

CS 251 (Software Systems Lab): Doxygen

Generated by Doxygen 1.8.17

1 Class Index	1
1.1 Class List	1
2 File Index	3
2.1 File List	3
3 Class Documentation	5
3.1 BinarySearchTree Class Reference	5
3.1.1 Detailed Description	6
3.1.2 Member Enumeration Documentation	6
3.1.2.1 order	6
3.1.3 Constructor & Destructor Documentation	6
3.1.3.1 BinarySearchTree()	6
3.1.4 Member Function Documentation	6
3.1.4.1 height()	6
3.1.4.2 insert()	7
3.1.4.3 traverse()	7
3.2 BSTNode Class Reference	7
3.2.1 Detailed Description	8
3.2.2 Constructor & Destructor Documentation	8
3.2.2.1 BSTNode()	8
3.3 DoublyLinkedList Class Reference	9
3.3.1 Detailed Description	9
3.3.2 Constructor & Destructor Documentation	10
3.3.2.1 DoublyLinkedList()	10
3.3.3 Member Function Documentation	10
3.3.3.1 insert()	10
3.3.3.2 printer()	10
3.3.3.3 reverse()	10
3.4 DoublyLinkedListNode Class Reference	11
3.4.1 Detailed Description	11
3.4.2 Constructor & Destructor Documentation	11
3.4.2.1 DoublyLinkedListNode() [1/2]	12
3.4.2.2 DoublyLinkedListNode() [2/2]	12
3.5 Heap Class Reference	12
3.5.1 Detailed Description	12
3.5.2 Member Function Documentation	13
3.5.2.1 left()	13
3.5.2.2 min()	13
3.5.2.3 parent()	13
3.5.2.4 right()	14
3.6 SinglyLinkedList Class Reference	14
3.6.1 Detailed Description	15

3.6.2 Constructor & Destructor Documentation	15
3.6.2.1 SinglyLinkedList()	15
3.6.3 Member Function Documentation	15
3.6.3.1 deleteVal()	16
3.6.3.2 find()	16
3.6.3.3 insert()	16
3.6.3.4 printer()	17
3.6.3.5 reverse()	17
3.7 SinglyLinkedListNode Class Reference	17
3.7.1 Detailed Description	18
3.7.2 Constructor & Destructor Documentation	18
3.7.2.1 SinglyLinkedListNode() [1/2]	18
3.7.2.2 SinglyLinkedListNode() [2/2]	18
3.8 Trie Class Reference	18
3.8.1 Detailed Description	19
3.8.2 Constructor & Destructor Documentation	19
3.8.2.1 Trie()	19
3.8.3 Member Function Documentation	19
3.8.3.1 checkPrefix()	19
3.8.3.2 countPrefix()	20
3.8.3.3 find()	20
3.8.3.4 insert()	21
4 File Documentation	23
4.1 DSA.cpp File Reference	23
4.1.1 Detailed Description	24
4.1.2 Function Documentation	24
4.1.2.1 merge()	24
4.1.2.2 operator<<() [1/3]	25
4.1.2.3 operator<<() [2/3]	25
4.1.2.4 operator<<() [3/3]	25
Index	27

Chapter 1

Class Index

1.1 Class List

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

BinarySearchTree	The class for a Binary Search Tree	5
BSTNode	The class for each node in a Binary Search Tree	7
DoublyLinkedList	The class for a doubly linked list	9
DoublyLinkedListNode	The class for each node in a doubly linked list	11
Heap	This class is used to represent a min heap	12
SinglyLinkedList	The class for a singly linked list	14
SinglyLinkedListNode	The class for each node in a singly linked list	17
Trie	The class for a Suffix-Trie	18

Chapter 2

File Index

2.1 File List

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

DSA.cpp	Some implementations of common Data Structures and Algorithms	23
-------------------------	---	--------------------

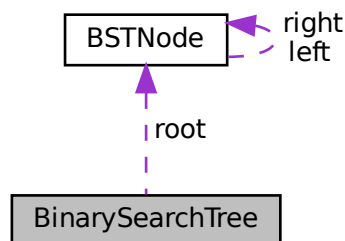
Chapter 3

Class Documentation

3.1 BinarySearchTree Class Reference

The class for a Binary Search Tree.

Collaboration diagram for BinarySearchTree:



Public Types

- enum **order** { **PRE**, **IN**, **POST** }

An enumeration of the possible orders in which the tree may be traversed.

Public Member Functions

- BinarySearchTree** ()
Construct a new Binary Search Tree object with no root.
- void **insert** (long long int val)
Inserts data in the Binary Search Tree.
- void **traverse** (BSTNode *T, order tt)
traverses a Binary Search Tree in a given manner from a given node
- long long int **height** (BSTNode *T)
Returns the height of a Node in a Binary Search Tree.

Public Attributes

- `BSTNode * root`

A pointer to the root of the Binary Search Tree.

3.1.1 Detailed Description

The class for a Binary Search Tree.

3.1.2 Member Enumeration Documentation

3.1.2.1 order

enum `BinarySearchTree::order`

An enumeration of the possible orders in which the tree may be traversed.

Enumerator

PRE	Preorder traversal
IN	Inorder traversal
POST	Postorder traversal

3.1.3 Constructor & Destructor Documentation

3.1.3.1 BinarySearchTree()

```
BinarySearchTree::BinarySearchTree ( ) [inline]
```

Construct a new Binary Search Tree object with no root.

3.1.4 Member Function Documentation

3.1.4.1 height()

```
long long int BinarySearchTree::height (
    BSTNode * T ) [inline]
```

Returns the height of a Node in a Binary Search Tree.

Parameters

<i>T</i>	the Node whose height is to be found
----------	--------------------------------------

Returns

the height

3.1.4.2 insert()

```
void BinarySearchTree::insert (
    long long int val ) [inline]
```

Inserts data in the Binary Search Tree.

Parameters

<i>val</i>	Data to be inserted in the Binary Search Tree
------------	---

3.1.4.3 traverse()

```
void BinarySearchTree::traverse (
    BSTNode * T,
    order tt ) [inline]
```

traverses a Binary Search Tree in a given manner from a given node

Parameters

<i>T</i>	Node from which we start our traversal
<i>tt</i>	Manner in which we traverse the Binary Search Tree (Preorder/Inorder/Postorder)

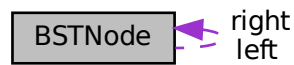
The documentation for this class was generated from the following file:

- [DSA.cpp](#)

3.2 BSTNode Class Reference

The class for each node in a Binary Search Tree.

Collaboration diagram for BSTNode:



Public Member Functions

- [BSTNode](#) (long long int val)
Construct a new [BSTNode](#) object with no left or right children.

Public Attributes

- long long int [info](#)
Data stored in the node.
- long long int [level](#)
Level of the node (Distance from the root)
- [BSTNode](#) * [left](#)
Pointer to the left child of the node.
- [BSTNode](#) * [right](#)
Pointer to the right child of the node.

3.2.1 Detailed Description

The class for each node in a Binary Search Tree.

3.2.2 Constructor & Destructor Documentation

3.2.2.1 BSTNode()

```
BSTNode::BSTNode (
    long long int val ) [inline]
```

Construct a new [BSTNode](#) object with no left or right children.

Parameters

<i>val</i>	Data to be stored in the node
------------	-------------------------------

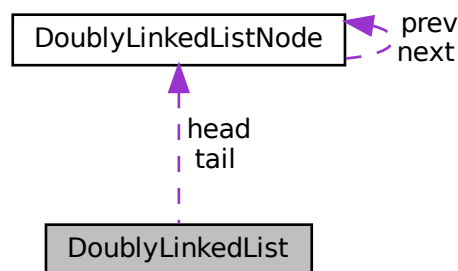
The documentation for this class was generated from the following file:

- [DSA.cpp](#)

3.3 DoublyLinkedList Class Reference

The class for a doubly linked list.

Collaboration diagram for DoublyLinkedList:



Public Member Functions

- [DoublyLinkedList](#) ()
Construct a new Doubly Linked List object with the head and tail pointers initialized to null.
- void [insert](#) (long long int data)
Inserts the given data at the tail of the Doubly Linked List.
- void [printer](#) (string sep=", ")
Prints out the entire Doubly Linked List.
- void [reverse](#) ()
Reverses the list.

Public Attributes

- [DoublyLinkedListNode](#) * [head](#)
A pointer to the head of the doubly linked list.
- [DoublyLinkedListNode](#) * [tail](#)
A pointer to the tail of the doubly linked list.

3.3.1 Detailed Description

The class for a doubly linked list.

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DoublyLinkedList()

```
DoublyLinkedList::DoublyLinkedList ( ) [inline]
```

Construct a new Doubly Linked List object with the head and tail pointers initialized to null.

3.3.3 Member Function Documentation

3.3.3.1 insert()

```
void DoublyLinkedList::insert (
    long long int data ) [inline]
```

Inserts the given data at the tail of the Doubly Linked List.

Parameters

<i>data</i>	The data to be inserted
-------------	-------------------------

3.3.3.2 printer()

```
void DoublyLinkedList::printer (
    string sep = ", " ) [inline]
```

Prints out the entire Doubly Linked List.

Parameters

<i>sep</i>	An optional parameter that denotes the separator of the values. By default it is ", "
------------	---

3.3.3.3 reverse()

```
void DoublyLinkedList::reverse ( ) [inline]
```

Reverses the list.

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

- [DSA.cpp](#)

3.4 DoublyLinkedListNode Class Reference

The class for each node in a doubly linked list.

Collaboration diagram for DoublyLinkedListNode:



Public Member Functions

- [DoublyLinkedListNode](#) ()
Construct a new Doubly Linked List Node object, with data set to -1 and the pointers to the previous and next nodes set to null.
- [DoublyLinkedListNode](#) (long long int val)
Construct a new Doubly Linked List Node object with pointers to the previous and next nodes set to null.

Public Attributes

- long long int [data](#)
An integer describing the data stored in the node.
- [DoublyLinkedListNode](#) * [next](#)
A pointer to the next node in the Doubly Linked List.
- [DoublyLinkedListNode](#) * [prev](#)
A pointer to the previous node in the Doubly Linked List.

3.4.1 Detailed Description

The class for each node in a doubly linked list.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 DoublyLinkedListNode() [1/2]

```
DoublyLinkedListNode::DoublyLinkedListNode ( ) [inline]
```

Construct a new Doubly Linked List Node object, with data set to -1 and the pointers to the previous and next nodes set to null.

3.4.2.2 DoublyLinkedListNode() [2/2]

```
DoublyLinkedListNode::DoublyLinkedListNode (
    long long int val ) [inline]
```

Construct a new Doubly Linked List Node object with pointers to the previous and next nodes set to null.

Parameters

<i>val</i>	Data to be stored in the node
------------	-------------------------------

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

- [DSA.cpp](#)

3.5 Heap Class Reference

This class is used to represent a min heap.

Public Member Functions

- int [parent](#) (int i)
Returns the index of the parent of the element with the given index.
- int [left](#) (int i)
Returns the index of the left child of the element with the given index.
- int [right](#) (int i)
Returns the index of the right child of the element with the given index.
- long long int [min](#) ()
Returns the smallest element in the heap.

Public Attributes

- vector< long long int > [arr](#)
Dynamic Array representing the heap.

3.5.1 Detailed Description

This class is used to represent a min heap.

3.5.2 Member Function Documentation

3.5.2.1 left()

```
int Heap::left (
    int i ) [inline]
```

Returns the index of the left child of the element with the given index.

Parameters

<i>i</i>	The index of the element
----------	--------------------------

Returns

index

-1 if the input index is out of bounds or the element corresponding to it has no left child

3.5.2.2 min()

```
long long int Heap::min ( ) [inline]
```

Returns the smallest element in the heap.

Returns

Returns the smallest element in the heap

-1 if the heap is empty

3.5.2.3 parent()

```
int Heap::parent (
    int i ) [inline]
```

Returns the index of the parent of the element with the given index.

Parameters

<i>i</i>	The index of the element
----------	--------------------------

Returns

index

-1 if the input index is out of bounds or the element corresponding to it has no parent

3.5.2.4 right()

```
int Heap::right (
    int i ) [inline]
```

Returns the index of the right child of the element with the given index.

Parameters

<i>i</i>	The index of the element
----------	--------------------------

Returns

index

-1 if the input index is out of bounds or the element corresponding to it has no right child

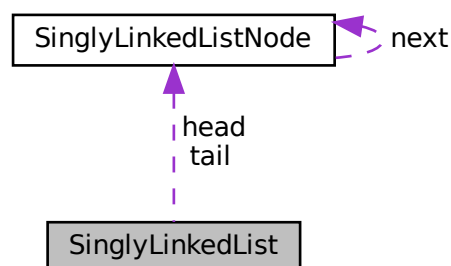
The documentation for this class was generated from the following file:

- [DSA.cpp](#)

3.6 SinglyLinkedList Class Reference

The class for a singly linked list.

Collaboration diagram for SinglyLinkedList:



Public Member Functions

- [SinglyLinkedList](#) ()
Constructs a new Singly Linked List object with head and tail pointers both set to null.
- void [insert](#) (long long int data)
Inserts data into a linked list at the end.
- [SinglyLinkedListNode](#) * [find](#) (long long int data)
Returns a pointer to first node containing the data.
- bool [deleteVal](#) (long long int data)
Deletes a given value from a linked list.
- void [printer](#) (string sep=", ")
Prints out the entire singly linked list.
- void [reverse](#) ()
Reverses our list.

Public Attributes

- [SinglyLinkedListNode](#) * [head](#)
Pointer to the head node of the list (a public variable)
- [SinglyLinkedListNode](#) * [tail](#)
Pointer to the tail node of the list (a public variable)

3.6.1 Detailed Description

The class for a singly linked list.

3.6.2 Constructor & Destructor Documentation

3.6.2.1 SinglyLinkedList()

```
SinglyLinkedList::SinglyLinkedList ( ) [inline]
```

Constructs a new Singly Linked List object with head and tail pointers both set to null.

Parameters

<i>None</i>	
-------------	--

3.6.3 Member Function Documentation

3.6.3.1 deleteVal()

```
bool SinglyLinkedList::deleteVal (
    long long int data ) [inline]
```

Deletes a given value from a linked list.

Parameters

<i>data</i>	(value to be deleted)
-------------	-----------------------

Returns

true (if the data was successfully deleted)
false (if the data was not present in the list)

3.6.3.2 find()

```
SinglyLinkedListNode* SinglyLinkedList::find (
    long long int data ) [inline]
```

Returns a pointer to first node containing the data.

Parameters

<i>data</i>	
-------------	--

Returns

SinglyLinkedListNode*
tail (if the data occurs first at the tail or if it is not present at all)

3.6.3.3 insert()

```
void SinglyLinkedList::insert (
    long long int data ) [inline]
```

Inserts data into a linked list at the end.

Parameters

<i>data</i>	(The inserted data)
-------------	---------------------

Returns

void

3.6.3.4 printer()

```
void SinglyLinkedList::printer (
    string sep = ", " ) [inline]
```

Prints out the entire singly linked list.

Parameters

<i>sep</i>	An optional parameter that denotes the separator of the values. By default it is ", "
------------	---

3.6.3.5 reverse()

```
void SinglyLinkedList::reverse ( ) [inline]
```

Reverses our list.

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

- [DSA.cpp](#)

3.7 SinglyLinkedListNode Class Reference

The class for each node in a singly linked list.

Collaboration diagram for SinglyLinkedListNode:



Public Member Functions

- [SinglyLinkedListNode](#) ()
Construct a new Singly Linked List Node object with next as null and data as -1.
- [SinglyLinkedListNode](#) (long long int val)
Construct a new Singly Linked List Node object with next as null and data as -1.

Public Attributes

- long long int [data](#)
A large integer to store data. This variable is public.
- [SinglyLinkedListNode](#) * [next](#)
A pointer to the next node. This variable is public.

3.7.1 Detailed Description

The class for each node in a singly linked list.

3.7.2 Constructor & Destructor Documentation

3.7.2.1 [SinglyLinkedListNode\(\)](#) [1/2]

```
SinglyLinkedListNode::SinglyLinkedListNode ( ) [inline]
```

Construct a new Singly Linked List Node object with next as null and data as -1.

Parameters

<i>None</i>	
-------------	--

3.7.2.2 [SinglyLinkedListNode\(\)](#) [2/2]

```
SinglyLinkedListNode::SinglyLinkedListNode (
    long long int val ) [inline]
```

Construct a new Singly Linked List Node object with next as null and data as -1.

Parameters

<i>val</i>	(A large integer)
------------	-------------------

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

- [DSA.cpp](#)

3.8 Trie Class Reference

The class for a Suffix-Trie.

Public Member Functions

- [Trie](#) ()
Construct a new [Trie](#) object with no nodes and an empty Dictionary.
- bool [find](#) ([Trie](#) *T, char c)
Checks if a character is present in the dictionary.
- void [insert](#) (string s)
Inserts a string into the Suffix [Trie](#).
- bool [checkPrefix](#) (string s)
Checks if a prefix of a given string is present in our [Trie](#).
- long long int [countPrefix](#) (string s)
Counts the number of prefixes of a given string present in our [Trie](#).

Public Attributes

- long long int [count](#)
Count of nodes in the trie.
- map< char, [Trie](#) * > [nodes](#)
Dictionary of pointers to nodes with characters as keys and pointers to Tries as values.

3.8.1 Detailed Description

The class for a Suffix-Trie.

3.8.2 Constructor & Destructor Documentation

3.8.2.1 [Trie\(\)](#)

```
Trie::Trie ( ) [inline]
```

Construct a new [Trie](#) object with no nodes and an empty Dictionary.

3.8.3 Member Function Documentation

3.8.3.1 [checkPrefix\(\)](#)

```
bool Trie::checkPrefix (
    string s ) [inline]
```

Checks if a prefix of a given string is present in our [Trie](#).

Parameters

<i>s</i>	The given string
----------	------------------

Returns

true if any prefix of the given string is present in our [Trie](#)

false if no prefix of the given string is present in our [Trie](#)

3.8.3.2 countPrefix()

```
long long int Trie::countPrefix (
    string s ) [inline]
```

Counts the number of prefixes of a given string present in our [Trie](#).

Parameters

<i>s</i>	The given string
----------	------------------

Returns

The number of prefixes of this string

3.8.3.3 find()

```
bool Trie::find (
    Trie * T,
    char c ) [inline]
```

Checks if a character is present in the dictionary.

Parameters

<i>T</i>	A pointer to the trie
<i>c</i>	The character whose existence in the dictionary is to be checked

Returns

true If the character is present

false If the character is not present

3.8.3.4 insert()

```
void Trie::insert (  
    string s ) [inline]
```

Inserts a string into the Suffix [Trie](#).

Parameters

s	The string to be inserted
---	---------------------------

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

- [DSA.cpp](#)

Chapter 4

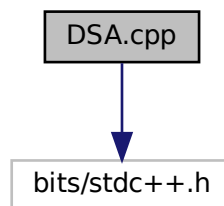
File Documentation

4.1 DSA.cpp File Reference

Some implementations of common Data Structures and Algorithms.

```
#include <bits/stdc++.h>
```

Include dependency graph for DSA.cpp:



Classes

- class [SinglyLinkedListNode](#)
The class for each node in a singly linked list.
- class [SinglyLinkedList](#)
The class for a singly linked list.
- class [DoublyLinkedListNode](#)
The class for each node in a doubly linked list.
- class [DoublyLinkedList](#)
The class for a doubly linked list.
- class [BSTNode](#)
The class for each node in a Binary Search Tree.
- class [BinarySearchTree](#)
The class for a Binary Search Tree.
- class [Trie](#)
The class for a Suffix-Trie.
- class [Heap](#)
This class is used to represent a min heap.

Macros

- `#define ll long long int`
- `#define vi vector<int>`
- `#define vll vector<ll>`

Functions

- `ostream & operator<<` (`ostream &out`, `const SinglyLinkedListNode &node`)
A function that prints out the data in a node object.
- `SinglyLinkedList merge` (`SinglyLinkedList list1`, `SinglyLinkedList list2`)
Merges two sorted singly linked lists and returns the new list.
- `ostream & operator<<` (`ostream &out`, `const DoublyLinkedListNode &node`)
Prints out the data in a node.
- `ostream & operator<<` (`ostream &out`, `const BSTNode &node`)
Prints out the data stored in a node.

4.1.1 Detailed Description

Some implementations of common Data Structures and Algorithms.

Author

CS 251 TAs

Date

21st September 2022

4.1.2 Function Documentation

4.1.2.1 merge()

```
SinglyLinkedList merge (  
    SinglyLinkedList list1,  
    SinglyLinkedList list2 )
```

Merges two sorted singly linked lists and returns the new list.

Parameters

<i>list1</i>	A sorted singly linked list
<i>list2</i>	Another sorted singly linked list

Returns

[SinglyLinkedList](#)

4.1.2.2 operator<<() [1/3]

```
ostream& operator<< (
    ostream & out,
    const BSTNode & node )
```

Prints out the data stored in a node.

Parameters

<i>out</i>	The stream to which data is to be printed
<i>node</i>	The node whose data is to be printed

Returns

ostream&

4.1.2.3 operator<<() [2/3]

```
ostream& operator<< (
    ostream & out,
    const DoublyLinkedListNode & node )
```

Prints out the data in a node.

Parameters

<i>out</i>	Stream in which data is to be printed
<i>node</i>	Node whose data is to be printed

Returns

ostream&

4.1.2.4 operator<<() [3/3]

```
ostream& operator<< (
    ostream & out,
    const SinglyLinkedListNode & node )
```

A function that prints out the data in a node object.

Parameters

<i>out</i>	(The stream)
<i>node</i>	(The node object)

Returns

ostream&

Index

- BinarySearchTree, 5
 - BinarySearchTree, 6
 - height, 6
 - IN, 6
 - insert, 7
 - order, 6
 - POST, 6
 - PRE, 6
 - traverse, 7
- BSTNode, 7
 - BSTNode, 8
- checkPrefix
 - Trie, 19
- countPrefix
 - Trie, 20
- deleteVal
 - SinglyLinkedList, 15
- DoublyLinkedList, 9
 - DoublyLinkedList, 10
 - insert, 10
 - printer, 10
 - reverse, 10
- DoublyLinkedListNode, 11
 - DoublyLinkedListNode, 11, 12
- DSA.cpp, 23
 - merge, 24
 - operator<<, 25
- find
 - SinglyLinkedList, 16
 - Trie, 20
- Heap, 12
 - left, 13
 - min, 13
 - parent, 13
 - right, 14
- height
 - BinarySearchTree, 6
- IN
 - BinarySearchTree, 6
- insert
 - BinarySearchTree, 7
 - DoublyLinkedList, 10
 - SinglyLinkedList, 16
 - Trie, 20
- left
 - Heap, 13
- merge
 - DSA.cpp, 24
- min
 - Heap, 13
- operator<<
 - DSA.cpp, 25
- order
 - BinarySearchTree, 6
- parent
 - Heap, 13
- POST
 - BinarySearchTree, 6
- PRE
 - BinarySearchTree, 6
- printer
 - DoublyLinkedList, 10
 - SinglyLinkedList, 17
- reverse
 - DoublyLinkedList, 10
 - SinglyLinkedList, 17
- right
 - Heap, 14
- SinglyLinkedList, 14
 - deleteVal, 15
 - find, 16
 - insert, 16
 - printer, 17
 - reverse, 17
 - SinglyLinkedList, 15
- SinglyLinkedListNode, 17
 - SinglyLinkedListNode, 18
- traverse
 - BinarySearchTree, 7
- Trie, 18
 - checkPrefix, 19
 - countPrefix, 20
 - find, 20
 - insert, 20
 - Trie, 19