University System

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

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

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

T	the type of values stored in the BinaryTree 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

value	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

∗*root*	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

value	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 addPrequisite (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

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

3.2.3 Member Function Documentation

3.2.3.1 addPrequisite()

```
bool Course::addPrequisite (
             Course course)
```

A function that adds prerequisites to this course.

Parameters

course	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

student	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

student	The student to enroll.

Returns

boolean.

3.2.3.4 operator"!=()

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

Inequality operator.

Parameters

other	The objects to compare.

Returns

boolean.

3.2.3.5 operator<()

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

Less Than operator.

Parameters

other	The object to compare

Returns

boolean.

3.2.3.6 operator==()

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

Equality operator.

Parameters

other	The object to compare.

Returns

boolean.

3.2.3.7 operator>()

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

Greater Than operator.

Parameters

other	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

os	The input stream.
course	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 begining 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

T	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

value	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

index	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

value	The value to add.
position	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 begining of the list.

Parameters

value	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

node	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

K	The type of key values.
V	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

table_size	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

key	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

key	the key of the value.
value	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

key	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

T	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

value	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 begining 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

T	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

value	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

position	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

value	The value to add.
position	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 begining of the list.

Parameters

value	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

node	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

object	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

id	the student's ID.
name	this student's name.
email	this student's email.
password	this student's password.
address	this student's address.
phone	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

course	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

course	the course this student intends to be enrolled in.

3.10.3.3 operator"!=()

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

Inequality Operator.

Parameters

other	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

os	The input stream.
student	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

K	The type of key values.
V	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

value	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

course	Course 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

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

id	The student's ID.
name	This student's name.
email	This student's email.
password	This student's password.
address	This student's address.
phone	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

course	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

id	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

id	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

id	The course ID.

Returns

boolean.

3.13.2.8 searchCourse()

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

A function that displays a specific course details.

Parameters

id	Course ID.

Returns

boolean.

3.13.2.9 searchStudent()

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

A function that displays a specific student details.

Parameters

id	Student ID.

Returns

boolean.

3.13.2.10 studentExists()

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

A function that checks if a student exists.

Parameters

id	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.tpp"
```

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.

```


#pragma once

#include "binary_tree_node.h"
#include "includes.h"

template <typename T> class BinaryTree {
private:
  TNode<T> *root; //< A pointer to the BinaryTree's root.
  int size;       //< The size of the tree

public:
  BinaryTree();

  bool isEmpty();

  int getSize();

  bool insert(T value);

  bool deleteNode(T value);

  void displayTree();

  void displayNode(TNode<T> *root);
};

#include "../templates/binary_tree.tpp"
```

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.

```


#pragma once

#include "includes.h"

template <typename T> class TNode {
public:
  T value;
  TNode *right;
  TNode *left;
  bool is_root;
  bool is_right;
  bool is_left;
  TNode(T value) {


    this->value = value;
    this->right = this->left = NULL;
    this->is_root = this->is_right = this->is_left = false;
  }
};
```

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.

```
#pragma once

#include "includes.h"

template <typename T> class DNode {
public:
  T value;
  DNode *prev;
  DNode *next;

  DNode(T value) {


    this->value = value;
    prev = NULL;
    next = NULL;
  }
};
```

4.7 include/doubly_linked_list.h File Reference

Defines the DoublyLinkedList class.

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

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.

```
#pragma once

#include "double_node.h"

template <typename T> class DoublyLinkedList {
private:
  DNode<T> *head,
      *tail;  //< Pointers to the head and tail of the DoublyLinkedList object.
  int length; //< The length of the DoublyLinkedList object.

public:
  DoublyLinkedList();

  bool isEmpty();

  int getLength();

  DNode<T> *getHead();

  bool append(T value);

  bool insert(T value, int position);

  bool push(T value);

  bool removeHead();

  bool removeTail();

  bool removeNode(DNode<T> *node);

  bool deleteNode(int index);

  void display();
};

#include "../templates/doubly_linked_list.tpp"
```

4.9 include/entities.h File Reference

Defines the Student and Course classes.

```
#include "structures.h"
#include "../templates/student.tpp"
#include "../templates/course.tpp"
```

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.

```


#pragma once

#include "structures.h"

class Student;
class Course;

class Course {
public:
  int id;
  int credits;
  int max_seats;
  int seats;
  string name;
  string instructor;
  Stack<Course> *prerequisites;
  Queue<Student> *waitlist;

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

  Course();

  bool isEligible(Student student);

  bool addToWaitlist(Student student);

  bool addPrequisite(Course course);

  void displayDetails();

  bool operator<(const Course &other) {
    return this->id < other.id;
  }

  bool operator>(const Course &other) {
    return this->id > other.id;
  }

  bool operator==(const Course &other) {
    return this->id == other.id;
  }

  bool operator!=(const Course &other) {
    return this->id != other.id;
  }

  friend ostream &operator«(std::ostream &os, Course &course);
};

class Student {

public:
  string name;
  string email;
  string password;
  string address;
  int id;
  int phone;
  DoublyLinkedList<Course>
      *course_history;

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

  Student();

  bool alreadyEnrolled(Course course);

  bool addCourse(Course *course);

  void viewCourses();

  void displayDetails();

  bool operator!=(const Student &other) {
    return this->id != other.id;
  }

  friend ostream &operator«(std::ostream &os, Student &student);
};

#include "../templates/student.tpp"

#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.

```


#pragma once

#include "includes.h"
#include "singly_linked_list.h"

template <typename K, typename V> struct table_pair {
  K key;
  V value;
};

template <typename K, typename V> class HashTable {
private:
  int table_size = 0;
  SinglyLinkedList<table_pair<K, V>>
      *hash_array;

  int hash(K key);

public:
  HashTable(int table_size);

  bool insert(K key, V value);

  V *get(K key);

  bool remove(K key);
};

#include "../templates/hash_table.tpp"
```

4.13 include/includes.h File Reference

Collects the included files used throughout the project.

```
#include <cstddef>
#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 <cstddef>
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.tpp"
```

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.

```


#pragma once

#include "includes.h"
#include "single_node.h"

template <typename T> class Queue {
private:
  SNode<T> *rear,
      *front; //< Pointers to the front and rear of the Queue object.
  int size;   //< The size of the Queue object.

public:
  Queue();

  bool isEmpty();

  int getSize();

  bool enqueue(T value);

  bool dequeue();

  T *peek();
};

#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.

```


#pragma once

#include "includes.h"

template <typename T> class SNode {
public:
  T value;
  SNode *next;

  SNode(T value) {
    this->value = value;
    next = NULL;
  }
};
```

4.19 singly_linked_list.h

```

#pragma once

#include "single_node.h"

template <typename T> class SinglyLinkedList {
private:
  SNode<T> *head,
      *tail;  //< Pointers to the head and tail of the SinglyLinkedList object;
  int length; //< The length of the SinglyLinkedList object;

public:
  SinglyLinkedList();

  SNode<T> *getHead();

  bool isEmpty();

  bool append(T value);

  bool insert(T value, int position);

  bool push(T value);

  bool removeNode(SNode<T> *node);

  bool removeHead();

  bool removeTail();

  bool deleteNode(int position);

  void display();

  int getLength();
};

#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.

```
#pragma once

#include "includes.h"
#include "single_node.h"

template <typename T> class Stack {
private:
  SNode<T> *stack_top; //< A pointer to the top of the Stack object.
  int size;            //< The size of the Stack object.

public:
  Stack();

  bool isEmpty();

  int getSize();

  bool push(T object);

  bool pop();

  T peek();
};

#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.

```
#pragma once

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

4.24 include/system.h File Reference

Defines the University System (UniSystem) class.

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

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.

```

#pragma once

#include "entities.h"
#include "structures.h"

class UniSystem {
private:
  SinglyLinkedList<Student>
      *students;
  BinaryTree<Course>
      *courses;

public:
  HashTable<int, Course>
      *courses_table;
  HashTable<int, Student>
      *students_table;

  UniSystem();

  bool courseExists(int id);

  bool studentExists(int id);

  bool addStudent(int id, string name, string email, string password,
                  string address, int phone);

  bool deleteStudent(int id);

  void listStudents();

  void listCourses();

  bool addCourse(int id, string name, int credits, string instructor,
                 int max_seats, int seats);

  bool addCourse(Course course);

  bool dropCourse(int id);

  bool checkWaitlist(Course &course);

  bool searchStudent(int id);

  bool searchCourse(int id);
};

#include "../templates/system.tpp"
```

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

arg	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

arg	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

commands	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

arg	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

input	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

arg	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

arg	The object to list.

Returns

boolean for debugging purposes.

Index