

My Project

Generated by Doxygen 1.9.5

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 Constructor & Destructor Documentation	6
3.1.2.1 BinarySearchTree()	6
3.1.3 Member Function Documentation	6
3.1.3.1 height()	6
3.1.3.2 insert()	7
3.1.3.3 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	9
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
3.4 DoublyLinkedListNode Class Reference	11
3.4.1 Detailed Description	11
3.4.2 Constructor & Destructor Documentation	11
3.4.2.1 DoublyLinkedListNode()	12
3.5 SinglyLinkedList Class Reference	12
3.5.1 Detailed Description	13
3.5.2 Constructor & Destructor Documentation	13
3.5.2.1 SinglyLinkedList()	13
3.5.3 Member Function Documentation	13
3.5.3.1 deleteVal()	13
3.5.3.2 find()	14
3.5.3.3 insert()	14
3.5.3.4 printer()	14
3.6 SinglyLinkedListNode Class Reference	15
3.6.1 Detailed Description	15
3.6.2 Constructor & Destructor Documentation	16
3.6.2.1 SinglyLinkedListNode()	16

3.7 Trie Class Reference	16
3.7.1 Detailed Description	17
3.7.2 Member Function Documentation	17
3.7.2.1 checkPrefix()	17
3.7.2.2 countPrefix()	18
3.7.2.3 find()	18
3.7.2.4 insert()	18
4 File Documentation	21
4.1 DSA.cpp File Reference	21
4.1.1 Detailed Description	21
4.1.2 Function Documentation	22
4.1.2.1 merge()	22
4.1.2.2 operator<<() [1/3]	22
4.1.2.3 operator<<() [2/3]	22
4.1.2.4 operator<<() [3/3]	23
4.2 DSA.h	23
Index	25

Chapter 1

Class Index

1.1 Class List

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

[BinarySearchTree](#)

[BinarySearchTree](#) It has constructor with no parameters [BinarySearchTree \(\)](#)
It has following member functions (i)void [insert\(ll val\)](#) (ii)void traverse (BSTNode* T, order tt) (iii)ll
[height\(BSTNode *T\)](#)
[5](#)

[BSTNode](#)

[BSTNode](#) It contains a constructor [BSTNode\(ll val\)](#)
It has no member functions
[7](#)

[DoublyLinkedList](#)

[DoublyLinkedList](#)
It contains a constructor with no parameters [DoublyLinkedListNode\(\)](#)
It has following member functions (i)void insert (ll data) (ii)void printer (string sep = ", ") (iii)void
reverse ()
[9](#)

[DoublyLinkedListNode](#)

[DoublyLinkedListNode](#)
It contains 2 types of constructor (i)[DoublyLinkedListNode \(\)](#) (ii)[DoublyLinkedListNode \(ll val\)](#)
which creates node with given value by default value is -1 and point next and prev to NULL
It has no member functions
[11](#)

[SinglyLinkedList](#)

[SinglyLinkedList](#) It contains constructor with no parameters [SinglyLinkedList \(\)](#) which when in-
stantiated points head and tail to NULL
It contains member functions (i)void [insert\(ll data\)](#) (ii)[SinglyLinkedListNode* find \(ll data\)](#) (iii)bool
[deleteVal \(ll data\)](#) (iv)void printer (string sep = ", ")
(v)void reverse ()
[12](#)

[SinglyLinkedListNode](#)

[SinglyLinkedListNode](#) It contains 2 types of constructor (i)[SinglyLinkedListNode \(\)](#) (ii)[SinglyLinkedListNode](#)
(ll val) which creates node with given value by default value is -1 and point next to NULL
It has no member functions
[15](#)

Trie

Trie

It has constructor with no parameters [Trie \(\)](#)

It has following member functions (i)bool [find\(Trie* T, char c\)](#) (ii)void [insert\(string s\)](#) (iii)bool [checkPrefix\(string s\)](#) (iv)ll [countPrefix\(string s\)](#)

16

Chapter 2

File Index

2.1 File List

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

DSA.cpp	This file contains classes <code>SingleLinkedListNode</code> , <code>SingleLinkedList</code> , <code>DoublyClassLinkedList</code> , <code>Node</code> , <code>DoublyLinkedList</code> , <code>BSTNode</code> , <code>BinarySearchTree</code> , <code>Trie</code> which contains all required definitions and basic utilities functions of datastructures like Singly Linked List, Doubly Linked List, Binary Search Tree, Suffix <code>Trie</code>	21
DSA.h	??

Chapter 3

Class Documentation

3.1 BinarySearchTree Class Reference

[BinarySearchTree](#) It has constructor with no parameters [BinarySearchTree](#) ()

It has following member functions (i)void [insert](#)(ll val) (ii)void traverse (BSTNode* T, order tt) (iii)ll [height](#)(BSTNode *T)

.

```
#include <DSA.h>
```

Public Types

- enum [order](#) { **PRE** , **IN** , **POST** }
- order*

Public Member Functions

- [BinarySearchTree](#) ()
 - void [insert](#) (ll val)
*It is a member function and having 1 parameter1
First it traverse through BST to find correct position to insert this new node and then change parent of this node to which we should make node a child and also make child of prev node to new node*
 - void [traverse](#) (BSTNode *T, [order](#) tt)
*It is a member function and having 2 parameters
It traverses through Binary search tree according to whether it is pre/IN/POST and then prints all nodes according to it.*
 - ll [height](#) (BSTNode *T)
*It is a member function and having 1 parameter
It traverses through Binary search tree and find height of tree by using recursion*
- .

Public Attributes

- [BSTNode](#) * **root**
*root Datatype BSTNode**

3.1.1 Detailed Description

[BinarySearchTree](#) It has constructor with no parameters [BinarySearchTree](#) ()

It has following member functions (i)void [insert](#)([ll val](#)) (ii)void traverse (BSTNode* T, order tt) (iii)ll [height](#)(BSTNode *T)

.

Parameters

<i>root</i>	Datatype BSTNode*
-------------	-------------------

3.1.2 Constructor & Destructor Documentation

3.1.2.1 BinarySearchTree()

```
BinarySearchTree::BinarySearchTree ( )
```

constructor taking no parameters

Whenever this constructor is called it initialises variable root to NULL

3.1.3 Member Function Documentation

3.1.3.1 height()

```
ll BinarySearchTree::height (
    BSTNode * T )
```

It is a member function and having 1 parameter

It traverses through Binary search tree and find height of tree by using recursion

.

Parameters

<i>in</i>	<i>T</i>	of datatype BSTNode*
-----------	----------	----------------------

Returns

1+max of height of left tree and right tree

3.1.3.2 insert()

```
BinarySearchTree::insert (
    ll val )
```

It is a member function and having 1 parameter1

First it traverse through BST to find correct position to insert this new node and then change parent of this node to which we should make node a child and also make child of prev node to new node

.

Parameters

in	val	of datatype ll
----	-----	----------------

3.1.3.3 traverse()

```
BinarySearchTree::traverse (
    BSTNode * T,
    order tt )
```

It is a member function and having 2 parameters

It traverses through Binary search tree according to whether it is pre/IN/POST and then prints all nodes according to it.

Parameters

in	T	of datatype BSTNode*
in	tt	of datatype order

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

- DSA.h
- [DSA.cpp](#)

3.2 BSTNode Class Reference

[BSTNode](#) It contains a constructor [BSTNode\(ll val\)](#)

It has no member functions

.

```
#include <DSA.h>
```

Public Member Functions

- [BSTNode](#) (ll val)
constructor taking 1 parameter
Whenever this constructor is called it initialises variable info to val and variable level to 0 and variable left to NULL and variable right to NULL

Public Attributes

- **ll info**
info Datatype ll
- **ll level**
level Datatype ll
- **BSTNode * left**
*left Datatype BSTNode**
- **BSTNode * right**
*right Datatype BSTNode**

3.2.1 Detailed Description

[BSTNode](#) It contains a constructor [BSTNode\(ll val\)](#)
It has no member functions

.

Parameters

<i>info</i>	Datatype ll
<i>level</i>	Datatype ll
<i>left</i>	Datatype BSTNode*
<i>right</i>	Datatype BSTNode*

3.2.2 Constructor & Destructor Documentation

3.2.2.1 BSTNode()

```
BSTNode::BSTNode (
    ll val )
```

constructor taking 1 parameter

Whenever this constructor is called it initialises variable info to val and variable level to 0 and variable left to NULL and variable right to NULL

Parameters

in	val	of datatype ll
----	-----	----------------

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

- DSA.h
- [DSA.cpp](#)

3.3 DoublyLinkedList Class Reference

DoublyLinkedList

It contains a constructor with no parameters [DoublyLinkedListNode\(\)](#)

It has following member functions (i)void insert (ll data) (ii)void printer (string sep = " , ") (iii)void reverse ()

.

```
#include <DSA.h>
```

Public Member Functions

- [DoublyLinkedList](#) ()
- void [insert](#) (ll data)

It is a member function and has 1 parameter
First it instantiates class SinglyListNode(data) and equates to node if head is NULL then it points head to node else points next of tail to node and points prev of node to tail then points tail to node
- void [printer](#) (string sep=" , ")

It is a member function and has 1 parameter
It traverse through list and prints all nodes until tail starting from head
- void [reverse](#) ()

It is a member function and has no parameters
It traverse through list and reverse list by just replacing left and right nodes and moving from ends to center.

Public Attributes

- [DoublyLinkedListNode](#) * **head**

*head Datatype DoublyLinkedListNode**
- [DoublyLinkedListNode](#) * **tail**

*tail Datatype DoublyLinkedListNode**

3.3.1 Detailed Description

DoublyLinkedList

It contains a constructor with no parameters [DoublyLinkedListNode\(\)](#)

It has following member functions (i)void insert (ll data) (ii)void printer (string sep = " , ") (iii)void reverse ()

.

Parameters

<i>head</i>	Datatype DoubleLinkedListNode*
<i>tail</i>	Datatype DoubleLinkedListNode*

3.3.2 Constructor & Destructor Documentation

3.3.2.1 DoublyLinkedList()

```
DoublyLinkedList::DoublyLinkedList ( )
```

constructor taking no parameters

Whenever this constructor is called it initialises variable head to NULL and variable tail to NULL

3.3.3 Member Function Documentation

3.3.3.1 insert()

```
void DoublyLinkedList::insert (
    ll data )
```

It is a member function and has 1 parameter

First it instantiates class SinglyListNode(data) and equates to node if head is NULL then it points head to node else points next of tail to node and points prev of node to tail then points tail to node

.

Parameters

in	<i>data</i>	of datatype ll
----	-------------	----------------

3.3.3.2 printer()

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

It is a member function and has 1 parameter

It traverse through list and prints all nodes until tail starting from head

.

Parameters

in	<i>sep</i>	of datatype sep
----	------------	-----------------

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

- DSA.h
- [DSA.cpp](#)

3.4 DoublyLinkedListNode Class Reference

DoublyLinkedListNode

It contains 2 types of constructor (i)[DoublyLinkedListNode](#) () (ii)[DoublyLinkedListNode](#) (ll val) which creates node with given value by default value is -1 and point next and prev to NULL

It has no member functions

.

```
#include <DSA.h>
```

Public Member Functions

- **DoublyLinkedListNode** ()
constructor taking no parameters
Whenever this constructor is called it initialises variable data to -1 and variable next to NULL and variable prev to NULL
- **DoublyLinkedListNode** (ll val)
constructor taking 1 parameter
Whenever this constructor is called it initialises variable data to val and variable next to NULL and variable prev to NULL

Public Attributes

- ll **data**
data Datatype ll
- **DoublyLinkedListNode** * **next**
*next Datatype DoublyLinkedListNode**
- **DoublyLinkedListNode** * **prev**
*prev Datatype DoublyLinkedListNode**

3.4.1 Detailed Description

DoublyLinkedListNode

It contains 2 types of constructor (i)[DoublyLinkedListNode](#) () (ii)[DoublyLinkedListNode](#) (ll val) which creates node with given value by default value is -1 and point next and prev to NULL

It has no member functions

.

Parameters

<i>data</i>	Datatype ll
<i>next</i>	Datatype DoubleLinkedListNode*
<i>prev</i>	Datatype DoubleLinkedListNode*

3.4.2 Constructor & Destructor Documentation

3.4.2.1 DoublyLinkedListNode()

```
DoublyLinkedListNode::DoublyLinkedListNode (
    ll val )
```

constructor taking 1 parameter

Whenever this constructor is called it initialises variable data to val and variable next to NULL and variable prev to NULL

Parameters

in	val	of datatype ll
----	-----	----------------

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

- DSA.h
- [DSA.cpp](#)

3.5 SinglyLinkedList Class Reference

[SinglyLinkedList](#) It contains constructor with no parameters [SinglyLinkedList](#) () which when instantiated points head and tail to NULL

It contains member functions (i)void [insert\(ll data\)](#) (ii)SinglyLinkedListNode* find (ll data) (iii)bool deleteVal (ll data) (iv)void printer (string sep = ", ") (v)void reverse ()

.

```
#include <DSA.h>
```

Public Member Functions

- [SinglyLinkedList](#) ()
- void [insert](#) (ll data)

It is a member function and has 1 parameter
First it instantiates class SinglyListNode(data) and equates to node if head is NULL then it points head to node else points next of tail to node then points tail to node
- [SinglyLinkedListNode](#) * [find](#) (ll data)

It is a member function and has 1 parameter
First it creates two variables ptr and prev of datatypes SinglyLinkedListNode and initialises to head and prev then by using while loop it traverse through list to find node if found then returns*
- bool [deleteVal](#) (ll data)

It is a member function and has 1 parameter
It goes to node which is to be deleted and delete taht node and retirn true if found else return false
- void [printer](#) (string sep=", ")

It is a member function and has 1 parameter
It traverse through list and prints all nodes until tail starting from head
- void [reverse](#) ()

It is a member function and has no parameters
It traverse through list and reverse list by just replacing left and right nodes and moving from ends to center.

Public Attributes

- [SinglyLinkedListNode](#) * **head**
*head Datatype SingleListNode**
- [SinglyLinkedListNode](#) * **tail**
*tail Datatype SingleListNode**

3.5.1 Detailed Description

[SinglyLinkedList](#) It contains constructor with no parameters [SinglyLinkedList](#) () which when instantiated points head and tail to NULL

It contains member functions (i)void [insert\(II data\)](#) (ii)[SinglyLinkedListNode*](#) find (II data) (iii)bool deleteVal (II data) (iv)void printer (string sep = ", ") (v)void reverse ()

.

Parameters

<i>head</i>	Datatype SingleListNode*
<i>tail</i>	Datatype SingleListNode*

3.5.2 Constructor & Destructor Documentation

3.5.2.1 SinglyLinkedList()

```
SinglyLinkedList::SinglyLinkedList ( )
```

constructor taking no parameters

Whenever this constructor is called it initialises variable head to NULL and variable tail to NULL

3.5.3 Member Function Documentation

3.5.3.1 deleteVal()

```
bool SinglyLinkedList::deleteVal (
    II data )
```

It is a member function and has 1 parameter

It goes to node which is to be deleted and delete taht node and retirn true if found else return false

.

Parameters

in	<i>data</i>	of datatype ll
----	-------------	----------------

Returns

which returns true/false according to function

3.5.3.2 find()

```
SinglyLinkedListNode * SinglyLinkedList::find (
    ll data )
```

It is a member function and has 1 parameter

First it creates two variables ptr and prev of datatypes SinglyLinkedListNode* and initialises to head and prev then by using while loop it traverse through list to find node if found then returns

.

Parameters

in	<i>data</i>	of datatype ll
----	-------------	----------------

Returns

which returns prev to function

3.5.3.3 insert()

```
void SinglyLinkedList::insert (
    ll data )
```

It is a member function and has 1 parameter

First it instantiates class SinglyListNode(data) and equates to node if head is NULL then it points head to node else points next of tail to node then points tail to node

.

Parameters

in	<i>data</i>	of datatype ll
----	-------------	----------------

3.5.3.4 printer()

```
void SinglyLinkedList::printer (
```

```
string sep = ", " )
```

It is a member function and has 1 parameter

It traverse through list and prints all nodes until tail starting from head

.

Parameters

in	sep	of datatype string
----	-----	--------------------

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

- DSA.h
- [DSA.cpp](#)

3.6 SinglyLinkedListNode Class Reference

[SinglyLinkedListNode](#) It contains 2 types of constructor (i)[SinglyLinkedListNode](#) () (ii)[SinglyLinkedListNode](#) (ll val) which creates node with given value by default value is -1 and point next to NULL

It has no member functions

.

```
#include <DSA.h>
```

Public Member Functions

- **[SinglyLinkedListNode](#) ()**
constructor taking no parameters
Whenever this constructor is called it initialises variable data to -1 and variable next to NULL
- **[SinglyLinkedListNode](#) (ll val)**
constructor taking 1 parameter
Whenever this constructor is called it initialises variable data to val and variable next to NULL

Public Attributes

- **ll data**
data Datatype ll
- **[SinglyLinkedListNode](#) * next**
*next Datatype SinglyLinkedListNode**

3.6.1 Detailed Description

[SinglyLinkedListNode](#) It contains 2 types of constructor (i)[SinglyLinkedListNode](#) () (ii)[SinglyLinkedListNode](#) (ll val) which creates node with given value by default value is -1 and point next to NULL

It has no member functions

.

Parameters

<i>data</i>	Datatype ll
<i>next</i>	Datatype SinglyLinkedListNode*

3.6.2 Constructor & Destructor Documentation

3.6.2.1 SinglyLinkedListNode()

```
SinglyLinkedListNode::SinglyLinkedListNode (
    ll val )
```

constructor taking 1 parameter

Whenever this constructor is called it initialises variable data to val and variable next to NULL

Parameters

in	val	of datatype ll
----	-----	----------------

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

- DSA.h
- [DSA.cpp](#)

3.7 Trie Class Reference

Trie

It has constructor with no parameters [Trie \(\)](#)

It has following member functions (i)bool [find\(Trie* T, char c\)](#) (ii)void [insert\(string s\)](#) (iii)bool [checkPrefix\(string s\)](#) (iv)ll [countPrefix\(string s\)](#)

.

```
#include <DSA.h>
```

Public Member Functions

- [Trie \(\)](#)
constructor taking no parameters
Whenever this constructor is called it initialises variable count to 0 and variable nodes to map<char,Trie>();*
- bool [find \(Trie *T, char c\)](#)
It is a member function and having 2 parameter
It returns true if c is present in [Trie](#) else return false
- .

- void `insert` (string s)
It is a member function and having 1 parameter
If c is not present in `Trie` then it inserts a new c into `Trie`
 .
- bool `checkPrefix` (string s)
It is a member function and having 1 parameter
It checks whether string s is prefix for any word or not
 .
- ll `countPrefix` (string s)
It is a member function and having 1 parameter
It counts for how many words string s is prefix
 .

Public Attributes

- ll `count`
count Datatype ll
- map< char, `Trie` * > `nodes`
nodes of Datatype map<char, Trie>*

3.7.1 Detailed Description

`Trie`

It has constructor with no parameters `Trie` ()

It has following member functions (i)bool `find(Trie* T, char c)` (ii)void `insert(string s)` (iii)bool `checkPrefix(string s)` (iv)ll `countPrefix(string s)`

.

Parameters

<code>count</code>	Datatype ll
--------------------	-------------

3.7.2 Member Function Documentation

3.7.2.1 `checkPrefix()`

```
bool Trie::checkPrefix (
    string s )
```

It is a member function and having 1 parameter

It checks whether string s is prefix for any word or not

.

Parameters

in	s	of datatype string
----	---	--------------------

3.7.2.2 countPrefix()

```
ll Trie::countPrefix (
    string s )
```

It is a member function and having 1 parameter
It counts for how many words string s is prefix

.

Parameters

in	s	of datatype string
----	---	--------------------

Returns

which returns count of prefix

3.7.2.3 find()

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

It is a member function and having 2 parameter
It returns true if c is present in [Trie](#) else return false

.

Parameters

in	T	of datatype Trie*
in	c	of datatype char

Returns

which returns true/false

3.7.2.4 insert()

```
void Trie::insert (
    string s )
```

It is a member function and having 1 parameter
If c is not present in [Trie](#) then it inserts a new c into [Trie](#)

.

Parameters

in	s	of datatype string
----	---	--------------------

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

- DSA.h
- [DSA.cpp](#)

File Documentation

This file contains classes `SingleLinkedListNode`, `SingleLinkedList`, `DoublyClassLinkedListNode`, `DoublyLinkedList`, `BSTNode`, `BinarySearchTree` which contains all required definitions and basic utilities functions of datastructures like Singly Linked List, Doubly Linked List, Binary Search Tree, Suffix Trie.

Functions

- It is a function and has 2 parameters
It merges two Singlylinkedlists and returns a combined lists

- ### 4.1.1 Detailed Description

This file contains classes `SingleNode`,`SingleLinkedList`,`DoublyClassLinkedListNode`,[DoublyLinkedList](#),`BSTNode`,`BinarySearchTree` which contains all required definitions and basic utilities functions of datastructures like Singly Linked List,Doubly Linked List,Binary Search Tree,Suffix [Trie](#).

Narkedamilli Harika

21/09/2022

4.1.2 Function Documentation

4.1.2.1 merge()

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

It is a function and has 2 parameters

It merges two Singlylinkedlists and returns a combined lists

.

Parameters

in	<i>list1</i>	of datatype SinglyLinkedList
in	<i>list2</i>	of datatype SinglyLinkedList

Returns

which returns a merged [SinglyLinkedList](#)

4.1.2.2 operator<<() [1/3]

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

defines the operator <<,the function takes two parameters

Parameters

in	<i>out</i>	
in	<i>node</i>	

Returns

ostream&

4.1.2.3 operator<<() [2/3]

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

defines the operator <<,the function takes two parameters

Parameters

in	<i>out</i>	
in	<i>node</i>	

Returns

ostream&

4.1.2.4 operator<<() [3/3]

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

defines the operator <<,the function takes two parameters

Parameters

in	<i>out</i>	
in	<i>node</i>	

Returns

ostream&

4.2 DSA.h

```
1
2
3
4
5
6
7
8 #include <bits/stdc++.h>
9 #define ll long long int
10 #define vi vector<int>
11 #define vll vector<ll>
12 using namespace std;
13
14 /* ----- Data Structures ----- */
15
16 // ----- Singly Linked List -----
17
18
19
20
21
22
23
24
25
26
27 class SinglyLinkedListNode
28 {
29     public:
30
31     ll data;
32     SinglyLinkedListNode* next;
33     SinglyLinkedListNode () ;
34     SinglyLinkedListNode (ll val) ;
35 };
36
37 ostream& operator<<(ostream &out, const SinglyLinkedListNode &node);
38
39 class SinglyLinkedList {
40
41     public:
42     SinglyLinkedListNode *head;
43     SinglyLinkedListNode *tail;
44
45     SinglyLinkedList () ;
46
47     void insert (ll data) ;
48
49     SinglyLinkedListNode* find (ll data) ;
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```

101
109         bool deleteVal (ll data) ;
110
117         void printer (string sep = ", " );
118
124         void reverse () ;
125
126 };
135 SinglyLinkedList merge (SinglyLinkedList list1, SinglyLinkedList list2) ;
136
137 // ----- Doubly Linked List -----
147 class DoublyLinkedListNode {
148     public:
152         ll data;
156         DoublyLinkedListNode *next;
160         DoublyLinkedListNode *prev;
161
166         DoublyLinkedListNode () ;
167
173         DoublyLinkedListNode (ll val) ;
174
175 };
176
183 ostream& operator<<(ostream &out, const DoublyLinkedListNode &node);
184
193 class DoublyLinkedList {
194     public:
198         DoublyLinkedListNode *head;
202         DoublyLinkedListNode *tail;
207         DoublyLinkedList () ;
208
216         void insert (ll data) ;
217
224         void printer (string sep = ", " );
225
231         void reverse () ;
232
233 };
234
235 // ----- Binary Search Tree -----
246 class BSTNode {
247     public:
251         ll info;
255         ll level;
259         BSTNode *left;
263         BSTNode *right;
264
270         BSTNode (ll val);
271 };
278 ostream& operator<<(ostream &out, const BSTNode &node) ;
288 class BinarySearchTree {
289     public:
293         BSTNode *root;
297         enum order {PRE, IN, POST};
302         BinarySearchTree () ;
303
310         void insert (ll val) ;
311
319         void traverse (BSTNode* T, order tt) ;
320
328         ll height (BSTNode *T) ;
329 };
330
331 // ----- Suffix Trie -----
339 class Trie {
340     public:
344         ll count;
348         map<char,Trie*> nodes;
349
354         Trie () ;
355
363         bool find(Trie* T, char c) ;
364
371         void insert(string s) ;
372
379         bool checkPrefix(string s) ;
380
388         ll countPrefix(string s) ;
389
390 };

```

Index

- BinarySearchTree, [5](#)
 - BinarySearchTree, [6](#)
 - height, [6](#)
 - insert, [6](#)
 - traverse, [7](#)
- BSTNode, [7](#)
 - BSTNode, [8](#)
- checkPrefix
 - Trie, [17](#)
- countPrefix
 - Trie, [18](#)
- deleteVal
 - SinglyLinkedList, [13](#)
- DoublyLinkedList, [9](#)
 - DoublyLinkedList, [9](#)
 - insert, [10](#)
 - printer, [10](#)
- DoublyLinkedListNode, [11](#)
 - DoublyLinkedListNode, [11](#)
- DSA.cpp, [21](#)
 - merge, [22](#)
 - operator<<, [22](#), [23](#)
- find
 - SinglyLinkedList, [14](#)
 - Trie, [18](#)
- height
 - BinarySearchTree, [6](#)
- insert
 - BinarySearchTree, [6](#)
 - DoublyLinkedList, [10](#)
 - SinglyLinkedList, [14](#)
 - Trie, [18](#)
- merge
 - DSA.cpp, [22](#)
- operator<<
 - DSA.cpp, [22](#), [23](#)
- printer
 - DoublyLinkedList, [10](#)
 - SinglyLinkedList, [14](#)
- SinglyLinkedList, [12](#)
 - deleteVal, [13](#)
 - find, [14](#)
 - insert, [14](#)
 - printer, [14](#)
 - SinglyLinkedList, [13](#)
 - SinglyLinkedListNode, [15](#)
 - SinglyLinkedListNode, [16](#)
- traverse
 - BinarySearchTree, [7](#)
- Trie, [16](#)
 - checkPrefix, [17](#)
 - countPrefix, [18](#)
 - find, [18](#)
 - insert, [18](#)