Maze

Generated by Doxygen 1.9.8

1 Class Index

1.1 Class List

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

2 Class Documentation

2.1 Cell Class Reference

```
#include <Cell.hpp>
```

Public Member Functions

- Cell (int r=0, int c=0)
- Cell (const Cell &copy)
- Cell & operator= (const Cell &copy)

Public Attributes

- int row
- int col

Friends

- bool operator== (const Cell &c1, const Cell &c2)
- bool operator!= (const Cell &c1, const Cell &c2)
- ostream & operator<< (ostream &out, const Cell cell)

2.1.1 Detailed Description

A Cell is a single grid on a 2-dimensional array, identified by its row and column number.

2.1.2 Constructor & Destructor Documentation

Cell() [1/2]

```
Cell::Cell (
            int r = 0,
            int c = 0 )  [inline]
```

Creates a single cell.

Parameters

r	the row index (0 ... rows-1)
c	the col index (0 ... cols-1)

Cell() [2/2]

```
Cell::Cell (
            const Cell & copy )  [inline]
```

Creates a duplicate cell.

Parameters

copy	the cell to copy.

2.1.3 Member Function Documentation

operator=()

```
Cell & Cell::operator= (
            const Cell & copy )  [inline]
```

Allows the assignment operator to make a copy.

Parameters

copy	the Cell to be copied.

Returns

the copied Cell.

2.1.4 Friends And Related Symbol Documentation

operator"!=

```
bool operator!= (
            const Cell & c1,
            const Cell & c2 )  [friend]
```

Determines inequality of two cells.

Parameters

c1	one of the cells.
c2	the other cell.

Returns

true if they not equivalent.

operator<<

```
ostream & operator<< (
            ostream & out,
            const Cell cell )  [friend]
```

Loads an output stream with the cell.

Parameters

out	the output stream.
cell	the Cell.

Returns

the loaded output stream.

operator==

```
bool operator== (
            const Cell & c1,
            const Cell & c2 )  [friend]
```

Compares two cells for equivalence: both row and col are equal.

Parameters

c1	one of the cells.
c2	the other cell.

Returns

true if they are equivalent.

2.1.5 Member Data Documentation

col

```
int Cell::col
```

The column 0...cols-1 index of the cell.

row

```
int Cell::row
```

The row 0...rows-1 index of the cell.

2.2 Dispenser< E > Class Template Reference

```
#include <Dispenser.hpp>
```

Public Member Functions

- Dispenser ()
- Dispenser (const Dispenser< E > &copy)
- Dispenser & operator= (const Dispenser< E > &copy)
- Dispenser (Dispenser &&replace)
- Dispenser & operator= (Dispenser &&replace)
- ~Dispenser ()
- E top () const
- E bottom () const
- bool is_empty () const
- void push (E &element)
- E pop ()
- E dispense ()
- void clear ()
- string elements_bottom_up () const
- string elements_top_down () const

2.2.1 Detailed Description

template<typename E>
class Dispenser< E >

The Dispenser (sometimes called a Deque, for 'double-ended queue') is a combination of the stack and queue, where inserts and removals can occur at either end of the sequence. For our particular purposes, we use the dispenser object as a stack to find a path and then 'dispense' the resulting stack from the bottom. This effect allows for the bottom of the stack to act as a queue, Thanks to Bruce for the name, which has more characters to type, but is a more suitable name than 'Deque'.

Author

: B. Bultena for VIU CSCI 161 Spring 2024

2.2.2 Constructor & Destructor Documentation

Dispenser() [1/3]

```
template<typename E >
Dispenser< E >::Dispenser ( )
```

Creates an empty dispenser.

Dispenser() [2/3]

```
template<typename E >
Dispenser< E >::Dispenser (
            const Dispenser< E > & copy )
```

Creates a dispenser that is an exact (deep) copy.

Parameters

copy	The exact copy.

Dispenser() [3/3]

```
template<typename E >
Dispenser< E >::Dispenser (
            Dispenser< E > && replace )
```

Constructs a dispenser that replaces the input dispenser.

Parameters

replace	the dispenser to replace (which will be rendered inactive).

~Dispenser()

```
template<typename E >
Dispenser< E >::~Dispenser ( )
```

Frees all the memory that was allocated to this sequence.

2.2.3 Member Function Documentation

bottom()

```
template<typename E >
E Dispenser< E >::bottom ( ) const
```

'Looks at' the bottom element, but does not remove it.

Returns

the element that was first inserted into the dispenser (simulating a queue or providing access to the bottom of the stack).

Exceptions

DispenserException	if the dispenser is empty.

clear()

```
template<typename E >
void Dispenser< E >::clear ( )
```

Empties the Dispenser.

dispense()

```
template<typename E >
E Dispenser< E >::dispense ( )
```

Removes the element at the bottom of the stack (the earliest element that was pushed).

Returns

the element that is removed.

Exceptions

DispenserException	if the dispnser is empty.

elements_bottom_up()

```
template<typename E >
string Dispenser< E >::elements_bottom_up ( ) const
```

Creates a string representation of all the elements, from the bottom of the stack to the top (all one line, with a space between the individual elements).

Returns

the string

elements_top_down()

```
template<typename E >
string Dispenser< E >::elements_top_down ( ) const
```

Creates a string representation of all the elements, from the top of the stack to the bottom. (all one line, with a space between the individual elements).

Returns

the string

is_empty()

```
template<typename E >
bool Dispenser< E >::is_empty ( ) const
```

Returns

true if the dispenser is currently empty.

operator=() [1/2]

```
template<typename E >
Dispenser & Dispenser< E >::operator= (
            const Dispenser< E > & copy )
```

Allows the assignment operator to create a copy.

Parameters

copy	The dispenser to copy.

Returns

the address of the newly copied dispenser.

operator=() [2/2]

```
template<typename E >
Dispenser & Dispenser< E >::operator= (
            Dispenser< E > && replace )
```

Allows the assignment operator to replace the incoming dispenser.

Parameters

replace	the dispenser to replace (which will be rendered inactive).

Returns

the address of the replacement.

pop()

```
template<typename E >
E Dispenser< E >::pop ( )
```

Removes the element at the top of the stack (the last element that was pushed).

Returns

the element that is removed.

Exceptions

DispenserException	if the dispenser is empty.

push()

```
template<typename E >
void Dispenser< E >::push (
            E & element )
```

Inserts an element at the top (like a stack).

Parameters

element	the element to 'push'.

top()

```
template<typename E >
E Dispenser< E >::top ( ) const
```

'Looks at' the top element, but does not remove it.

Returns

the element that was last inserted into the dispenser (simulating a stack).

Exceptions

DispenserException	if the dispenser is empty.

2.3 DispenserException Class Reference

```
#include <Dispenser.hpp>
```

Public Member Functions

- DispenserException (const string &where_thrown, const string &msg)
- string to_string () const

Friends

- ostream & operator<< (ostream &out, const DispenserException &de)

2.3.1 Detailed Description

An exception class, specific to the Dispenser class.

2.3.2 Constructor & Destructor Documentation

DispenserException()

```
DispenserException::DispenserException (
            const string & where_thrown,
            const string & msg )  [inline]
```

Creates the exception.

Parameters

where_thrown	the location of the function identifies an exception.
msg	the cause of the exception: usually that the Dispenser is empty.

2.3.3 Member Function Documentation

to_string()

```
string DispenserException::to_string ( ) const  [inline]
```

Creates a string representation of the exception that includes where_thrown and the msg.

Returns

the statement as described above.

2.3.4 Friends And Related Symbol Documentation

operator<<

```
ostream & operator<< (
            ostream & out,
            const DispenserException & de )  [friend]
```

Allows the string representation to be loaded onto an ostream object.

Parameters

out	the ostream object to load.
de	The exception.

Returns

the loaded ostream object.

2.4 Maze Class Reference

```
#include <Maze.hpp>
```

Public Member Functions

- Maze (string *textmaze, int num_strings, Cell start, Cell finish)
- ~Maze ()
- **Maze** (Maze &copy)=delete
- Dispenser< Cell > & solve ()
- string to_string ()

Friends

- ostream & operator<< (ostream &out, Maze &maze)

2.4.1 Detailed Description

A simple maze with either no solution or a solution that does not require any looping.

Author

B. Bultena for VIU CSCI 161 Spring 2024

2.4.2 Constructor & Destructor Documentation

Maze()

```
Maze::Maze (
            string * textmaze,
            int num_strings,
            Cell start,
            Cell finish )
```

Creates a maze from a list of strings containing only {'+"|"-'} characters that represent the walls of the maze and ' '(blank characters) representing the open corridors.

Parameters

textmaze	the array of such strings, each of equal length.
num_strings	the number of strings (rows of the maze).
start	the cell at the top of the maze that indicates the opening for the start of the solution path.
finish	the cell at the bottom of the maze that indicates the opening for the end of the solution path.

~Maze()

```
Maze::~Maze ( )
```

Removes any memory associated with the maze.

2.4.3 Member Function Documentation

solve()

```
Dispenser< Cell > & Maze::solve ( )
```

Solves the maze by finding a path and storing it as a stack of Cell objects inside a Dispenser. Note that the stack, when emptied from the bottom produces the path in order. If there is no path, then the dispenser is empty.

Returns

the dispenser of Cells.

to_string()

```
string Maze::to_string ( )
```

A string representation of a the maze as a grid of characters. The current path is represented as a trail of '#' characters. However, any character (other than {'-','|','+'} can be used to represent a path taken.

Returns

the complete string that, when printed will represent the maze.

2.4.4 Friends And Related Symbol Documentation

operator<<

```
ostream & operator<< (
            ostream & out,
            Maze & maze )  [friend]
```

Overloads the << operator to direct a stream to an ostream object.

Parameters

out	the ostream object.
maze	The maze to represent.

Returns

the ostream object that is loaded with output from the to_string function.

Index