SFST and Friends

Generated by Doxygen 1.8.3

Mon Jun 24 2013 17:14:51

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

Module Documentation

4.1 Regex

Regex extension functions

4.2 Data

Underlying data structures

4.3 Helper

Chapter 5

Class Documentation

5.1 FSTGraph Class Reference

A class to keep track of all of the interconnected FST nodes.

`#include <graph.h>`

Public Member Functions

- FSTGraph ()

  Constructor for FSTGraph.
- FSTGraph (string fn)

  Constructor for FSTGraph.
- ∼FSTGraph ()

  Destructor for FSTGraph.
- void readFile (string fn)

  Reads in a file that has data about an xfst net.

5.1.1 Detailed Description

A class to keep track of all of the interconnected FST nodes.

Because states are really not frequently added or removed, there is not any structure storing any nodes but final and start states. There are four Markov models keeping track of data as well, for forward and backwards directions.

Author

Kip Price

Date

10 June 2012

5.1.2 Constructor & Destructor Documentation

5.1.2.1 FSTGraph::FSTGraph ( ) `[inline]`

Constructor for FSTGraph.

Garbage state Markov model that keeps track of characters Markov model that keeps track of morphemes

5.1.2.2 FSTGraph::FSTGraph (string *fn*) `[inline]`

Constructor for FSTGraph.

Parameters

fn	The file from which to draw the information for the FST model

5.1.3 Member Function Documentation

5.1.3.1 void FSTGraph::readFile (string *fn*) `[inline]`

Reads in a file that has data about an xfst net.

Result of the xfst command print net. Specifically designed for that

Parameters

fn	The name of the file being read in

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

- Classes/graph.h

5.2 FSTNode Class Reference

Class to store data about connections of FST.

`#include <graph.h>`

Public Member Functions

- FSTNode ()

  Constructor for the node.
- FSTNode (string s, string d)

  Constructor for the node.
- FSTNode (string s, string d, FSTNode ∗n, FSTNode ∗p)

  Constructor for the node.
- ∼FSTNode ()

  Destructor for the node.
- Set< FSTNode ∗ > ∗ getNext ()

  Gets the list of possible nodes to go to.
- FSTNode ∗ getPrev ()

  Gets the previous node.
- string getTS ()

  Gets the string to be matched.
- string getTransition ()

  Gets the string to be matched.
- string getTD ()

  Gets the string to tranduce to.
- string getTransduction ()

  Gets the string to transduce to.
- int getCount ()

Gets the count of this node.

- void setTS (string s)

  Set the string to match to.
- void setTransition (string s)

  Set the string to match to.
- void setTD (string d)

  Set the string to transduce to.
- void setTransduction (string d)

  Set the string to transduce to.
- void setPrev (FSTNode ∗p)

  Set the previous node.
- void addNext (FSTNode ∗f)

  Add a new node to possibly go to.
- Set< FSTNode ∗ > ∗ getProb (string s)

  Gets the most probable next node for the string given.

5.2.1 Detailed Description

Class to store data about connections of FST.

Entirely string based, with weights that allow probability calculations. Essentially equivalent to a multidimensional linked list

Author

Kip Price

Date

10 June 2013

5.2.2 Constructor & Destructor Documentation

5.2.2.1 FSTNode::FSTNode ( ) `[inline]`

Constructor for the node.

Whether the output will include the input

5.2.2.2 FSTNode::FSTNode (string *s*, string *d*) `[inline]`

Constructor for the node.

Parameters

s	The string that will be matched
d	The output to transduce to

5.2.2.3 FSTNode::FSTNode (string *s*, string *d*, FSTNode ∗ *n*, FSTNode ∗ *p*) `[inline]`

Constructor for the node.

Parameters

s	The string that will be matched
d	The output to transduce to
n	The next state to transition to
p	The state we came from

5.2.3 Member Function Documentation

5.2.3.1 void FSTNode::addNext (FSTNode ∗ *f*) `[inline]`

Add a new node to possibly go to.

Parameters

f	A new next FSTNode

5.2.3.2 int FSTNode::getCount ( ) `[inline]`

Gets the count of this node.

Returns

The count of this node

5.2.3.3 Set<FSTNode∗>∗ FSTNode::getNext ( ) `[inline]`

Gets the list of possible nodes to go to.

Returns

The list of possible next nodes

5.2.3.4 FSTNode∗ FSTNode::getPrev ( ) `[inline]`

Gets the previous node.

Returns

The previous node

5.2.3.5 Set<FSTNode∗>∗ FSTNode::getProb (string *s*) `[inline]`

Gets the most probable next node for the string given.

Finds the most probable next node using three different methods: straight counts, Markov predicting of characters, and Markov moeling of morphemes

Parameters

s	The string that is being inputted

Returns

The list of nodes, ranked by probability

5.2.3.6 string FSTNode::getTD ( ) [inline]

Gets the string to tranduce to.

Returns

The string to transduce to

5.2.3.7 string FSTNode::getTransduction ( ) [inline]

Gets the string to transduce to.

Returns

The string to transduce to

5.2.3.8 string FSTNode::getTransition ( ) [inline]

Gets the string to be matched.

Returns

The string to be matched

5.2.3.9 string FSTNode::getTS ( ) [inline]

Gets the string to be matched.

Returns

The string to be matched

5.2.3.10 void FSTNode::setPrev (FSTNode * *p*) [inline]

Set the previous node.

Parameters

The	previous node

5.2.3.11 void FSTNode::setTD (string *d*) [inline]

Set the string to transduce to.

Parameters

d	The string to transduce to

5.2.3.12 void FSTNode::setTransduction (string *d*) [inline]

Set the string to transduce to.

Parameters

d	The string to transduce to

5.2.3.13 void FSTNode::setTransition (string *s*) [inline]

Set the string to match to.

Parameters

s	The new string to match to

5.2.3.14 void FSTNode::setTS (string *s*) [inline]

Set the string to match to.

Parameters

s	The new string to match to

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

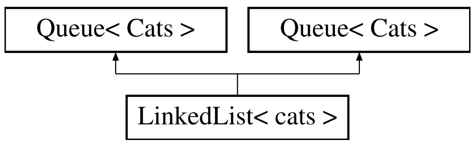
- Classes/graph.h

5.3 LinkedList< cats > Class Template Reference

Simple Linked List based on a queue.

`#include <dataStruct.h>`

Inheritance diagram for LinkedList< cats >:



Public Member Functions

- Cats get (int i)

  Gets the element at the given index.
- N< Cats > ∗ getNode (int i)

  Gets the node at a given index.
- Cats remove (int i)

  Removes the node at the given index.

5.3.1 Detailed Description

template<typename cats>class LinkedList< cats >

Simple Linked List based on a queue.

Author

Kip Price

Date

20 September 2012

Template Parameters

Cats	Any type

5.3.2 Member Function Documentation

5.3.2.1 template<typename cats > Cats LinkedList< cats >::get (int *i*) `[inline]`

Gets the element at the given index.

Parameters

i	The index to retrieve from

Returns

The data at that index

5.3.2.2 template<typename cats > N<Cats>* LinkedList< cats >::getNode (int *i*) `[inline]`

Gets the node at a given index.

Parameters

i	The index to retrieve from

Returns

The node at that index

5.3.2.3 template<typename cats > Cats LinkedList< cats >::remove (int *i*) `[inline]`

Removes the node at the given index.

Parameters

i	The index to remove

Returns

The data at that index

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

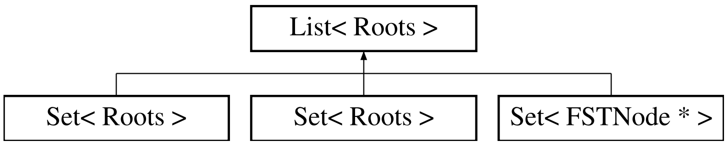
- Classes/dataStruct.h

5.4 List< Roots > Class Template Reference

Basic extendable list structure, similar to vector.

`#include <dataStruct.h>`

Inheritance diagram for List< Roots >:



Public Member Functions

- List ()

    Constructs a List.
- ∼List ()

    Destructs a list.
- void add (Roots r)

    Adds an element.
- void insert (Roots r, int i)

    Inserts an element.
- void move (int i, int j)

    Moves an element from one index to another.
- bool contains (Roots r)

    Checks containment of an element.
- Roots get (int i)

    Fetches an element.
- Roots operator[ ] (int val)

    Operator for array indices.
- int find (Roots r)

    Find an element.
- Roots remove (int i)

    Remove an element.
- bool fRemove (Roots r)

    Removes an element.
- int getSize ()

    Returns the size of the list.
- void addAll (List &ls)

    Adds all elements from another list.
- void addAll (List ∗ls)

    Adds all elements from another list.
- List< Roots > ∗ **intersect** (List< Roots > ∗r)
- void **sort** (int param)
- void **sort** ()

Protected Member Functions

- void resize ()

  Resizes the array.

Protected Attributes

- Roots ∗ **data**
- int **size**
- int **MAX**

Friends

- ostream & operator<< (ostream &strm, List &ls)

  Represents the list.
- ostream & operator<< (ostream &strm, List ∗ls)

  Represents the list.
- bool **operator==** (List &l1, List &l2)
- bool **operator==** (List ∗l1, List ∗l2)

5.4.1 Detailed Description

template<typename Roots>class List< Roots >

Basic extendable list structure, similar to vector.

Author

Kip Price

Date

20 September 2012

Template Parameters

Roots	Any type of data

5.4.2 Member Function Documentation

5.4.2.1 template<typename Roots> void List< Roots >::add (Roots *r*) [inline]

Adds an element.

Places an element at the next available index in the underlying array

Parameters

r	A new piece of data to add to the array

5.4.2.2 template<typename Roots> void List< Roots >::addAll (List< Roots > & *ls*) [inline]

Adds all elements from another list.

Parameters

ls	Another list to add

5.4.2.3 template<typename Roots> void List< Roots >::addAll (List< Roots > ∗ *ls*) `[inline]`

Adds all elements from another list.

Parameters

∗*ls*	A pointer to another list to add

5.4.2.4 template<typename Roots> bool List< Roots >::contains (Roots *r*) `[inline]`

Checks containment of an element.

Goes through every element of the array linearly to check if the given input is equivalent to the current index's contents

Parameters

r	The data to check for

Returns

Whether or not r was found in the array

5.4.2.5 template<typename Roots> int List< Roots >::find (Roots *r*) `[inline]`

Find an element.

Find the index of an element that is equivalent to the input

Parameters

r	The element to match to

Returns

The index at which that element occurs, -1 if it does not occur

5.4.2.6 template<typename Roots> bool List< Roots >::fRemove (Roots *r*) `[inline]`

Removes an element.

Finds the element that matches the input, removes it, and returns whether it was found at all

Parameters

r	The element to remove

Returns

Whether it was removed

5.4.2.7 template<typename Roots> Roots List< Roots >::get (int *i*) `[inline]`

Fetches an element.

Finds and returns the element at index i without changing the array

Parameters

i	The index to fetch

Returns

The item at that index

5.4.2.8 template<typename Roots> int List< Roots >::getSize ( ) `[inline]`

Returns the size of the list.

Returns

The size of the array

5.4.2.9 template<typename Roots> void List< Roots >::insert (Roots *r*, int *i*) `[inline]`

Inserts an element.

Inserts an element at a given index and shifts everything else over

Parameters

r	Element to add
i	The index to add it to

5.4.2.10 template<typename Roots> void List< Roots >::move (int *i*, int *j*) `[inline]`

Moves an element from one index to another.

Useful for insertion sort

Parameters

i	Index of element to move
j	Destination index

5.4.2.11 template<typename Roots> Roots List< Roots >::operator[] (int *val*) `[inline]`

Operator for array indices.

Allows the square brackets used in arrays to work with Lists

Parameters

i	The index to draw from

Returns

The element at that index

5.4.2.12 template< typename Roots > Roots List< Roots >::remove (int *i*) [inline]

Remove an element.

Finds the element at index i, removes it, and returns the item that was contained in it

Parameters

i	The index to remove

Returns

The element at that index

5.4.2.13 template< typename Roots > void List< Roots >::resize ( ) [inline], [protected]

Resizes the array.

Doubles the size of the array whenever the size gets to be larger than the storage available

5.4.3 Friends And Related Function Documentation

5.4.3.1 template< typename Roots > ostream& operator<< (ostream & *strm*, List< Roots > & *ls*) [friend]

Represents the list.

Parameters

strm	An output stream
ls	A list

Returns

The stream with a representation of the list added

5.4.3.2 template< typename Roots > ostream& operator<< (ostream & *strm*, List< Roots > * *ls*) [friend]

Represents the list.

Parameters

strm	An output stream
∗*ls*	A pointer to a list

Returns

The stream with a representation of the list added

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

- Classes/dataStruct.h

5.5 Markov< T > Class Template Reference

A class that keeps track of Markov probabilities based on morphemes.

```
#include <markov.h>
```

Public Member Functions

- Markov ()

    Default constructor.
- Markov (int k)

    Constructor with option for setting the length.
- ∼Markov ()

    Destructor.
- void addPre (T ∗pre, T s)

    Adds an element.
- void addPre (List< T > ∗pre, T s)

    Adds an element.
- void addSuff (T ∗suff, T s)

    Adds an element.
- void addSuff (List< T > ∗suff, T s)

    Adds an element.
- T lPredict (T ∗pre)

    Generates a probable next value given a certain prefix.
- T lPredict (List< T > ∗pre)

    Generates a probable next value given a certain prefix.
- T rPredict (T ∗suff)

    Gnerates a probable previous value given a certain suffix.
- T rPredict (List< T > ∗suff)

    Gnerates a probable previous value given a certain suffix.
- T sidePredict (List< T > ∗aff, int which)

    Generates a probable value either forwards or backwards.
- T predict (T ∗pre, T ∗suff)

    Generates a probable value for both forwards and backwards.
- List< Counter< T > > ∗ lProbable (T ∗pre)

    Generates all probable elements.
- List< Counter< T > > ∗ lProbable (List< T > ∗ls)

    Generates all probable elements.
- List< Counter< T > > ∗ rProbable (T ∗suff)

    Generates all probable elements.
- List< Counter > ∗ rProbable (List< T > ∗ls)

    Generates all probable elements.
- List< Counter< T > > ∗ sideProbable (List< T > ∗ls, int which)

    Generates all probable elements.
- List< Counter< T > > ∗ probable (T ∗pre, t ∗suff)

    Generates all probable elements.
- float lGetProbability (T ∗pre, T n)

    Returns the probability of a certain element occurring after a certain prefix.
- float lGetProbability (List< T > ∗ls, T n)

    Returns the probability of a certain element occurring after a certain prefix.
- float rGetProbability (T ∗suff, T n)

Returns the probability of a certain element occurring before a certain suffix.

- float rGetProbability (List< T > ∗ls, T n)

  Returns the probability of a certain element occurring before a certain suffix.
- float getSideProbability (List< T > ∗aff, T n, int which)

  Returns the probability of a certain element occurring in concurrence with a certain affix.
- float getProbability (T ∗pre, T ∗suff, T n)

  Returns the probability of a certain element occurring between two affixes.

Protected Member Functions

- void equalize ()

Protected Attributes

- HashMap< List< T > ∗, T > ∗ **hash_l**
- HashMap< List< T > ∗, T > ∗ **hash_r**
- int **length**

5.5.1 Detailed Description

template<typename T>class Markov< T >

A class that keeps track of Markov probabilities based on morphemes.

Keeps track of a list of morphemes that could come before the current morpheme match

Author

Kip Price

Date

20 June 2013

Version

0.1

Template Parameters

T	The anytype of data we are keeping track of

5.5.2 Constructor & Destructor Documentation

5.5.2.1 template<typename T> Markov< T >::Markov (int *k*) `[inline]`

Constructor with option for setting the length.

Parameters

k	The length of the arrays we will be looking at

5.5.3 Member Function Documentation

5.5.3.1 template<typename T> void Markov< T >::addPre (T * *pre*, T *s*) [inline]

Adds an element.

Adds to the map that is looking before

Parameters

pre	The prefix to consider
s	The element that will be our value

5.5.3.2 template<typename T> void Markov< T >::addPre (List< T > * *pre*, T *s*) [inline]

Adds an element.

Adds to the map that is looing before

Parameters

pre	The prefix to consider in list form
s	The element that will be our value

5.5.3.3 template<typename T> void Markov< T >::addSuff (T * *suff*, T *s*) [inline]

Adds an element.

Adds to the map that is looking after

Parameters

suff	The suffix to consider
s	The element that will be our value

5.5.3.4 template<typename T> void Markov< T >::addSuff (List< T > * *suff*, T *s*) [inline]

Adds an element.

Adds to the map that is looking after

Parameters

suff	The suffix in list form
s	The element that will be our value

5.5.3.5 template<typename T> void Markov< T >::equalize ( ) [protected]

The length of any given array stored in the map

5.5.3.6 template<typename T> float Markov< T >::getProbability (T * *pre*, T * *suff*, T *n*) [inline]

Returns the probability of a certain element occurring between two affixes.

Parameters

pre	The prefix to consider
suff	The suffix to consider
n	The element to consider

Returns

The percentage that this occurs

5.5.3.7 template<typename T> float Markov< T >::getSideProbability (List< T > * *aff*, T *n*, int *which*) [inline]

Returns the probability of a certain element occurring in concurrence with a certain affix.

Parameters

aff	The affix to consider
n	The element to look at
which	The type of affix we are considering

Returns

The percentage of time that this occurs

5.5.3.8 template<typename T> float Markov< T >::lGetProbability (T * *pre*, T *n*) [inline]

Returns the probability of a certain element occurring after a certain prefix.

Parameters

pre	The prefix to consider
n	The element to look at

Returns

The percentage of the time that this occurs

5.5.3.9 template<typename T> float Markov< T >::lGetProbability (List< T > * *ls*, T *n*) [inline]

Returns the probability of a certain element occurring after a certain prefix.

Parameters

ls	The prefix to consider in list form
n	The element to look at

Returns

The percentage of the time that this occurs

5.5.3.10 template<typename T> T Markov< T >::lPredict (T * *pre*) [inline]

Generates a probable next value given a certain prefix.

Looks at all previously seen next values and chooses one probabilistically

Parameters

pre	The prefix to consider

Returns

A probable next value

5.5.3.11 template<typename T> T Markov< T >::lPredict (List< T > * *pre*) `[inline]`

Generates a probable next value given a certain prefix.

Looks at all previously seen next values and chooses one probabilistically

Parameters

pre	The prefix to consider in list form

Returns

A probable next value

5.5.3.12 template<typename T> List<Counter<T> >* Markov< T >::lProbable (T * *pre*) `[inline]`

Generates all probable elements.

Uses the given prefix to find all elements that ever appeared after it

Parameters

pre	The prefix to consider

Returns

A list of all previously seen elements

5.5.3.13 template<typename T> List<Counter<T> >* Markov< T >::lProbable (List< T > * *ls*) `[inline]`

Generates all probable elements.

Uses the given prefix to find all elements that ever appeared after it

Parameters

ls	The prefix to consider in list form

Returns

A list of all previously seen elements

5.5.3.14 template<typename T> T Markov< T >::predict (T * *pre*, T * *suff*) `[inline]`

Generates a probable value for both forwards and backwards.

Generates a list for both probable endings for prefixes and probable beginnings for suffixes, then finds the one that is most likely to occur and occurs in both

Parameters

pre	The prefix to consider
suff	The suffix to consider

Returns

A probable value for both the prefix and the suffix given

5.5.3.15 template<typename T> List<Counter<T> >* Markov< T >::probable (T * *pre*, t * *suff*) [inline]

Generates all probable elements.

Takes in a prefix and suffix and returns a list of all previously seen elements in between them

Parameters

pre	The prefix to consider
suff	The suffix to consider

Returns

A list of all probable elements seen previously, sorted by both counts

5.5.3.16 template<typename T> float Markov< T >::rGetProbability (T * *suff*, T *n*) [inline]

Returns the probability of a certain element occurring before a certain suffix.

Parameters

suff	The suffix to consider
n	The element to look at

Returns

The percentage of the time that this occurs

5.5.3.17 template<typename T> float Markov< T >::rGetProbability (List< T > * *ls*, T *n*) [inline]

Returns the probability of a certain element occurring before a certain suffix.

Parameters

ls	The suffix to consider in list form
n	The element to look at

Returns

The percentage of the time that this occurs

5.5.3.18 template<typename T> T Markov< T >::rPredict (T * *suff*) [inline]

Gnerates a probable previous value given a certain suffix.

Looks at all previously seen previous values and chooses one probabilistically

Parameters

suff	The suffix to consider

Returns

A probable previous value

5.5.3.19 template<typename T> T Markov< T >::rPredict (List< T > * *suff*) [inline]

Gnerates a probable previous value given a certain suffix.

Looks at all previously seen previous values and chooses one probabilistically

Parameters

suff	The suffix to consider in list form

Returns

A probable previous value

5.5.3.20 template<typename T> List<Counter<T> >* Markov< T >::rProbable (T * *suff*) [inline]

Generates all probable elements.

Uses the given suffix to find all elements that ever appeared before it

Parameters

suff	The suffix to consider

Returns

A list of all previously seen elements

5.5.3.21 template<typename T> List<Counter>* Markov< T >::rProbable (List< T > * *ls*) [inline]

Generates all probable elements.

Uses the given suffix to find all elements that ever appeared before it

Parameters

ls	The suffix to consider in list form

Returns

A list of all previously seen elements

5.5.3.22 template<typename T> T Markov< T >::sidePredict (List< T > ∗ *aff*, int *which*) `[inline]`

Generates a probable value either forwards or backwards.

Looks at all previously seen values and chooses one probabilistically

Parameters

aff	The affix to consider in list form
which	Whether we are going forwards or backwards

Returns

A probable value for this type of affix

5.5.3.23 template<typename T> List<Counter<T> >∗ Markov< T >::sideProbable (List< T > ∗ *ls*, int *which*) `[inline]`

Generates all probable elements.

Takes in an affix and finds all of the previously seen elements before or after it

Parameters

aff	The affix to consider
which	The type of affix this is concerning

Returns

A list of all probable elements seen previously

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

- Classes/markov.h

5.6 N< Boots > Class Template Reference

A node for a linked list queue.

`#include <dataStruct.h>`

Public Member Functions

- N ()
  Constructor for N node.
- N (Boots b)
  Constructor for N node.

Public Attributes

- N< Boots > ∗ **next**
- N< Boots > ∗ prev
- Boots data

5.6.1 Detailed Description

template<typename Boots>class N< Boots >

A node for a linked list queue.

Author

Kip Price

Date

20 September 2012

Template Parameters

Boots	Any type

5.6.2 Constructor & Destructor Documentation

5.6.2.1 template<typename Boots> N< Boots >::N ( ) [inline]

Constructor for N node.

The actual data contained in the node

5.6.2.2 template<typename Boots> N< Boots >::N (Boots *b*) [inline]

Constructor for N node.

Parameters

b	The data held by this node

5.6.3 Member Data Documentation

5.6.3.1 template<typename Boots> Boots N< Boots >::data

The previous element in the linked list

5.6.3.2 template<typename Boots> N<Boots>* N< Boots >::prev

The next element in the linked list

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

- Classes/dataStruct.h

5.7 Node< Zip, Zap > Class Template Reference

A class to store a node of a HashMap.

`#include <hash.h>`

Public Member Functions

- Node (Zip k, Zap v)

    Constructor for a node.
- Node ()

    Default constructor for a node.
- ∼Node ()

    Destructor for a node.
- Zip getKey ()

    Gets the key of the node.
- Zap getVal ()

    Gets the value of the node.
- int getStat ()

    Gets the count of the node.
- void increment ()

    Adds to the count of the node.
- bool operator== (const Node< Zip, Zap > &other) const

    Checks two nodes for equality.

Public Attributes

- Zip **zip**
- Zap zap
- int stat

Friends

- ostream & operator<< (ostream &strm, const Node< Zip, Zap > &n)

    Represents the Node.
- ostream & **operator**<< (ostream &strm, Node< Zip, Zap > ∗n)
- bool **operator==** (Node ∗n1, Node ∗n2)

5.7.1 Detailed Description

template<typename Zip, typename Zap>class Node< Zip, Zap >

A class to store a node of a HashMap.

Author

Kip Price

Date

11 June 2013

Template Parameters

Zip	A key
Zap	A value

5.7.2 Constructor & Destructor Documentation

5.7.2.1 template<typename Zip, typename Zap> Node< Zip, Zap >::Node (Zip *k*, Zap *v*) `[inline]`

Constructor for a node.

The number of times this node has occurred

Parameters

k	The key for this node
v	The value for this node

5.7.3 Member Function Documentation

5.7.3.1 template<typename Zip, typename Zap> Zip Node< Zip, Zap >::getKey ( ) `[inline]`

Gets the key of the node.

Returns

The key of the node

5.7.3.2 template<typename Zip, typename Zap> int Node< Zip, Zap >::getStat ( ) `[inline]`

Gets the count of the node.

Returns

THe count of the node

5.7.3.3 template<typename Zip, typename Zap> Zap Node< Zip, Zap >::getVal ( ) `[inline]`

Gets the value of the node.

Returns

The value of the node

5.7.3.4 template<typename Zip, typename Zap> bool Node< Zip, Zap >::operator== (const Node< Zip, Zap > & *other*) const `[inline]`

Checks two nodes for equality.

Parameters

other	Another Node

Returns

Whether the two nodes are the same

5.7.4 Friends And Related Function Documentation

5.7.4.1 template<typename Zip, typename Zap> ostream& operator<< (ostream & *strm*, const Node< Zip, Zap > & *n*) [friend]

Represents the Node.

Parameters

strm	An ostream to output to
n	A node

Returns

The output stream with the representation of the node added to the end

5.7.5 Member Data Documentation

5.7.5.1 template<typename Zip, typename Zap> int Node< Zip, Zap >::stat

The value of the node

5.7.5.2 template<typename Zip, typename Zap> Zap Node< Zip, Zap >::zap

The key of the node

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

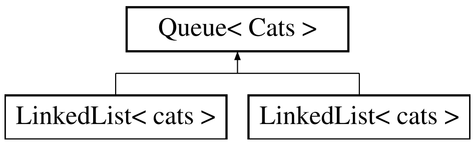
- Classes/hash.h

5.8 Queue< Cats > Class Template Reference

A linked list queue using N nodes.

`#include <dataStruct.h>`

Inheritance diagram for Queue< Cats >:



Public Member Functions

- Queue ()

    Constructor for Queue.
- ∼Queue ()

    Destructor for Queue.
- bool enqueue (Cats c)

    Adds an element.
- bool contains (Cats c)

    Checks if an element is contained.
- Cats dequeue ()

Removes an element.
- Cats peek ()

  Looks at the next element.
- int getSize ()

  Gets the size of the queue.

Friends

- ostream & operator<< (ostream &strm, Queue &q)

  Represents the queue.

5.8.1 Detailed Description

template<typename Cats>class Queue< Cats >

A linked list queue using N nodes.

Author

Kip Price

Date

20 September 2012

Template Parameters

Cats	Any type

5.8.2 Constructor & Destructor Documentation

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

Constructor for Queue.

The size of the list, not including the first or the last

Initializes head and tail to be empty nodes that are never touched

5.8.3 Member Function Documentation

5.8.3.1 template<typename Cats > bool Queue< Cats >::contains (Cats *c*) `[inline]`

Checks if an element is contained.

Goes through every element linearly to check for equivalence

Parameters

c	The data to check for containment

Returns

Whether the element is already present in the queue

5.8.3.2 template<typename Cats > Cats Queue< Cats >::dequeue ( ) [inline]

Removes an element.

Removes an element from the front, and links the head to the node that was next.

Returns

The data that was held by the element at the front of the queue

5.8.3.3 template<typename Cats > bool Queue< Cats >::enqueue (Cats *c*) [inline]

Adds an element.

Adds an element right before the tail of the list

Parameters

c	The data to be added

Returns

Whether the data was added (if it was already contained, it will not add)

5.8.3.4 template<typename Cats > int Queue< Cats >::getSize ( ) [inline]

Gets the size of the queue.

Returns

The size of the queue

5.8.3.5 template<typename Cats > Cats Queue< Cats >::peek ( ) [inline]

Looks at the next element.

Gets the information of the next-to-be-removed element without removing it

Returns

The data of the next element

5.8.4 Friends And Related Function Documentation

5.8.4.1 template<typename Cats > ostream& operator<< (ostream & *strm,* Queue< Cats > & *q*) [friend]

Represents the queue.

Lists all of the elements if the queue in FIFO order

Parameters

strm	An output stream
q	The queue to be represented

Returns

An output stream with the representation of the queue added

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

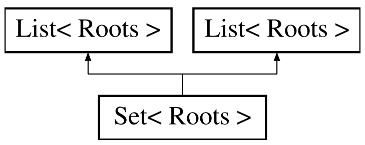
- Classes/dataStruct.h

5.9 Set< Roots > Class Template Reference

A non-repetitive list.

```
#include <dataStruct.h>
```

Inheritance diagram for Set< Roots >:



Public Member Functions

- bool add (Roots r)

  Adds an element.

Additional Inherited Members

5.9.1 Detailed Description

template<typename Roots>class Set< Roots >

A non-repetitive list.

Author

Kip Price

Date

20 September 2012

Template Parameters

Roots	Any type

5.9.2 Member Function Documentation

5.9.2.1 template<typename Roots> bool Set< Roots >::add (Roots *r*) `[inline]`

Adds an element.

The only different function than a list, only adds if it isn't already present

Parameters

r	The data to add

Returns

Whether the data was successfully added

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

- Classes/dataStruct.h

5.10 Stack$<$ Shoots $>$ Class Template Reference

A stack class built from a list.

`#include <dataStruct.h>`

Public Member Functions

- Stack ()

  Constructor for the stack.
- ~Stack ()

  Destructor for the stack.
- bool push (Shoots s)

  Adds an element.
- Shoots pop ()

  Removes an element.
- Shoots peek ()

  Looks at the next element.

5.10.1 Detailed Description

template$<$typename Shoots$>$class Stack$<$ Shoots $>$

A stack class built from a list.

Author

Kip Price

Date

20 September 2012

Template Parameters

Shoots	Any type

5.10.2 Constructor & Destructor Documentation

5.10.2.1 template<typename Shoots > Stack< Shoots >::Stack ( ) [inline]

Constructor for the stack.

The size of the stack

5.10.3 Member Function Documentation

5.10.3.1 template<typename Shoots > Shoots Stack< Shoots >::peek ( ) [inline]

Looks at the next element.

Gets the data of the last element added to the stack without removing it

Returns

The data of the last element of the stack

5.10.3.2 template<typename Shoots > Shoots Stack< Shoots >::pop ( ) [inline]

Removes an element.

Takes the last element added to the stack and removes it

Returns

The last element of the stack

5.10.3.3 template<typename Shoots > bool Stack< Shoots >::push (Shoots *s*) [inline]

Adds an element.

Puts an element in the last index available to the stack

Parameters

s	The data to add

Returns

Whether the data was added (if it wasn't, it already existed in the stack)

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

- Classes/dataStruct.h

Index