);
00067
00073     LinkedList<officeInformation> getRentedOffices();
00074
00079     ~office();
00080 };
00081 #endif
```

5.9 ADT/officerental.cpp File Reference

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

Macros

- #define **OFFICERENTAL_CPP**

5.9.1 Macro Definition Documentation

5.9.1.1 OFFICERENTAL_CPP

```
#define OFFICERENTAL_CPP
```

Definition at line **3** of file **officerental.cpp**.

5.10 officerental.cpp

Go to the documentation of this file.

```
00001 //* shows a list of AVAILABLE (NOT RENTED) offices.
00002 #ifndef OFFICERENTAL_CPP
00003 #define OFFICERENTAL_CPP
00004 #include "./officeRental.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 clientRent::clientRent(int clientId) : client(clientId), LinkedList<officeInformation>() {
00013     this->clientId = clientId;
00014     fileHandling file = fileHandling("offices.csv");
00015     std::vector<std::string> data = file.readFromFile();
00016     for (std::string line : data) {
00017         std::vector<std::string> officeData = splitData(line, ',');
00018         officeInformation office;
00019         office.id = std::stoi(officeData[0]);
00020         office.officeName = officeData[1];
00021         office.officeAddress = officeData[2];
00022         office.officePrice = std::stoi(officeData[3]);
00023         office.officeSize = officeData[4];
00024         office.isRented = officeData[5] == "1";
00025         this->LinkedList<officeInformation>::add(office);
00026     }
00027 };
00028 bool clientRent::rentOffice(int officeId) {
00029     LinkedList<officeInformation>::Node* current = LinkedList<officeInformation>::head;
00030     while (current != nullptr) {
00031         if (current->data.id != officeId) {
00032             current = current->next;
00033             continue;
00034         }
00035         if (current->data.isRented) {
00036             return false;
00037         }
00038         current->data.isRented = true;
00039         addRentedSpace(current->data);
```

```
00040         return true;
00041     }
00042     return false;
00043 }
00044 void clientRent::ShowAvailableOffices() {
00045     LinkedList<officeInformation>::Node* current = LinkedList<officeInformation>::head;
00046     while (current != nullptr) {
00047         if (!current->data.isRented) {
00048             std::string isRented = current->data.isRented ? "Yes" : "No";
00049             std::cout « "Office ID: " « current->data.id « std::endl;
00050             std::cout « "Office Name: " « current->data.officeName « std::endl;
00051             std::cout « "Office Address: " « current->data.officeAddress « std::endl;
00052             std::cout « "Office Size: " « current->data.officeSize « std::endl;
00053             std::cout « "Unit Price: " « current->data.officePrice « std::endl;
00054         }
00055         current = current->next;
00056     }
00057 }
00058 clientRent::~clientRent() {
00059     if (LinkedList<officeInformation>::head != nullptr)
00060         delete LinkedList<officeInformation>::head;
00061     return;
00062 }
00063 #endif
```

5.11 ADT/officeRental.h File Reference

```
#include "../includes/LinkedList.h"
#include "./client.h"
#include "./office.h"
```

Classes

- class **clientRent**

  Represents a client rental management class.

5.12 officeRental.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 "../includes/LinkedList.h"
00005 #include "./client.h"
00006 #include "./office.h"
00007
00013 class clientRent : public client, public LinkedList<officeInformation> {
00014    private:
00015     int clientId;
00016
00017    public:
00023     clientRent(int clientId);
00024
00032     bool rentOffice(int officeId);
00033
00038     void ShowAvailableOffices();
00039
00044     ~clientRent();
00045 };
00046 #endif
```

5.13 ADT/structData.h File Reference

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

Classes

- struct **officeInformation**

  Struct defining information about an office.
- struct **clientData**

  Struct defining information about a client.
- struct **User**

  Struct defining basic user information.

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
00042 struct User {
00043     std::string username;
00044     std::string password;
00045     int role;
00046 };
00047
00048 #endif
```

5.15 includes/FileHandling.cpp File Reference

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

Macros

- #define **FILEHANDLING_CPP**

5.15.1 Macro Definition Documentation

5.15.1.1 FILEHANDLING_CPP

```
#define FILEHANDLING_CPP
```

Definition at line **2** of file **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>
```

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
00013 class fileHandling {
00014    private:
00015     std::string filename;
00016     template <typename T>
00017     void writeToFileHelper(std::ofstream& file, const T& last) {
00018         file « last « std::endl;
00019     }
00020
00021     template <typename T, typename... Args>
00022     void writeToFileHelper(std::ofstream& file, const T& first, const Args&... args) {
00023         file « first;
00024         if constexpr (sizeof...(args) > 0) {
00025             file « ", ";
00026             writeToFileHelper(file, args...);
00027         } else {
00028             file « std::endl;
00029         }
00030     }
00031
00032    public:
00038     fileHandling(std::string filename);
00039
00044     ~fileHandling();
00045
00052     template <typename T>
00053     void writeToFile(const T& last) {
00054         std::ofstream file;
00055         file.open(filename, std::ios::app);
00056         file « last « std::endl;  // Writes the last (or only) item followed by a newline
00057         file.close();
00058     }
00059
00060     template <typename... Args>
00061     void writeToFile(const Args&... args) {
00062         std::ofstream file(filename, std::ios::app);  // Open file once
00063         if (!file.is_open()) {
00064             std::cerr « "Error opening file" « std::endl;
00065             return;
00066         }
00067         writeToFileHelper(file, args...);  // Start the recursive writing with the file stream
00068         file.close();                      // Close file after all writing is done
00069     }
00070
00071     std::vector<std::string> readFromFile() {
00072         std::vector<std::string> data;
00073         std::ifstream file;
00074         file.open(filename);
00075         if (!file.is_open()) {
00076             std::cerr « "Error opening file" « std::endl;
00077             return data;
00078         }
00079         std::string line;
00080         while (std::getline(file, line)) {
00081             data.push_back(line);
00082         }
00083         file.close();
00084         return data;
00085     };
00086 };
00087
00088 #endif
```

5.19 includes/handlePrint.h File Reference

```
#include <iostream>
```

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**.

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**.

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>`

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
00030     ~LinkedList() {
00031         std::cout « "Linked list Deleted" « std::endl;
00032         Node* current = head;
00033         if (current != nullptr) {
00034             delete current;
00035         }
00036         // while (current != nullptr) {
00037         //     next = current->next;
00038         //     delete current;
00039         //     current = next;
00040         // }
00041     }
00042
00048     void add(T dataStruct) {
00049         Node* newNode = new Node;
00050         newNode->data = dataStruct;
00051         newNode->next = nullptr;
00052         if (head == nullptr) {
00053             head = newNode;
00054             tail = newNode;
00055         } else {
00056             tail->next = newNode;
00057             tail = newNode;
00058         }
00059         size++;
00060     }
00061
00067     void remove(T data) {
00068         Node* current = head;
00069         Node* previous = nullptr;
00070         while (current != nullptr) {
00071             if (current->data.id == data.id) {
00072                 if (previous == nullptr) {
00073                     head = current->next;
00074                 } else {
00075                     previous->next = current->next;
00076                 }
00077                 delete current;
00078                 size--;
00079                 return;
```

```
00080             }
00081             previous = current;
00082             current = current->next;
00083         }
00084     }
00085
00090     void print() {
00091         Node* current = head;
00092         while (current != nullptr) {
00093             std::cout << current->data << std::endl;
00094             current = current->next;
00095         }
00096     }
00097
00103     int getSize() {
00104         return size;
00105     }
00106 };
00107
00108 #endif
```

5.23 includes/utils.h File Reference

```
#include <iostream>
#include <sstream>
#include <vector>
```

Functions

- std::vector< std::string > **splitData** (const std::string &data, char delimiter)

  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**.

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**.

5.23.1.3 splitData()

```
std::vector< std::string > splitData (
            const std::string & data,
            char delimiter)  [inline]
```

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 **16** of file **utils.h**.

5.24 utils.h

Go to the documentation of this file.

```
00001 #ifndef UTILS_H
00002 #define UTILS_H
00003
00004 #include <iostream>
00005 #include <sstream>
00006 #include <vector>
00007
00015 // Definition of splitData function
00016 inline std::vector<std::string> splitData(const std::string& data, char delimiter) {
00017     std::vector<std::string> result;
00018     std::stringstream dataStream(data);
00019     std::string piece;
00020
00021     while (std::getline(dataStream, piece, delimiter)) {
00022         result.push_back(piece);
00023     }
00024
00025     return result;
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 #endif  // UTILS_H
```

5.25 main.cpp File Reference

para san to

```
#include <iostream>
#include <string>
#include "./ADT/client.h"
#include "./ADT/office.h"
#include "./ADT/structData.h"
#include "./includes/fileHandling.h"
#include "./includes/utils.h"
```

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 **24** of file **main.cpp**.

5.26 main.cpp

Go to the documentation of this file.

```
00001
00013 #include <iostream>
00014 #include <string>
00015
00016 #include "./ADT/client.h"
00017 #include "./ADT/office.h"
00018 #include "./ADT/structData.h"
00019 #include "./includes/fileHandling.h"
00020 #include "./includes/utils.h"
00021
00022 using namespace std;
00023
00024 int main() {
00025     client clientList = client(-1);
00026     int clientId;
00027     char hasAccount;
00028     cout « "Welcome to Office Space Rental System\n";
00029     cout « "Do you have an account? (y/n): ";
00030     cin.get(hasAccount);
00031     hasAccount = static_cast<char>(toupper(hasAccount));
00032     switch (hasAccount) {
00033         case 'Y': {
00034             clientId = (int)getDouble("Enter your Client ID: ");
00035             clientList = client(clientId);
00036             break;
00037         }
00038         case 'N': {
00039             clientData currentClient;
00040             currentClient.id = clientId = (int)getDouble("Enter new Client ID: ");
00041             currentClient.isAdmin = false;
00042             cout « currentClient.id « endl;
00043             cout « "Enter new Client Name: ";
00044             getline(cin, currentClient.clientName);
00045             cout « "Enter new Client Address: ";
00046             getline(cin, currentClient.clientAddress);
00047             clientList.addClient(currentClient, true);
00048             cout « "New client added and logged in successfully!\n";
00049             clientId = currentClient.id;
00050             break;
00051         }
00052     }
00053     office officeList(clientId);
00054     bool isRunning = true;
00055     int choice;
00056     while (isRunning) {
00057         displayMenu();
00058         choice = (int)getDouble("Enter your choice: ");
00059         switch (choice) {
00060             case 1: {
00061                 officeInformation newOffice;
00062                 cout « "Enter Office ID: ";
00063                 cin » newOffice.id;
00064                 cin.ignore();
00065                 cout « "Enter Office Name: ";
00066                 getline(cin, newOffice.officeName);
00067                 cout « "Enter Office Address: ";
00068                 getline(cin, newOffice.officeAddress);
00069                 cout « "Enter Office Price: ";
00070                 cin » newOffice.officePrice;
00071                 cout « "Enter Office Size: ";
00072                 cin » newOffice.officeSize;
00073                 newOffice.isRented = false;
00074                 officeList.addOffice(newOffice);
00075                 cout « "New Office added successfully!\n";
00076                 break;
00077             }
```

```
00078             case 2: {
00079                 int officeId;
00080                 cout « "Enter Office ID to rent: ";
00081                 cin » officeId;
00082 
00083                 officeInformation office = officeList.getOffice(officeId);
00084                 if (office.id != 0 && !office.isRented) {
00085                     office.isRented = true;
00086                     officeList.rentOffice(office, clientId);
00087                     cout « "Office rented sucessfully!\n";
00088                 } else {
00089                     cout « "Office is not available for rent.\n";
00090                 }
00091                 break;
00092             }
00093             case 3: {
00094                 int officeId;
00095                 cout « "Enter office ID to return: ";
00096                 cin » officeId;
00097 
00098                 officeInformation office = officeList.getOffice(officeId);
00099                 if (office.id != 0 && office.isRented) {
00100                     office.isRented = false;
00101                     officeList.endRental(office);
00102                     cout « "Office returned successfully!\n";
00103                 } else {
00104                     cout « "Office not currently rented.\n";
00105                 }
00106                 break;
00107             }
00108             case 4: {
00109                 int officeId;
00110                 cout « "Enter Office ID to show details: ";
00111                 cin » officeId;
00112 
00113                 officeInformation officeData = officeList.getOffice(officeId);
00114                 if (officeData.id != 0) {
00115                     cout « "Office ID: " « officeData.id « endl;
00116                     cout « "Office Name: " « officeData.officeName « endl;
00117                     cout « "Office Address: " « officeData.officeAddress « endl;
00118                     cout « "Office Price: " « officeData.officePrice « endl;
00119                     cout « "Office Size: " « officeData.officeSize « endl;
00120                     cout « "Is Office Rented? " « (officeData.isRented ? "Yes" : "No") « endl;
00121                 } else {
00122                     cout « "Office is not found.\n";
00123                 }
00124                 break;
00125             }
00126             case 5: {
00127                 cout « "Displaying all offices: \n";
00128                 // officeList.print();
00129                 break;
00130             }
00131             case 6: {
00132                 clientData newClient;
00133                 cout « "Enter Client ID: ";
00134                 cin » newClient.id;
00135                 cin.ignore();
00136                 cout « "Enter Client Name: ";
00137                 getline(cin, newClient.clientName);
00138                 cout « "Enter Client Address: ";
00139                 getline(cin, newClient.clientAddress);
00140                 clientList.addClient(newClient);
00141                 cout « "New client added successfully!\n";
00142                 break;
00143             }
00144             case 7: {
00145                 cout « "Enter Client ID to show details: ";
00146                 cin » clientId;
00147                 clientData client = clientList.getClient(clientId);
00148                 if (client.id != 0) {
00149                     cout « "Client ID: " « client.id « endl;
00150                     cout « "Client Name: " « client.clientName « endl;
00151                     cout « "Client Address: " « client.clientAddress « endl;
00152                 } else {
00153                     cout « "Client not found.\n";
00154                 }
00155                 break;
00156             }
00157             case 8: {
00158                 cout « "Exiting program...\n";
00159                 isRunning = false;
00160                 return 0;
00161             }
00162             default:
00163                 cout « "Invalid choice. Please try again.\n";
00164         }
```

```
00165     }
00166
00167     return 0;
00168 }
```

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