

University System

Generated by Doxygen 1.13.2

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 BinaryTree< T > Class Template Reference	5
3.1.1 Detailed Description	5
3.1.2 Member Function Documentation	6
3.1.2.1 deleteNode()	6
3.1.2.2 displayNode()	6
3.1.2.3 getSize()	6
3.1.2.4 insert()	6
3.1.2.5 isEmpty()	7
3.2 Course Class Reference	7
3.2.1 Detailed Description	8
3.2.2 Constructor & Destructor Documentation	8
3.2.2.1 Course()	8
3.2.3 Member Function Documentation	9
3.2.3.1 addPrerequisite()	9
3.2.3.2 addToWaitlist()	9
3.2.3.3 isEligible()	9
3.2.3.4 operator"!="()	9
3.2.3.5 operator<()	10
3.2.3.6 operator==(())	10
3.2.3.7 operator>()	10
3.2.4 Friends And Related Symbol Documentation	11
3.2.4.1 operator<<	11
3.3 DNode< T > Class Template Reference	11
3.3.1 Detailed Description	11
3.3.2 Constructor & Destructor Documentation	12
3.3.2.1 DNode()	12
3.4 DoublyLinkedList< T > Class Template Reference	12
3.4.1 Detailed Description	13
3.4.2 Member Function Documentation	13
3.4.2.1 append()	13
3.4.2.2 deleteNode()	13
3.4.2.3 getHead()	14
3.4.2.4 getLength()	14
3.4.2.5 insert()	14
3.4.2.6 isEmpty()	14
3.4.2.7 push()	14

3.4.2.8 removeHead()	15
3.4.2.9 removeNode()	15
3.4.2.10 removeTail()	15
3.5 HashTable< K, V > Class Template Reference	16
3.5.1 Detailed Description	16
3.5.2 Constructor & Destructor Documentation	16
3.5.2.1 HashTable()	16
3.5.3 Member Function Documentation	16
3.5.3.1 get()	16
3.5.3.2 insert()	17
3.5.3.3 remove()	17
3.6 Queue< T > Class Template Reference	18
3.6.1 Detailed Description	18
3.6.2 Member Function Documentation	18
3.6.2.1 dequeue()	18
3.6.2.2 enqueue()	18
3.6.2.3 getSize()	19
3.6.2.4 isEmpty()	19
3.6.2.5 peek()	19
3.7 SinglyLinkedList< T > Class Template Reference	20
3.7.1 Detailed Description	20
3.7.2 Member Function Documentation	20
3.7.2.1 append()	20
3.7.2.2 deleteNode()	21
3.7.2.3 getHead()	21
3.7.2.4 getLength()	21
3.7.2.5 insert()	21
3.7.2.6 isEmpty()	22
3.7.2.7 push()	22
3.7.2.8 removeHead()	22
3.7.2.9 removeNode()	22
3.7.2.10 removeTail()	23
3.8 SNode< T > Class Template Reference	23
3.8.1 Detailed Description	23
3.8.2 Constructor & Destructor Documentation	24
3.8.2.1 SNode()	24
3.9 Stack< T > Class Template Reference	24
3.9.1 Detailed Description	24
3.9.2 Member Function Documentation	25
3.9.2.1 getSize()	25
3.9.2.2 isEmpty()	25
3.9.2.3 peek()	25

3.9.2.4 pop()	25
3.9.2.5 push()	25
3.10 Student Class Reference	26
3.10.1 Detailed Description	27
3.10.2 Constructor & Destructor Documentation	27
3.10.2.1 Student()	27
3.10.3 Member Function Documentation	27
3.10.3.1 addCourse()	27
3.10.3.2 alreadyEnrolled()	27
3.10.3.3 operator"!=()	28
3.10.4 Friends And Related Symbol Documentation	28
3.10.4.1 operator<<	28
3.11 table_pair< K, V > Class Template Reference	28
3.11.1 Detailed Description	29
3.12 TNode< T > Class Template Reference	29
3.12.1 Detailed Description	30
3.12.2 Constructor & Destructor Documentation	30
3.12.2.1 TNode()	30
3.13 UniSystem Class Reference	30
3.13.1 Detailed Description	31
3.13.2 Member Function Documentation	31
3.13.2.1 addCourse() [1/2]	31
3.13.2.2 addCourse() [2/2]	31
3.13.2.3 addStudent()	32
3.13.2.4 checkWaitlist()	32
3.13.2.5 courseExists()	33
3.13.2.6 deleteStudent()	33
3.13.2.7 dropCourse()	33
3.13.2.8 searchCourse()	33
3.13.2.9 searchStudent()	34
3.13.2.10 studentExists()	34
4 File Documentation	35
4.1 include/binary_tree.h File Reference	35
4.1.1 Detailed Description	35
4.2 binary_tree.h	35
4.3 include/binary_tree_node.h File Reference	36
4.3.1 Detailed Description	36
4.4 binary_tree_node.h	36
4.5 include/double_node.h File Reference	36
4.5.1 Detailed Description	37
4.6 double_node.h	37

4.7 include/doubly_linked_list.h File Reference	37
4.7.1 Detailed Description	37
4.8 doubly_linked_list.h	38
4.9 include/entities.h File Reference	38
4.9.1 Detailed Description	38
4.10 entities.h	39
4.11 include/hash_table.h File Reference	40
4.11.1 Detailed Description	40
4.12 hash_table.h	40
4.13 include/includes.h File Reference	41
4.13.1 Detailed Description	41
4.14 includes.h	41
4.15 include/queue.h File Reference	41
4.15.1 Detailed Description	41
4.16 queue.h	42
4.17 include/single_node.h File Reference	42
4.17.1 Detailed Description	42
4.18 single_node.h	42
4.19 singly_linked_list.h	43
4.20 include/stack.h File Reference	43
4.20.1 Detailed Description	43
4.21 stack.h	44
4.22 include/structures.h File Reference	44
4.22.1 Detailed Description	44
4.23 structures.h	44
4.24 include/system.h File Reference	45
4.24.1 Detailed Description	45
4.25 system.h	45
4.26 src/main.cpp File Reference	46
4.26.1 Detailed Description	46
4.26.2 Function Documentation	46
4.26.2.1 add()	46
4.26.2.2 enroll()	47
4.26.2.3 freeSeat()	47
4.26.2.4 loop()	47
4.26.2.5 main()	47
4.26.2.6 printHelp()	47
4.26.2.7 remove()	47
4.26.2.8 runCommand()	48
4.26.2.9 search()	48
4.26.2.10 splitInput()	48
4.26.2.11 strip()	49

4.26.2.12 testData()	49
4.26.2.13 view()	49
Index	51

Chapter 1

Class Index

1.1 Class List

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

BinaryTree< T >	Definition of the BinaryTree class	5
Course	The Course class	7
DNode< T >	The nodes for the DoublyLinkedList class	11
DoublyLinkedList< T >	The DoublyLinkedList class	12
HashTable< K, V >	The HashTable class	16
Queue< T >	The Queue class	18
SinglyLinkedList< T >	The SinglyLinkedList class	20
SNode< T >	The nodes for the SinglyLinkedList class	23
Stack< T >	The Stack class	24
Student	The Student class	26
table_pair< K, V >	A container for a key and its value	28
TNode< T >	The nodes for the BinaryTree class	29
UniSystem	The UniSystem class	30

Chapter 2

File Index

2.1 File List

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

include/binary_tree.h	
Defines the BinaryTree class	35
include/binary_tree_node.h	
Defines the Binary Tree Nodes (TNode) class	36
include/double_node.h	
Defines the Double Node (DNode) class	36
include/doubly_linked_list.h	
Defines the DoublyLinkedList class	37
include/entities.h	
Defines the Student and Course classes	38
include/hash_table.h	
Defines the HashTable and table_pair Classes	40
include/includes.h	
Collects the included files used throughout the project	41
include/queue.h	
Defines the Queue class	41
include/single_node.h	
Defines the Single Node (SNode) class	42
include/singly_linked_list.h	
	43
include/stack.h	
Defines the Stack class	43
include/structures.h	
Collects all the implemented data structures in one file	44
include/system.h	
Defines the University System (UniSystem) class	45
src/main.cpp	
Main program code	46

Chapter 3

Class Documentation

3.1 `BinaryTree< T >` Class Template Reference

Definition of the `BinaryTree` class.

```
#include <binary_tree.h>
```

Public Member Functions

- `BinaryTree ()`
Constructs an empty tree.
- `bool isEmpty ()`
A function to check if the tree is empty.
- `int getSize ()`
A function that returns the size of the tree.
- `bool insert (T value)`
A function that inserts a value in the correct node in the tree.
- `bool deleteNode (T value)`
A function that removes a certain value from the tree.
- `void displayTree ()`
A function that displays the tree in-order.
- `void displayNode (TNode< T > *root)`
A function that traverses nodes in an in-order traversal.

3.1.1 Detailed Description

```
template<typename T>  
class BinaryTree< T >
```

Definition of the `BinaryTree` class.

Template Parameters

<code>T</code>	the type of values stored in the <code>BinaryTree</code> object.
----------------	--

3.1.2 Member Function Documentation

3.1.2.1 deleteNode()

```
template<typename T>
bool BinaryTree< T >::deleteNode (
    T value)
```

A function that removes a certain value from the tree.

Parameters

<i>value</i>	The value to delete.
--------------	----------------------

Returns

boolean.

3.1.2.2 displayNode()

```
template<typename T>
void BinaryTree< T >::displayNode (
    TNode< T > * root)
```

A function that traverses nodes in an in-order traversal.

Parameters

<i>*root</i>	A pointer to the starting node.
--------------	---------------------------------

3.1.2.3 getSize()

```
template<typename T>
int BinaryTree< T >::getSize ()
```

A function that returns the size of the tree.

Returns

Tree size.

3.1.2.4 insert()

```
template<typename T>
bool BinaryTree< T >::insert (
    T value)
```

A function that inserts a value in the correct node in the tree.

Parameters

<i>value</i>	The value to store.
--------------	---------------------

Returns

boolean.

3.1.2.5 isEmpty()

```
template<typename T>
bool BinaryTree< T >::isEmpty ()
```

A function to check if the tree is empty.

Returns

True if the tree is empty, false otherwise.

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

- include/[binary_tree.h](#)

3.2 Course Class Reference

The [Course](#) class.

```
#include <entities.h>
```

Public Member Functions

- [Course](#) (int *id*, int *credits*, string *name*, string *instructor*, int *max_seats*, int *seats*)
Constructs a new [Course](#) object with given values.
- [Course](#) ()
Constructs an empty [Course](#) object.
- bool [isEligible](#) ([Student](#) student)
Checks if a student is eligible to enroll in this course.
- bool [addToWaitlist](#) ([Student](#) student)
A function that adds students to this course's waitlist when no seats are available.
- bool [addPrerequisite](#) ([Course](#) course)
A function that adds prerequisites to this course.
- void [displayDetails](#) ()
A function that displays this course's details.
- bool [operator<](#) (const [Course](#) &other)
Less Than operator.
- bool [operator>](#) (const [Course](#) &other)
Greater Than operator.
- bool [operator==](#) (const [Course](#) &other)
Equality operator.
- bool [operator!=](#) (const [Course](#) &other)
Inequality operator.

Public Attributes

- `int id`
The [Course](#)'s ID.
- `int credits`
No. of credits.
- `int max_seats`
Max seats.
- `int seats`
Taken seats.
- `string name`
The [Course](#)'s name.
- `string instructor`
Instructor.
- `Stack< Course > * prerequisites`
A stack holding the [Course](#)'s prerequisites.
- `Queue< Student > * waitlist`
A queue managing the [Course](#)'s wait-list.

Friends

- `ostream & operator<< (std::ostream &os, Course &course)`
Stream insertion operator.

3.2.1 Detailed Description

The [Course](#) class.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 Course()

```
Course::Course (
    int id,
    int credits,
    string name,
    string instructor,
    int max_seats,
    int seats)
```

Constructs a new [Course](#) object with given values.

Parameters

<i>id</i>	the course's ID.
<i>credits</i>	the credits taken from this course.
<i>name</i>	this course's name.
<i>instructor</i>	this course instructor's name.
<i>max_seats</i>	this course's max seats.
<i>seats</i>	this course's taken seats.

3.2.3 Member Function Documentation

3.2.3.1 addPrerequisite()

```
bool Course::addPrerequisite (
    Course course)
```

A function that adds prerequisites to this course.

Parameters

<i>course</i>	The prerequisite course.
---------------	--------------------------

Returns

boolean.

3.2.3.2 addToWaitlist()

```
bool Course::addToWaitlist (
    Student student)
```

A function that adds students to this course's waitlist when no seats are available.

Parameters

<i>student</i>	The student enrolling in this course.
----------------	---------------------------------------

Returns

boolean.

3.2.3.3 isEligible()

```
bool Course::isEligible (
    Student student)
```

Checks if a student is eligible to enroll in this course.

Parameters

<i>student</i>	The student to enroll.
----------------	------------------------

Returns

boolean.

3.2.3.4 operator!=(())

```
bool Course::operator!=(
    const Course & other) [inline]
```

Inequality operator.

Parameters

<i>other</i>	The objects to compare.
--------------	-------------------------

Returns

boolean.

3.2.3.5 operator<()

```
bool Course::operator< (  
    const Course & other) [inline]
```

Less Than operator.

Parameters

<i>other</i>	The object to compare
--------------	-----------------------

Returns

boolean.

3.2.3.6 operator==()

```
bool Course::operator== (  
    const Course & other) [inline]
```

Equality operator.

Parameters

<i>other</i>	The object to compare.
--------------	------------------------

Returns

boolean.

3.2.3.7 operator>()

```
bool Course::operator> (  
    const Course & other) [inline]
```

Greater Than operator.

Parameters

<i>other</i>	The object to compare.
--------------	------------------------

Returns

boolean.

3.2.4 Friends And Related Symbol Documentation

3.2.4.1 operator<<

```
ostream & operator<< (
    std::ostream & os,
    Course & course) [friend]
```

Stream insertion operator.

Parameters

<i>os</i>	The input stream.
<i>course</i>	The object to insert in the stream.

Returns

os

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

- include/[entities.h](#)

3.3 DNode< T > Class Template Reference

The nodes for the [DoublyLinkedList](#) class.

```
#include <double_node.h>
```

Public Member Functions

- [DNode](#) (T *value*)

Public Attributes

- T **value**
The stored value.
- [DNode](#) * **prev**
Pointers to the previous [DNode](#) object.
- [DNode](#) * **next**
Pointers to the next [DNode](#) object.

3.3.1 Detailed Description

```
template<typename T>
class DNode< T >
```

The nodes for the [DoublyLinkedList](#) class.

Template Parameters

T	The type of values stored inside the DNode object.
----------	--

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DNode()

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

Constructs an empty [DNode](#) object.

Parameters

value	The value to store inside the DNode object.
--------------	---

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

- include/[double_node.h](#)

3.4 DoublyLinkedList< T > Class Template Reference

The [DoublyLinkedList](#) class.

```
#include <doubly_linked_list.h>
```

Public Member Functions

- **DoublyLinkedList ()**
Constructs an empty [DoublyLinkedList](#) object.
- bool **isEmpty ()**
A function to check if the list is empty.
- int **getLength ()**
A function that returns the length of the list.
- [DNode< T > *](#) **getHead ()**
A function that returns the head of the list.
- bool **append (T value)**
Adds a value to the end of the list.
- bool **insert (T value, int position)**
Adds a value to an index in the list.
- bool **push (T value)**
Adds a value to the beginning of the list.
- bool **removeHead ()**
A function that removes the head of the list.
- bool **removeTail ()**
a function that removes the tail of the list.
- bool **removeNode (DNode< T > *node)**
A function that removes a specific node.
- bool **deleteNode (int index)**
A wrapper function that removes a node from anywhere in the list.
- void **display ()**
A function that displays the list.

3.4.1 Detailed Description

```
template<typename T>
class DoublyLinkedList< T >
```

The [DoublyLinkedList](#) class.

Template Parameters

<i>T</i>	The type of values stored inside the DoublyLinkedList object.
----------	---

3.4.2 Member Function Documentation

3.4.2.1 `append()`

```
template<typename T>
bool DoublyLinkedList< T >::append (
    T value)
```

Adds a value to the end of the list.

Parameters

<i>value</i>	The value to add.
--------------	-------------------

Returns

boolean.

3.4.2.2 `deleteNode()`

```
template<typename T>
bool DoublyLinkedList< T >::deleteNode (
    int index)
```

A wrapper function that removes a node from anywhere in the list.

Parameters

<i>index</i>	The index of the node to remove.
--------------	----------------------------------

Returns

boolean.

3.4.2.3 getHead()

```
template<typename T>
DNode< T > * DoublyLinkedList< T >::getHead ()
```

A function that returns the head of the list.

Returns

A pointer to the head.

3.4.2.4 getLength()

```
template<typename T>
int DoublyLinkedList< T >::getLength ()
```

A function that returns the length of the list.

Returns

List length.

3.4.2.5 insert()

```
template<typename T>
bool DoublyLinkedList< T >::insert (
    T value,
    int position)
```

Adds a value to an index in the list.

Parameters

<i>value</i>	The value to add.
<i>position</i>	The position to add the value at.

Returns

boolean.

3.4.2.6 isEmpty()

```
template<typename T>
bool DoublyLinkedList< T >::isEmpty ()
```

A function to check if the list is empty.

Returns

True if the list is empty, false otherwise.

3.4.2.7 push()

```
template<typename T>
bool DoublyLinkedList< T >::push (
    T value)
```

Adds a value to the beginning of the list.

Parameters

<i>value</i>	The value to add.
--------------	-------------------

Returns

boolean.

3.4.2.8 removeHead()

```
template<typename T>
bool DoublyLinkedList< T >::removeHead ()
```

A function that removes the head of the list.

Returns

boolean.

3.4.2.9 removeNode()

```
template<typename T>
bool DoublyLinkedList< T >::removeNode (
    DNode< T > * node)
```

A function that removes a specific node.

Parameters

<i>node</i>	The pointer to the node to delete.
-------------	------------------------------------

Returns

boolean.

3.4.2.10 removeTail()

```
template<typename T>
bool DoublyLinkedList< T >::removeTail ()
```

a function that removes the tail of the list.

Returns

boolean.

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

- include/[doubly_linked_list.h](#)

3.5 HashTable< K, V > Class Template Reference

The [HashTable](#) class.

```
#include <hash_table.h>
```

Public Member Functions

- [HashTable](#) (int table_size)
Constructs an empty [HashTable](#) object given a size.
- bool [insert](#) (K key, V value)
A function that inserts a value into the [HashTable](#) object.
- V * [get](#) (K key)
A function that returns a pointer to the value using a key.
- bool [remove](#) (K key)
A function that removes a value from the [HashTable](#) object.

3.5.1 Detailed Description

```
template<typename K, typename V>
class HashTable< K, V >
```

The [HashTable](#) class.

Template Parameters

<i>K</i>	The type of key values.
<i>V</i>	The type of values.

3.5.2 Constructor & Destructor Documentation

3.5.2.1 HashTable()

```
template<typename K, typename V>
HashTable< K, V >::HashTable (
    int table_size)
```

Constructs an empty [HashTable](#) object given a size.

Parameters

<i>table_size</i>	The size intended for the table.
-------------------	----------------------------------

3.5.3 Member Function Documentation

3.5.3.1 get()

```
template<typename K, typename V>
V * HashTable< K, V >::get (
    K key)
```

A function that returns a pointer to the value using a key.

Parameters

<i>key</i>	The key of the value.
------------	-----------------------

Returns

A pointer to the value.

3.5.3.2 insert()

```
template<typename K, typename V>
bool HashTable< K, V >::insert (
    K key,
    V value)
```

A function that inserts a value into the [HashTable](#) object.

Parameters

<i>key</i>	the key of the value.
<i>value</i>	the value to insert.

Returns

boolean.

3.5.3.3 remove()

```
template<typename K, typename V>
bool HashTable< K, V >::remove (
    K key)
```

A function that removes a value from the [HashTable](#) object.

Parameters

<i>key</i>	The key of the value to remove.
------------	---------------------------------

Returns

boolean.

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

- [include/hash_table.h](#)

3.6 Queue< T > Class Template Reference

The [Queue](#) class.

```
#include <queue.h>
```

Public Member Functions

- **Queue ()**
Constructs an empty [Queue](#) object.
- **bool isEmpty ()**
A function to check if the queue is empty.
- **int getSize ()**
A function that returns the length of the queue.
- **bool enqueue (T value)**
A function that adds a value to the end of the queue.
- **bool dequeue ()**
A function that removes the first value in the queue.
- **T * peek ()**
A function that returns a pointer to the first element in the queue.

3.6.1 Detailed Description

```
template<typename T>
class Queue< T >
```

The [Queue](#) class.

Template Parameters

<i>T</i>	The type of values store in the Queue object.
----------	---

3.6.2 Member Function Documentation

3.6.2.1 dequeue()

```
template<typename T>
bool Queue< T >::dequeue ()
```

A function that removes the first value in the queue.

Returns

boolean.

3.6.2.2 enqueue()

```
template<typename T>
bool Queue< T >::enqueue (
    T value)
```

A function that adds a value to the end of the queue.

Parameters

<i>value</i>	The value to add.
--------------	-------------------

Returns

boolean.

3.6.2.3 getSize()

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

A function that returns the length of the queue.

Returns

queue size.

3.6.2.4 isEmpty()

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

A function to check if the queue is empty.

Returns

True if the list is empty, false otherwise.

3.6.2.5 peek()

```
template<typename T>
T * Queue< T >::peek ()
```

A function that returns a pointer to the first element in the queue.

Returns

Pointer to the stack top value.

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

- [include/queue.h](#)

3.7 SinglyLinkedList< T > Class Template Reference

The [SinglyLinkedList](#) class.

```
#include <singly_linked_list.h>
```

Public Member Functions

- **SinglyLinkedList** ()
Constructs an empty [SinglyLinkedList](#) object.
- **SNode**< T > * [getHead](#) ()
A function that returns the head of the list.
- bool [isEmpty](#) ()
A function to check if the list is empty.
- bool [append](#) (T value)
Adds a value to the end of the list.
- bool [insert](#) (T value, int position)
Adds a value to an index in the list.
- bool [push](#) (T value)
Adds a value to the beginning of the list.
- bool [removeNode](#) (**SNode**< T > *node)
A function that removes a specific node.
- bool [removeHead](#) ()
A function that removes the head of the list.
- bool [removeTail](#) ()
a function that removes the tail of the list.
- bool [deleteNode](#) (int position)
A wrapper function that removes a node from anywhere in the list.
- void **display** ()
A function that displays the list.
- int [getLength](#) ()
A function that returns the length of the list.

3.7.1 Detailed Description

```
template<typename T>
class SinglyLinkedList< T >
```

The [SinglyLinkedList](#) class.

Template Parameters

<i>T</i>	The type of values stored inside the SinglyLinkedList object.
----------	---

3.7.2 Member Function Documentation

3.7.2.1 [append\(\)](#)

```
template<typename T>
bool SinglyLinkedList< T >::append (
    T value)
```

Adds a value to the end of the list.

Parameters

<i>value</i>	The value to add.
--------------	-------------------

Returns

boolean.

3.7.2.2 deleteNode()

```
template<typename T>
bool SinglyLinkedList< T >::deleteNode (
    int position)
```

A wrapper function that removes a node from anywhere in the list.

Parameters

<i>position</i>	The index of the node to remove.
-----------------	----------------------------------

Returns

boolean.

3.7.2.3 getHead()

```
template<typename T>
SNode< T > * SinglyLinkedList< T >::getHead ()
```

A function that returns the head of the list.

Returns

A pointer to the head.

3.7.2.4 getLength()

```
template<typename T>
int SinglyLinkedList< T >::getLength ()
```

A function that returns the length of the list.

Returns

List length.

3.7.2.5 insert()

```
template<typename T>
bool SinglyLinkedList< T >::insert (
    T value,
    int position)
```

Adds a value to an index in the list.

Parameters

<i>value</i>	The value to add.
<i>position</i>	The position to add the value at.

Returns

boolean.

3.7.2.6 isEmpty()

```
template<typename T>
bool SinglyLinkedList< T >::isEmpty ()
```

A function to check if the list is empty.

Returns

True if the list is empty, false otherwise.

3.7.2.7 push()

```
template<typename T>
bool SinglyLinkedList< T >::push (
    T value)
```

Adds a value to the beginning of the list.

Parameters

<i>value</i>	The value to add.
--------------	-------------------

Returns

boolean.

3.7.2.8 removeHead()

```
template<typename T>
bool SinglyLinkedList< T >::removeHead ()
```

A function that removes the head of the list.

Returns

boolean.

3.7.2.9 removeNode()

```
template<typename T>
bool SinglyLinkedList< T >::removeNode (
    SNode< T > * node)
```

A function that removes a specific node.

Parameters

<i>node</i>	The pointer to the previous node to the target to delete.
-------------	---

Returns

boolean.

3.7.2.10 removeTail()

```
template<typename T>
bool SinglyLinkedList< T >::removeTail ()
```

a function that removes the tail of the list.

Returns

boolean.

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

- include/singly_linked_list.h

3.8 SNode< T > Class Template Reference

The nodes for the [SinglyLinkedList](#) class.

```
#include <single_node.h>
```

Public Member Functions

- [SNode](#) (T value)
Constructs an empty [SNode](#) object.

Public Attributes

- T value
The value stored in the [SNode](#) object.
- [SNode](#) * next
A pointer to the next [SNode](#) object.

3.8.1 Detailed Description

```
template<typename T>
class SNode< T >
```

The nodes for the [SinglyLinkedList](#) class.

Template Parameters

T	The type of values stored inside the SNode object.
----------	--

3.8.2 Constructor & Destructor Documentation

3.8.2.1 SNode()

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

Constructs an empty [SNode](#) object.

Parameters

value	The value to hold inside the SNode object.
--------------	--

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

- include/[single_node.h](#)

3.9 Stack< T > Class Template Reference

The [Stack](#) class.

```
#include <stack.h>
```

Public Member Functions

- **Stack ()**
Constructs an empty [Stack](#) object.
- bool [isEmpty](#) ()
A function to check if the stack is empty.
- int [getSize](#) ()
A function to get the size of the stack.
- bool [push](#) (T object)
A function that pushes values into the stack.
- bool [pop](#) ()
A function that removes the top value of the stack.
- T [peek](#) ()
A function that returns the top value of the stack.

3.9.1 Detailed Description

```
template<typename T>
class Stack< T >
```

The [Stack](#) class.

Template Parameters

T	The type of value stored in the stack.
----------	--

3.9.2 Member Function Documentation

3.9.2.1 getSize()

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

A function to get the size of the stack.

Returns

The size of the stack.

3.9.2.2 isEmpty()

```
template<typename T>
bool Stack< T >::isEmpty ()
```

A function to check if the stack is empty.

Returns

True if the list is empty, false otherwise.

3.9.2.3 peek()

```
template<typename T>
T Stack< T >::peek ()
```

A function that returns the top value of the stack.

Returns

T The top of the stack.

3.9.2.4 pop()

```
template<typename T>
bool Stack< T >::pop ()
```

A function that removes the top value of the stack.

Returns

boolean.

3.9.2.5 push()

```
template<typename T>
bool Stack< T >::push (
    T object)
```

A function that pushes values into the stack.

Parameters

<i>object</i>	The value to add to the stack.
---------------	--------------------------------

Returns

boolean.

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

- include/[stack.h](#)

3.10 Student Class Reference

The [Student](#) class.

```
#include <entities.h>
```

Public Member Functions

- [Student](#) (int [id](#), string [name](#), string [email](#), string [password](#), string [address](#), int [phone](#))
Constructs a new [Student](#) object with given values.
- **Student** ()
Constructs an empty [Student](#) object.
- bool [alreadyEnrolled](#) ([Course](#) course)
Checks if this student is already enrolled in a course.
- bool [addCourse](#) ([Course](#) *course)
A function that adds a course to this student's enrollment history.
- void **viewCourses** ()
A function that views course enrollment history.
- void **displayDetails** ()
A function that displays this course's details.
- bool [operator!=](#) (const [Student](#) &other)
Inequality Operator.

Public Attributes

- string **name**
[Student](#) name.
- string **email**
[Student](#) email.
- string **password**
[Student](#) password.
- string **address**
[Student](#) address.
- int **id**
[Student](#) ID.
- int **phone**
[Student](#) phone number.
- [DoublyLinkedList](#)< [Course](#) > * **course_history**
The [Student](#)'s enrollment history.

Friends

- ostream & operator<< (std::ostream &os, Student &student)
Stream insertion operator.

3.10.1 Detailed Description

The Student class.

3.10.2 Constructor & Destructor Documentation

3.10.2.1 Student()

```
Student::Student (
    int id,
    string name,
    string email,
    string password,
    string address,
    int phone)
```

Constructs a new Student object with given values.

Parameters

<i>id</i>	the student's ID.
<i>name</i>	this student's name.
<i>email</i>	this student's email.
<i>password</i>	this student's password.
<i>address</i>	this student's address.
<i>phone</i>	this student's phone.

3.10.3 Member Function Documentation

3.10.3.1 addCourse()

```
bool Student::addCourse (
    Course * course)
```

A function that adds a course to this student's enrollment history.

Parameters

<i>course</i>	a pointer to the course to add.
---------------	---------------------------------

3.10.3.2 alreadyEnrolled()

```
bool Student::alreadyEnrolled (
    Course course)
```

Checks if this student is already enrolled in a course.

Parameters

<i>course</i>	the course this student intends to be enrolled in.
---------------	--

3.10.3.3 operator"!=()

```
bool Student::operator!= (
    const Student & other) [inline]
```

Inequality Operator.

Parameters

<i>other</i>	The objects to compare.
--------------	-------------------------

Returns

boolean.

3.10.4 Friends And Related Symbol Documentation**3.10.4.1 operator<<**

```
ostream & operator<< (
    std::ostream & os,
    Student & student) [friend]
```

Stream insertion operator.

Parameters

<i>os</i>	The input stream.
<i>student</i>	The object to insert in the stream.

Returns

os

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

- include/[entities.h](#)

3.11 table_pair< K, V > Class Template Reference

A container for a key and its value.

```
#include <hash_table.h>
```

Public Attributes

- **K key**
The key to store.
- **V value**
The value to store.

3.11.1 Detailed Description

```
template<typename K, typename V>
class table_pair< K, V >
```

A container for a key and its value.

Template Parameters

<i>K</i>	The type of key values.
<i>V</i>	The type of values.

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

- [include/hash_table.h](#)

3.12 TNode< T > Class Template Reference

The nodes for the [BinaryTree](#) class.

```
#include <binary_tree_node.h>
```

Public Member Functions

- [TNode](#) ([T value](#))

Public Attributes

- **T value**
The value stored inside the [TNode](#) object.
- [TNode](#) * **right**
The pointer to the right [TNode](#) object.
- [TNode](#) * **left**
The pointer to the left [TNode](#) object.
- bool **is_root**
True when the object is the root.
- bool **is_right**
True when the object is the right child.
- bool **is_left**
True when the object is the left child.

3.12.1 Detailed Description

```
template<typename T>
class TNode< T >
```

The nodes for the [BinaryTree](#) class.

3.12.2 Constructor & Destructor Documentation

3.12.2.1 TNode()

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

Constructs an empty [TNode](#) object.

Parameters

<i>value</i>	The value stored inside the TNode object.
--------------	---

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

- include/[binary_tree_node.h](#)

3.13 UniSystem Class Reference

The [UniSystem](#) class.

```
#include <system.h>
```

Public Member Functions

- **UniSystem ()**
Constructs an empty [UniSystem](#) object.
- bool [courseExists](#) (int id)
A function that checks if a course exists.
- bool [studentExists](#) (int id)
A function that checks if a student exists.
- bool [addStudent](#) (int id, string name, string email, string password, string address, int phone)
A function that adds a student to the [UniSystem](#) object.
- bool [deleteStudent](#) (int id)
A function that removes a student from the [UniSystem](#) object.
- void [listStudents](#) ()
A function that lists all students in the [UniSystem](#) object.
- void [listCourses](#) ()
A function that lists all courses in the [UniSystem](#) object.

- bool `addCourse` (int id, string name, int credits, string instructor, int max_seats, int seats)
A function that adds a course to the `UniSystem` with given parameters.
- bool `addCourse` (`Course` course)
A function that adds a course to the `UniSystem`.
- bool `dropCourse` (int id)
A function that removes a course from the `UniSystem` object.
- bool `checkWaitlist` (`Course` &course)
A function that checks the waitlist when a course gets a free seat.
- bool `searchStudent` (int id)
A function that displays a specific student details.
- bool `searchCourse` (int id)
A function that displays a specific course details.

Public Attributes

- `HashTable`< int, `Course` > * `courses_table`
A hash table to store `Course` objects.
- `HashTable`< int, `Student` > * `students_table`
A hash table to store `Student` objects.

3.13.1 Detailed Description

The `UniSystem` class.

3.13.2 Member Function Documentation

3.13.2.1 `addCourse()` [1/2]

```
bool UniSystem::addCourse (
    Course course)
```

A function that adds a course to the `UniSystem`.

Parameters

<i>course</i>	<code>Course</code> object to add.
---------------	------------------------------------

Returns

boolean.

3.13.2.2 `addCourse()` [2/2]

```
bool UniSystem::addCourse (
    int id,
    string name,
    int credits,
    string instructor,
    int max_seats,
    int seats)
```

A function that adds a course to the `UniSystem` with given parameters.

Parameters

<i>id</i>	The course's ID.
<i>credits</i>	The credits taken from this course.
<i>name</i>	This course's name.
<i>instructor</i>	This course instructor's name.
<i>max_seats</i>	This course's max seats.
<i>seats</i>	This course's taken seats.

Returns

boolean.

3.13.2.3 addStudent()

```
bool UniSystem::addStudent (
    int id,
    string name,
    string email,
    string password,
    string address,
    int phone)
```

A function that adds a student to the [UniSystem](#) object.

Parameters

<i>id</i>	The student's ID.
<i>name</i>	This student's name.
<i>email</i>	This student's email.
<i>password</i>	This student's password.
<i>address</i>	This student's address.
<i>phone</i>	This student's phone.

Returns

boolean.

3.13.2.4 checkWaitlist()

```
bool UniSystem::checkWaitlist (
    Course & course)
```

A function that checks the waitlist when a course gets a free seat.

Parameters

<i>course</i>	A reference to the Course Object.
---------------	---

Returns

boolean.

3.13.2.5 courseExists()

```
bool UniSystem::courseExists (
    int id)
```

A function that checks if a course exists.

Parameters

<i>id</i>	The course's ID.
-----------	------------------

Returns

True if the course exists, false otherwise.

3.13.2.6 deleteStudent()

```
bool UniSystem::deleteStudent (
    int id)
```

A function that removes a student from the [UniSystem](#) object.

Parameters

<i>id</i>	the student's ID.
-----------	-------------------

Returns

boolean.

3.13.2.7 dropCourse()

```
bool UniSystem::dropCourse (
    int id)
```

A function that removes a course from the [UniSystem](#) object.

Parameters

<i>id</i>	The course ID.
-----------	----------------

Returns

boolean.

3.13.2.8 searchCourse()

```
bool UniSystem::searchCourse (
    int id)
```

A function that displays a specific course details.

Parameters

<i>id</i>	Course ID.
-----------	------------

Returns

boolean.

3.13.2.9 searchStudent()

```
bool UniSystem::searchStudent (  
    int id)
```

A function that displays a specific student details.

Parameters

<i>id</i>	Student ID.
-----------	-------------

Returns

boolean.

3.13.2.10 studentExists()

```
bool UniSystem::studentExists (  
    int id)
```

A function that checks if a student exists.

Parameters

<i>id</i>	The student's ID.
-----------	-------------------

Returns

True if the student exists, false otherwise.

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

- include/[system.h](#)

Chapter 4

File Documentation

4.1 include/binary_tree.h File Reference

Defines the [BinaryTree](#) class.

```
#include "binary_tree_node.h"
#include "includes.h"
#include "../templates/binary_tree.hpp"
```

Classes

- class [BinaryTree< T >](#)
Definition of the [BinaryTree](#) class.

4.1.1 Detailed Description

Defines the [BinaryTree](#) class.

4.2 binary_tree.h

[Go to the documentation of this file.](#)

```
00001
00005
00006 #pragma once
00007
00008 #include "binary_tree_node.h"
00009 #include "includes.h"
00010
00016 template <typename T> class BinaryTree {
00017 private:
00018     TNode<T> *root; ///< A pointer to the BinaryTree's root.
00019     int size;        ///< The size of the tree
00020
00021 public:
00025     BinaryTree();
00026
00031     bool isEmpty();
00032
00037     int getSize();
```

```

00038
00044     bool insert(T value);
00045
00051     bool deleteNode(T value);
00052
00056     void displayTree();
00057
00062     void displayNode(TNode<T> *root);
00063 };
00064
00065 #include "../templates/binary_tree.hpp"

```

4.3 include/binary_tree_node.h File Reference

Defines the Binary Tree Nodes ([TNode](#)) class.

```
#include "includes.h"
```

Classes

- class [TNode< T >](#)
The nodes for the [BinaryTree](#) class.

4.3.1 Detailed Description

Defines the Binary Tree Nodes ([TNode](#)) class.

4.4 binary_tree_node.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #pragma once
00007
00008 #include "includes.h"
00009
00014 template <typename T> class TNode {
00015 public:
00016     T value;
00017     TNode *right;
00018     TNode *left;
00019     bool is_root;
00020     bool is_right;
00021     bool is_left;
00022     TNode(T value) {
00023
00028
00029         this->value = value;
00030         this->right = this->left = NULL;
00031         this->is_root = this->is_right = this->is_left = false;
00032     }
00033 };

```

4.5 include/double_node.h File Reference

Defines the Double Node ([DNode](#)) class.

```
#include "includes.h"
```

Classes

- class [DNode< T >](#)

The nodes for the [DoublyLinkedList](#) class.

4.5.1 Detailed Description

Defines the Double Node ([DNode](#)) class.

4.6 double_node.h

[Go to the documentation of this file.](#)

```
00001
00005
00006 #pragma once
00007
00008 #include "includes.h"
00009
00015 template <typename T> class DNode {
00016 public:
00017     T value;
00018     DNode *prev;
00019     DNode *next;
00020
00021     DNode(T value) {
00022
00027
00028         this->value = value;
00029         prev = NULL;
00030         next = NULL;
00031     }
00032 };
```

4.7 include/doubly_linked_list.h File Reference

Defines the [DoublyLinkedList](#) class.

```
#include "double_node.h"
#include "../templates/doubly_linked_list.hpp"
```

Classes

- class [DoublyLinkedList< T >](#)

The [DoublyLinkedList](#) class.

4.7.1 Detailed Description

Defines the [DoublyLinkedList](#) class.

4.8 doubly_linked_list.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #pragma once
00007
00008 #include "double_node.h"
00009
00015 template <typename T> class DoublyLinkedList {
00016 private:
00017     DNode<T> *head,
00018     *tail;    ///< Pointers to the head and tail of the DoublyLinkedList object.
00019     int length; ///< The length of the DoublyLinkedList object.
00020
00021 public:
00025     DoublyLinkedList();
00026
00031     bool isEmpty();
00032
00037     int getLength();
00038
00043     DNode<T> *getHead();
00044
00050     bool append(T value);
00051
00058     bool insert(T value, int position);
00059
00065     bool push(T value);
00066
00071     bool removeHead();
00072
00077     bool removeTail();
00078
00084     bool removeNode(DNode<T> *node);
00085
00091     bool deleteNode(int index);
00092
00096     void display();
00097 };
00098
00099 #include "../templates/doubly_linked_list.hpp"

```

4.9 include/entities.h File Reference

Defines the [Student](#) and [Course](#) classes.

```

#include "structures.h"
#include "../templates/student.hpp"
#include "../templates/course.hpp"

```

Classes

- class [Course](#)
The *Course* class.
- class [Student](#)
The *Student* class.

4.9.1 Detailed Description

Defines the [Student](#) and [Course](#) classes.

4.10 entities.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #pragma once
00007
00008 #include "structures.h"
00009
00010 class Student;
00011 class Course;
00012
00017 class Course {
00018 public:
00019     int id;
00020     int credits;
00021     int max_seats;
00022     int seats;
00023     string name;
00024     string instructor;
00025     Stack<Course> *prerequisites;
00026     Queue<Student> *waitlist;
00027
00037     Course(int id, int credits, string name, string instructor, int max_seats,
00038             int seats);
00039
00043     Course();
00044
00050     bool isEligible(Student student);
00051
00058     bool addToWaitlist(Student student);
00059
00065     bool addPrerequisite(Course course);
00066
00070     void displayDetails();
00071
00077     bool operator<(const Course &other) {
00078         return this->id < other.id;
00079     }
00080
00086     bool operator>(const Course &other) {
00087         return this->id > other.id;
00088     }
00089
00095     bool operator==(const Course &other) {
00096         return this->id == other.id;
00097     }
00098
00104     bool operator!=(const Course &other) {
00105         return this->id != other.id;
00106     }
00107
00114     friend ostream &operator<<(std::ostream &os, Course &course);
00115 };
00116
00121 class Student {
00122
00123 public:
00124     string name;
00125     string email;
00126     string password;
00127     string address;
00128     int id;
00129     int phone;
00130     DoublyLinkedList<Course>
00131         *course_history;
00132
00142     Student(int id, string name, string email, string password, string address,
00143             int phone);
00144
00148     Student();
00149
00154     bool alreadyEnrolled(Course course);
00155
00160     bool addCourse(Course *course);
00161
00165     void viewCourses();
00166
00170     void displayDetails();
00171
00177     bool operator!=(const Student &other) {
00178         return this->id != other.id;
00179     }
00180
00187     friend ostream &operator<<(std::ostream &os, Student &student);

```

```

00188 };
00189
00190 #include "../templates/student.tpp"
00191
00192 #include "../templates/course.tpp"

```

4.11 include/hash_table.h File Reference

Defines the [HashTable](#) and [table_pair](#) Classes.

```

#include "includes.h"
#include "singly_linked_list.h"
#include "../templates/hash_table.tpp"

```

Classes

- class [table_pair](#)< K, V >
A container for a key and its value.
- class [HashTable](#)< K, V >
The [HashTable](#) class.

4.11.1 Detailed Description

Defines the [HashTable](#) and [table_pair](#) Classes.

4.12 hash_table.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #pragma once
00007
00008 #include "includes.h"
00009 #include "singly_linked_list.h"
00010
00017 template <typename K, typename V> struct table_pair {
00018     K key;
00019     V value;
00020 };
00021
00028 template <typename K, typename V> class HashTable {
00029 private:
00030     int table_size = 0;
00031     SinglyLinkedList<table_pair<K, V>>
00032         *hash_array;
00033
00039     int hash(K key);
00040
00041 public:
00046     HashTable(int table_size);
00047
00054     bool insert(K key, V value);
00055
00061     V *get(K key);
00062
00068     bool remove(K key);
00069 };
00070
00071 #include "../templates/hash_table.tpp"

```


4.13 include/includes.h File Reference

Collects the included files used throughout the project.

```
#include <cstdint>
#include <iostream>
#include <optional>
#include <string>
```

4.13.1 Detailed Description

Collects the included files used throughout the project.

4.14 includes.h

[Go to the documentation of this file.](#)

```
00001
00005
00006 #pragma once
00007
00008 #include <cstdint>
00009 #include <iostream>
00010 #include <optional>
00011 #include <string>
00012
00013 using namespace std;
```

4.15 include/queue.h File Reference

Defines the [Queue](#) class.

```
#include "includes.h"
#include "single_node.h"
#include "../templates/queue.hpp"
```

Classes

- class [Queue< T >](#)
The [Queue](#) class.

4.15.1 Detailed Description

Defines the [Queue](#) class.

4.16 queue.h

[Go to the documentation of this file.](#)

```
00001
00005
00006 #pragma once
00007
00008 #include "includes.h"
00009 #include "single_node.h"
00010
00016 template <typename T> class Queue {
00017 private:
00018     SNode<T> *rear,
00019     *front; //< Pointers to the front and rear of the Queue object.
00020     int size; //< The size of the Queue object.
00021
00022 public:
00026     Queue();
00027
00032     bool isEmpty();
00033
00038     int getSize();
00039
00045     bool enqueue(T value);
00046
00051     bool dequeue();
00052
00057     T *peek();
00058 };
00059
00060 #include "../templates/queue.tpp"
```

4.17 include/single_node.h File Reference

Defines the Single Node ([SNode](#)) class.

```
#include "includes.h"
```

Classes

- class [SNode< T >](#)
The nodes for the [SinglyLinkedList](#) class.

4.17.1 Detailed Description

Defines the Single Node ([SNode](#)) class.

Defines the [SinglyLinkedList](#) class.

4.18 single_node.h

[Go to the documentation of this file.](#)

```
00001
00005
00006 #pragma once
00007
00008 #include "includes.h"
00009
00015 template <typename T> class SNode {
00016 public:
00017     T value;
00018     SNode *next;
00019
00024     SNode(T value) {
00025         this->value = value;
00026         next = NULL;
00027     }
00028 };
```

4.19 singly_linked_list.h

```

00001
00005
00006 #pragma once
00007
00008 #include "single_node.h"
00009
00015 template <typename T> class SinglyLinkedList {
00016 private:
00017     SNode<T> *head,
00018         *tail;    //< Pointers to the head and tail of the SinglyLinkedList object;
00019     int length;    //< The length of the SinglyLinkedList object;
00020
00021 public:
00025     SinglyLinkedList();
00026
00031     SNode<T> *getHead();
00032
00037     bool isEmpty();
00038
00044     bool append(T value);
00045
00052     bool insert(T value, int position);
00053
00059     bool push(T value);
00060
00066     bool removeNode(SNode<T> *node);
00067
00072     bool removeHead();
00073
00078     bool removeTail();
00079
00085     bool deleteNode(int position);
00086
00090     void display();
00091
00096     int getLength();
00097 };
00098
00099 #include "../templates/singly_linked_list.tpp"

```

4.20 include/stack.h File Reference

Defines the [Stack](#) class.

```

#include "includes.h"
#include "single_node.h"
#include "../templates/stack.tpp"

```

Classes

- class [Stack< T >](#)
The [Stack](#) class.

4.20.1 Detailed Description

Defines the [Stack](#) class.

4.21 stack.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #pragma once
00007
00008 #include "includes.h"
00009 #include "single_node.h"
00010
00016 template <typename T> class Stack {
00017 private:
00018     SNode<T> *stack_top; //< A pointer to the top of the Stack object.
00019     int size;           //< The size of the Stack object.
00020
00021 public:
00025     Stack();
00026
00031     bool isEmpty();
00032
00037     int getSize();
00038
00044     bool push(T object);
00045
00050     bool pop();
00051
00056     T peek();
00057 };
00058
00059 #include "../templates/stack.tpp"

```

4.22 include/structures.h File Reference

Collects all the implemented data structures in one file.

```

#include "binary_tree.h"
#include "doubly_linked_list.h"
#include "hash_table.h"
#include "queue.h"
#include "singly_linked_list.h"
#include "stack.h"

```

4.22.1 Detailed Description

Collects all the implemented data structures in one file.

4.23 structures.h

[Go to the documentation of this file.](#)

```

00001
00005
00006 #pragma once
00007
00008 #include "binary_tree.h"
00009 #include "doubly_linked_list.h"
00010 #include "hash_table.h"
00011 #include "queue.h"
00012 #include "singly_linked_list.h"
00013 #include "stack.h"

```

4.24 include/system.h File Reference

Defines the University System ([UniSystem](#)) class.

```
#include "entities.h"
#include "structures.h"
#include "../templates/system.hpp"
```

Classes

- class [UniSystem](#)
The *UniSystem* class.

4.24.1 Detailed Description

Defines the University System ([UniSystem](#)) class.

4.25 system.h

[Go to the documentation of this file.](#)

```
00001
00005
00006 #pragma once
00007
00008 #include "entities.h"
00009 #include "structures.h"
00010
00015 class UniSystem {
00016 private:
00017     SinglyLinkedList<Student>
00018         *students;
00019     BinaryTree<Course>
00020         *courses;
00021
00022 public:
00023     HashTable<int, Course>
00024         *courses_table;
00025     HashTable<int, Student>
00026         *students_table;
00027
00031     UniSystem();
00032
00038     bool courseExists(int id);
00039
00045     bool studentExists(int id);
00046
00057     bool addStudent(int id, string name, string email, string password,
00058                     string address, int phone);
00059
00065     bool deleteStudent(int id);
00066
00070     void listStudents();
00071
00075     void listCourses();
00076
00088     bool addCourse(int id, string name, int credits, string instructor,
00089                   int max_seats, int seats);
00090
00096     bool addCourse(Course course);
00097
00103     bool dropCourse(int id);
00104
00110     bool checkWaitlist(Course &course);
00111
00117     bool searchStudent(int id);
00118
00124     bool searchCourse(int id);
00125 };
00126
00127 #include "../templates/system.hpp"
```

4.26 src/main.cpp File Reference

Main program code.

```
#include "../include/entities.h"
#include "../include/includes.h"
#include "../include/structures.h"
#include "../include/system.h"
```

Macros

- `#define slls SinglyLinkedList<string>`
SinglyLinkedList containing strings.

Functions

- `bool runCommand (slls *commands)`
- `slls * splitInput (string input)`
- `string strip (string arg)`
- `bool add (string arg)`
- `bool search (string arg)`
- `bool view (string arg)`
- `bool remove (string arg)`
- `bool enroll ()`
- `void testData ()`
- `bool freeSeat ()`
- `void printHelp ()`
- `void loop ()`
- `int main ()`

The main driver code for the program.

Variables

- `UniSystem us`
The main system object.

4.26.1 Detailed Description

Main program code.

4.26.2 Function Documentation

4.26.2.1 add()

```
bool add (  
    string arg)
```

A function that adds students, courses, or prerequisites in their respective data structures.

Parameters

<i>arg</i>	The argument string.
------------	----------------------

Returns

boolean for debugging purposes.

4.26.2.2 enroll()

```
bool enroll ()
```

A function that adds courses to students' enrollment histories.

Returns

boolean for debugging purposes.

4.26.2.3 freeSeat()

```
bool freeSeat ()
```

a function that frees course seats.

Returns

boolean for debugging purposes.

4.26.2.4 loop()

```
void loop ()
```

The main program loop.

4.26.2.5 main()

```
int main ()
```

The main driver code for the program.

Returns

0

4.26.2.6 printHelp()

```
void printHelp ()
```

A function that prints a help manual.

4.26.2.7 remove()

```
bool remove (  
    string arg)
```

A function that removes students or courses from their respective data structures.

Parameters

<i>arg</i>	The object to remove.
------------	-----------------------

Returns

boolean for debugging purposes.

4.26.2.8 runCommand()

```
bool runCommand (  
    slls * commands)
```

The function responsible for running commands.

Parameters

<i>commands</i>	The pointer to the SLL with the command as well as the arguments.
-----------------	---

Returns

A boolean for debugging purposes.

4.26.2.9 search()

```
bool search (  
    string arg)
```

A function that searches the respective hash table for students or courses.

Parameters

<i>arg</i>	The argument to search for.
------------	-----------------------------

Returns

boolean for debugging purposes.

4.26.2.10 splitInput()

```
slls * splitInput (  
    string input)
```

Splits the string input and stores it in a linked list.

Parameters

<i>input</i>	The input string.
--------------	-------------------

Returns

A singly linked list with each node holding a part of the string.

4.26.2.11 strip()

```
string strip (  
    string arg)
```

strips all whitespaces in a string.

Parameters

<i>arg</i>	The string.
------------	-------------

Returns

A string free of whitespace.

4.26.2.12 testData()

```
void testData ()
```

A function that adds dummy data for testing purposes

4.26.2.13 view()

```
bool view (  
    string arg)
```

A function that lists students, courses, or enrollment histories.

Parameters

<i>arg</i>	The object to list.
------------	---------------------

Returns

boolean for debugging purposes.

Index

add
 main.cpp, 46
addCourse
 Student, 27
 UniSystem, 31
addPrerequisite
 Course, 9
addStudent
 UniSystem, 32
addToWaitlist
 Course, 9
alreadyEnrolled
 Student, 27
append
 DoublyLinkedList< T >, 13
 SinglyLinkedList< T >, 20

BinaryTree< T >, 5
 deleteNode, 6
 displayNode, 6
 getSize, 6
 insert, 6
 isEmpty, 7

checkWaitlist
 UniSystem, 32
Course, 7
 addPrerequisite, 9
 addToWaitlist, 9
 Course, 8
 isEligible, 9
 operator!=, 9
 operator<, 10
 operator<<, 11
 operator>, 10
 operator==, 10
courseExists
 UniSystem, 32

deleteNode
 BinaryTree< T >, 6
 DoublyLinkedList< T >, 13
 SinglyLinkedList< T >, 21
deleteStudent
 UniSystem, 33
dequeue
 Queue< T >, 18
displayNode
 BinaryTree< T >, 6
DNode
 DNode< T >, 12
DNode< T >, 11
 DNode, 12
DoublyLinkedList< T >, 12
 append, 13
 deleteNode, 13
 getHead, 13
 getLength, 14
 insert, 14
 isEmpty, 14
 push, 14
 removeHead, 15
 removeNode, 15
 removeTail, 15
dropCourse
 UniSystem, 33

enqueue
 Queue< T >, 18
enroll
 main.cpp, 47

freeSeat
 main.cpp, 47

get
 HashTable< K, V >, 16
getHead
 DoublyLinkedList< T >, 13
 SinglyLinkedList< T >, 21
getLength
 DoublyLinkedList< T >, 14
 SinglyLinkedList< T >, 21
getSize
 BinaryTree< T >, 6
 Queue< T >, 19
 Stack< T >, 25

HashTable
 HashTable< K, V >, 16
HashTable< K, V >, 16
 get, 16
 HashTable, 16
 insert, 17
 remove, 17

include/binary_tree.h, 35
include/binary_tree_node.h, 36
include/double_node.h, 36, 37
include/doubly_linked_list.h, 37, 38
include/entities.h, 38, 39

- include/hash_table.h, 40
- include/includes.h, 41
- include/queue.h, 41, 42
- include/single_node.h, 42
- include/singly_linked_list.h, 43
- include/stack.h, 43, 44
- include/structures.h, 44
- include/system.h, 45
- insert
 - BinaryTree< T >, 6
 - DoublyLinkedList< T >, 14
 - HashTable< K, V >, 17
 - SinglyLinkedList< T >, 21
- isEligible
 - Course, 9
- isEmpty
 - BinaryTree< T >, 7
 - DoublyLinkedList< T >, 14
 - Queue< T >, 19
 - SinglyLinkedList< T >, 22
 - Stack< T >, 25
- loop
 - main.cpp, 47
- main
 - main.cpp, 47
- main.cpp
 - add, 46
 - enroll, 47
 - freeSeat, 47
 - loop, 47
 - main, 47
 - printHelp, 47
 - remove, 47
 - runCommand, 48
 - search, 48
 - splitInput, 48
 - strip, 49
 - testData, 49
 - view, 49
- operator!=
 - Course, 9
 - Student, 28
- operator<
 - Course, 10
- operator<<
 - Course, 11
 - Student, 28
- operator>
 - Course, 10
- operator==
 - Course, 10
- peek
 - Queue< T >, 19
 - Stack< T >, 25
- pop
 - Stack< T >, 25
- printHelp
 - main.cpp, 47
- push
 - DoublyLinkedList< T >, 14
 - SinglyLinkedList< T >, 22
 - Stack< T >, 25
- Queue< T >, 18
 - dequeue, 18
 - enqueue, 18
 - getSize, 19
 - isEmpty, 19
 - peek, 19
- remove
 - HashTable< K, V >, 17
 - main.cpp, 47
- removeHead
 - DoublyLinkedList< T >, 15
 - SinglyLinkedList< T >, 22
- removeNode
 - DoublyLinkedList< T >, 15
 - SinglyLinkedList< T >, 22
- removeTail
 - DoublyLinkedList< T >, 15
 - SinglyLinkedList< T >, 23
- runCommand
 - main.cpp, 48
- search
 - main.cpp, 48
- searchCourse
 - UniSystem, 33
- searchStudent
 - UniSystem, 34
- SinglyLinkedList< T >, 20
 - append, 20
 - deleteNode, 21
 - getHead, 21
 - getLength, 21
 - insert, 21
 - isEmpty, 22
 - push, 22
 - removeHead, 22
 - removeNode, 22
 - removeTail, 23
- SNode
 - SNode< T >, 24
- SNode< T >, 23
 - SNode, 24
- splitInput
 - main.cpp, 48
- src/main.cpp, 46
- Stack< T >, 24
 - getSize, 25
 - isEmpty, 25
 - peek, 25
 - pop, 25

- push, [25](#)
- strip
 - main.cpp, [49](#)
- Student, [26](#)
 - addCourse, [27](#)
 - alreadyEnrolled, [27](#)
 - operator!=, [28](#)
 - operator<<, [28](#)
 - Student, [27](#)
- studentExists
 - UniSystem, [34](#)
- table_pair< K, V >, [28](#)
- testData
 - main.cpp, [49](#)
- TNode
 - TNode< T >, [30](#)
- TNode< T >, [29](#)
 - TNode, [30](#)
- UniSystem, [30](#)
 - addCourse, [31](#)
 - addStudent, [32](#)
 - checkWaitlist, [32](#)
 - courseExists, [32](#)
 - deleteStudent, [33](#)
 - dropCourse, [33](#)
 - searchCourse, [33](#)
 - searchStudent, [34](#)
 - studentExists, [34](#)
- view
 - main.cpp, [49](#)