AVL tree

Generated by Doxygen 1.9.3

Chapter 1

Binary Tree

1.1 Description

This program developed in C++17 allows using an Abstract Data Type such as binary search trees (BST) and binary balanced search trees (AVL), where you can perform basic operations and see the content of the tree by console. command line (CLI), for more information the documentation is in the /doc directory

1.2 Usage

To compile the program, the *make* command is used in the main directory and the executable of the program is located in the /bin/main directory.

1.3 Author

Fabrizzio Daniell Perilli Martín – `alu0101138589@ull.edu.es`

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

File Index

4.1 File List

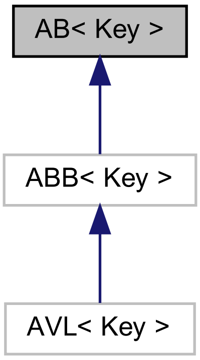
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 AB< Key > Class Template Reference

Inheritance diagram for AB< Key >:



Public Member Functions

- AB ()

  Construct a new AB<Key>::AB object.
- virtual ∼AB ()

  Destroy the AB<Key>::AB object.
- bool isEmpty (NodeB< Key > ∗)

  Check if node is empty or not.
- bool isLeafNode (NodeB< Key > ∗)

  Check if node is leaf.
- int size () const

  This method return the size of tree.
- int height () const

Return the height tree.

- bool isEquilibrium () const

  This method check if the tree is equilibrium.
- virtual void insert (const Key &)=0
- virtual bool search (const Key &) const =0
- void **inorden** () const

Protected Attributes

- NodeB< Key > ∗ **root_**

Friends

- template<class T >
  std::ostream & **operator<<** (std::ostream &, const AB< T > &)

5.1.1 Constructor & Destructor Documentation

5.1.1.1 AB()

```
template<class Key >
AB< Key >::AB
```

Construct a new AB<Key>::AB object.

Template Parameters

Key	

5.1.1.2 ∼AB()

```
template<class Key >
AB< Key >::∼AB  [virtual]
```

Destroy the AB<Key>::AB object.

Template Parameters

Key	

5.1.2 Member Function Documentation

5.1.2.1 height()

```
template<class Key >
int AB< Key >::height  [inline]
```

Return the height tree.

Template Parameters

Key	

Returns

const int

5.1.2.2 insert()

```
template<class Key >
virtual void AB< Key >::insert (
             const Key &  )  [pure virtual]
```

Implemented in AVL< Key >.

5.1.2.3 isEmpty()

```
template<class Key >
bool AB< Key >::isEmpty (
             NodeB< Key > * node )
```

Check if node is empty or not.

Template Parameters

Key	

Parameters

node	

Returns

true

false

5.1.2.4 isEquilibrium()

```
template<class Key >
bool AB< Key >::isEquilibrium  [inline]
```

This method check if the tree is equilibrium.

Template Parameters

Key	

Returns

true

false

5.1.2.5 isLeafNode()

```
template<class Key >
bool AB< Key >::isLeafNode (
            NodeB< Key > * node )
```

Check if node is leaf.

Template Parameters

Key	

Parameters

node	

Returns

true

false

5.1.2.6 search()

```
template<class Key >
virtual bool AB< Key >::search (
             const Key &  ) const  [pure virtual]
```

Implemented in ABB< Key >.

5.1.2.7 size()

```
template<class Key >
int AB< Key >::size  [inline]
```

This method return the size of tree.

Template Parameters

Key	

Returns

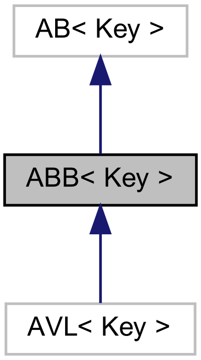
const int

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

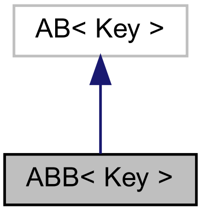
- include/AB.h

5.2 ABB< Key > Class Template Reference

Inheritance diagram for ABB< Key >:



Collaboration diagram for ABB< Key >:



Public Member Functions

- ABB ()

  Construct a new ABB<Key>::ABB object.
- ∼ABB ()

  Destroy the ABB<Key>::ABB object.
- void insert (const Key &) override
- virtual bool search (const Key &) const override

  Search a node in the branch tree.

Additional Inherited Members

5.2.1 Constructor & Destructor Documentation

5.2.1.1 ABB()

```
template<class Key >
ABB< Key >::ABB
```

Construct a new ABB<Key>::ABB object.

Template Parameters

Key	

5.2.1.2 ∼ABB()

```
template<class Key >
```

```
ABB< Key >::~ABB
```

Destroy the ABB<Key>::ABB object.

Template Parameters

Key	

5.2.2 Member Function Documentation

5.2.2.1 insert()

```
template<class Key >
void ABB< Key >::insert (
             const Key & key )  [override], [virtual]
```

Implements AB< Key >.

Reimplemented in AVL< Key >.

5.2.2.2 search()

```
template<class Key >
bool ABB< Key >::search (
             const Key & key ) const  [override], [virtual]
```

Search a node in the branch tree.

Template Parameters

Key	

Parameters

key	

Returns

true

false

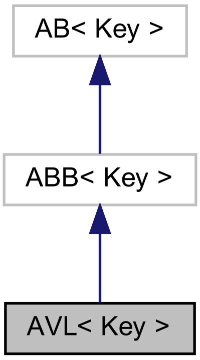
Implements AB< Key >.

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

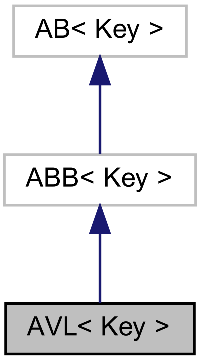
- include/ABB.h

5.3 AVL< Key > Class Template Reference

Inheritance diagram for AVL< Key >:



Collaboration diagram for AVL< Key >:



Public Member Functions

- AVL ()

    Construct a new AVL<Key>::AVL object.
- ∼AVL ()

    Destroy the AVL<Key>::AVL object.
- NodeAVL< Key > ∗ getRootAVL () const

    Return the root of the branch tree.
- NodeAVL< Key > ∗& getRootAVL ()

    Return the root of the branch tree.
- void insert (const Key &k) override

    Insert a node in the branch tree.

Additional Inherited Members

5.3.1 Constructor & Destructor Documentation

5.3.1.1 AVL()

```
template<class Key >
AVL< Key >::AVL
```

Construct a new AVL<Key>::AVL object.

Template Parameters

Key	

5.3.1.2 ∼AVL()

```
template<class Key >
AVL< Key >::∼AVL
```

Destroy the AVL<Key>::AVL object.

Template Parameters

Key	

5.3.2 Member Function Documentation

5.3.2.1 getRootAVL() [1/2]

```
template<class Key >
NodeAVL< Key > *& AVL< Key >::getRootAVL  [inline]
```

Return the root of the branch tree.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.3.2.2 getRootAVL() [2/2]

```
template<class Key >
NodeAVL< Key > * AVL< Key >::getRootAVL  [inline]
```

Return the root of the branch tree.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.3.2.3 insert()

```
template<class Key >
void AVL< Key >::insert (
             const Key & key )  [override], [virtual]
```

Insert a node in the branch tree.

Template Parameters

Key	

Parameters

node	
key	

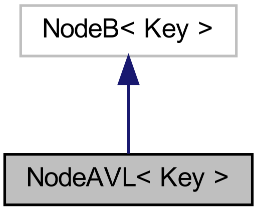
Reimplemented from ABB< Key >.

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

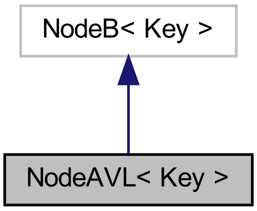
- include/AVL.h

5.4 NodeAVL< Key > Class Template Reference

Inheritance diagram for NodeAVL< Key >:



Collaboration diagram for NodeAVL< Key >:



Public Member Functions

- NodeAVL (const Key &, NodeAVL< Key > ∗=NULL, NodeAVL< Key > ∗=NULL)

  Construct a new Node AVL< Key>:: Node AVL object.
- ∼NodeAVL ()

  Destroy the Node AVL< Key>:: Node AVL object.
- int getBalanceFactor () const

  Return the balance factor of node.
- int & getBalanceFactor ()

  Return the balance factor of node.
- NodeAVL< Key > ∗ getLeftAVL () const

  Return the left node.
- NodeAVL< Key > ∗& getLeftAVL ()

  Return the left node.
- NodeAVL< Key > ∗ getRightAVL () const

  Return the right node.
- NodeAVL< Key > ∗& getRightAVL ()

  Return the right node.

Additional Inherited Members

5.4.1 Constructor & Destructor Documentation

5.4.1.1 NodeAVL()

```
template<class Key >
NodeAVL< Key >::NodeAVL (
             const Key & data,
             NodeAVL< Key > * left = NULL,
             NodeAVL< Key > * right = NULL )
```

Construct a new Node AVL< Key>:: Node AVL object.

Template Parameters

Key	

Parameters

data	
left	
right	

5.4.1.2 ∼NodeAVL()

```
template<class Key >
NodeAVL< Key >::∼NodeAVL
```

Destroy the Node AVL< Key>:: Node AVL object.

Template Parameters

Key	

5.4.2 Member Function Documentation

5.4.2.1 getBalanceFactor() [1/2]

```
template<class Key >
int & NodeAVL< Key >::getBalanceFactor  [inline]
```

Return the balance factor of node.

Template Parameters

Key	

Returns

int

5.4.2.2 getBalanceFactor() [2/2]

```
template<class Key >
int NodeAVL< Key >::getBalanceFactor  [inline]
```

Return the balance factor of node.

Template Parameters

Key	

Returns

int

5.4.2.3 getLeftAVL() [1/2]

```
template<class Key >
NodeAVL< Key > *& NodeAVL< Key >::getLeftAVL  [inline]
```

Return the left node.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.4.2.4 getLeftAVL() [2/2]

```
template<class Key >
NodeAVL< Key > * NodeAVL< Key >::getLeftAVL [inline]
```

Return the left node.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.4.2.5 getRightAVL() [1/2]

```
template<class Key >
NodeAVL< Key > *& NodeAVL< Key >::getRightAVL [inline]
```

Return the right node.

Template Parameters

Key	

Returns

NodeAVL<Key>*

5.4.2.6 getRightAVL() [2/2]

```
template<class Key >
NodeAVL< Key > * NodeAVL< Key >::getRightAVL [inline]
```

Return the right node.

Template Parameters

Key	

Returns

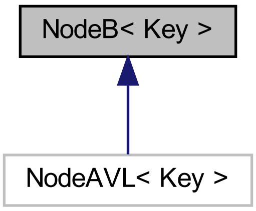
NodeAVL<Key>*

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

- include/NodeAVL.h

5.5 NodeB< Key > Class Template Reference

Inheritance diagram for NodeB< Key >:



Public Member Functions

- NodeB (const Key &, NodeB< Key > ∗=NULL, NodeB< Key > ∗=NULL)

  Construct a new Node B< Key>:: Node B object.
- virtual ∼NodeB ()

  Destroy the Node B< Key>:: Node B object.
- Key getData () const

  Return the dato of node.
- NodeB< Key > ∗ getLeft () const

  Return the left node.
- NodeB< Key > ∗& getLeft ()

  Return the left node.
- NodeB< Key > ∗ getRight () const

  Return the right node.
- NodeB< Key > ∗& getRight ()

  Return the right node.

Protected Attributes

- NodeB< Key > ∗ **left_**
- NodeB< Key > ∗ **right_**

Friends

- template<class T >
  std::ostream & **operator**<< (std::ostream &, const NodeB< T > &)

5.5.1 Constructor & Destructor Documentation

5.5.1.1 NodeB()

```
template<class Key >
NodeB< Key >::NodeB (
             const Key & data,
             NodeB< Key > * left = NULL,
             NodeB< Key > * right = NULL )
```

Construct a new Node B< Key>:: Node B object.

Template Parameters

Key	

Parameters

data	
left	
right	

5.5.1.2 ∼NodeB()

```
template<class Key >
NodeB< Key >::∼NodeB  [virtual]
```

Destroy the Node B< Key>:: Node B object.

Template Parameters

Key	

5.5.2 Member Function Documentation

5.5.2.1 getData()

```
template<class Key >
Key NodeB< Key >::getData  [inline]
```

Return the dato of node.

Template Parameters

Key	

Returns

Key

5.5.2.2 getLeft() [1/2]

```
template<class Key >
NodeB< Key > *& NodeB< Key >::getLeft  [inline]
```

Return the left node.

Template Parameters

Key	

Returns

NodeB<Key>*&

5.5.2.3 getLeft() [2/2]

```
template<class Key >
NodeB< Key > * NodeB< Key >::getLeft  [inline]
```

Return the left node.

Template Parameters

Key	

Returns

NodeB<Key>*

5.5.2.4 getRight() [1/2]

```
template<class Key >
NodeB< Key > *& NodeB< Key >::getRight  [inline]
```

Return the right node.

Template Parameters

Key	

Returns

NodeB<Key>*&

5.5.2.5 getRight() [2/2]

```
template<class Key >
NodeB< Key > * NodeB< Key >::getRight  [inline]
```

Return the right node.

Template Parameters

Key	

Returns

NodeB<Key>*

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

- include/NodeB.h

Chapter 6

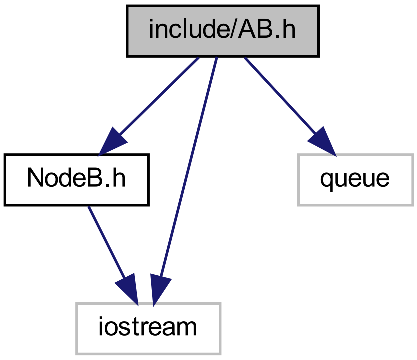
File Documentation

6.1 include/AB.h File Reference

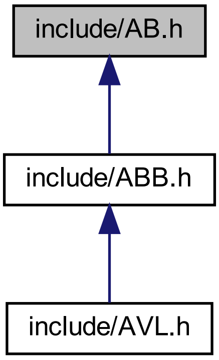
This class is Binary Tree.

```
#include "NodeB.h"
#include <iostream>
#include <queue>
```

Include dependency graph for AB.h:



This graph shows which files directly or indirectly include this file:



Classes

- class AB< Key >

Functions

- template<class Key >
  std::ostream & operator<< (std::ostream &os, const AB< Key > &ab_tree)

  This method print the tree.

6.1.1 Detailed Description

This class is Binary Tree.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-23

Copyright

Copyright (c) 2023

6.1.2 Function Documentation

6.1.2.1 operator<<()

```
template<class Key >
std::ostream & operator<< (
             std::ostream & os,
             const AB< Key > & ab_tree )
```

This method print the tree.

Template Parameters

Key	

Parameters

os	
ab_tree	

Returns

std::ostream&

6.2 AB.h

Go to the documentation of this file.

```

#pragma once
#include "NodeB.h"
#include <iostream>
#include <queue>

template <class Key>
class AB
{
private:
  int branchSize_(NodeB<Key> *) const;
  int heightN_(NodeB<Key> *) const;
  bool isBranchEquilibrium_(NodeB<Key> *) const;
  void prune_(NodeB<Key> *&);
  void route_(NodeB<Key> *) const;

protected:
  NodeB<Key> *root_;

public:
  AB();
  virtual ~AB();

  bool isEmpty(NodeB<Key> *);
  bool isLeafNode(NodeB<Key> *);
  inline int size() const;
  inline int height() const;
  inline bool isEquilibrium() const;

  virtual void insert(const Key &) = 0;
  virtual bool search(const Key &) const = 0;
  void inorden() const;

  template <class T>
  friend std::ostream &operator«(std::ostream &, const AB<T> &);
};

template <class Key>
AB<Key>::AB() : root_(NULL) {}

template <class Key>
AB<Key>::~AB()
{
  prune_(root_);
}

template <class Key>
void AB<Key>::prune_(NodeB<Key> *&node)
{
  if (node == NULL)
    return;
  prune_(node->getLeft());
  prune_(node->getRight());
  delete node;
  node = NULL;
}

template <class Key>
bool AB<Key>::isEmpty(NodeB<Key> *node)
{
  return node == NULL;
}

template <class Key>
bool AB<Key>::isLeafNode(NodeB<Key> *node)
{
  return !node->getLeft() && !node->getRight();
}

template <class Key>
int AB<Key>::branchSize_(NodeB<Key> *node) const
{
  if (node == NULL)
    return 0;
  return (1 + branchSize_(node->getLeft) + branchSize_(node->getRight()));
}

template <class Key>
int AB<Key>::size() const
{
  return branchSize_(root_);
}

template <class Key>
int AB<Key>::heightN_(NodeB<Key> *node) const
{
  if (node == NULL)
    return 0;
  int height_left = heightN_(node->getLeft());
  int height_right = heightN_(node->getRight());

  if (height_right > height_left)
    return ++height_right;
  else
    return ++height_left;
}

template <class Key>
int AB<Key>::height() const
{
  return heightN_(root_);
}

template <class Key>
bool AB<Key>::isBranchEquilibrium_(NodeB<Key> *node) const
{
  if (node == NULL)
    return true;
  const int eq = branchSize_(node->getLeft()) - branchSize_(node->getRight());
  switch (eq)
  {
  case -1:
  case 0:
  case 1:
    return isBranchEquilibrium_(node->getLeft()) && isBranchEquilibrium_(node->getRight());
  default:
    return false;
  }
}
```

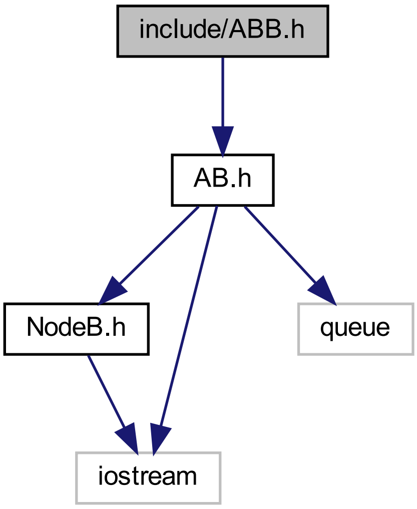
```

template <class Key>
bool AB<Key>::isEquilibrium() const
{
  return isBranchEquilibrium_(root_);
}

template <class Key>
void AB<Key>::inorden() const
{
  route_(root_);
}

template <class Key>
void AB<Key>::route_(NodeB<Key> *node) const
{
  if (node == NULL)
    return;
  route_(node->getLeft());
  std::cout « node->getData() « " ";
  route_(node->getRight());
}

template <class Key>
std::ostream &operator«(std::ostream &os, const AB<Key> &ab_tree)
{
  std::queue<std::pair<NodeB<Key> *, int» queue;
  queue.push(std::make_pair(ab_tree.root_, 0));
  int current_level = 0;

  std::cout « "Level " « current_level « ": ";
  while (!queue.empty())
  {
    std::pair<NodeB<Key> *, int> node_current = queue.front();
    queue.pop();

    if (node_current.second > current_level)
    {
      current_level++;
      os « "\nLevel " « current_level « ": ";
    }

    if (node_current.first != NULL)
    {
      os « "[" « *node_current.first « "] ";
      queue.push(std::make_pair(node_current.first->getLeft(), current_level + 1));
      queue.push(std::make_pair(node_current.first->getRight(), current_level + 1));
    }
    else
    {
      os « "[.] ";
    }
  }

  return os;
}
```

6.3 include/ABB.h File Reference

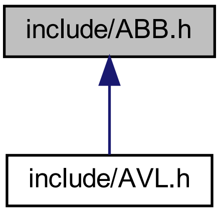
This class content ABB Binary tree search is derivated class of AB.

```
#include "AB.h"
```

Include dependency graph for ABB.h:



This graph shows which files directly or indirectly include this file:



Classes

- class ABB< Key >

6.3.1 Detailed Description

This class content ABB Binary tree search is derivated class of AB.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-25

Copyright

6.4 ABB.h

Go to the documentation of this file.

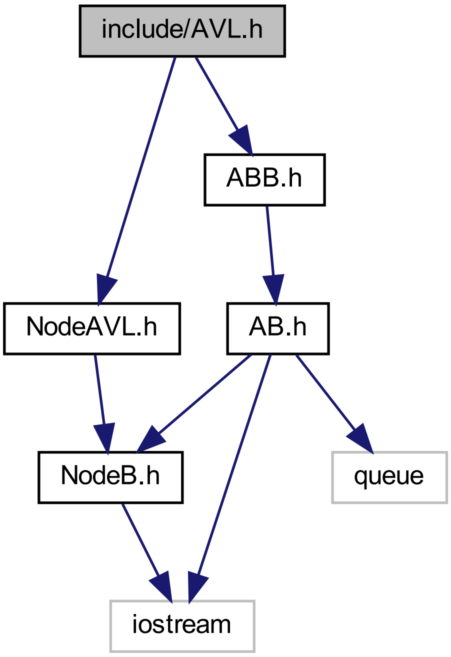
```cpp

#pragma once
#include "AB.h"

template <class Key>
class ABB : public AB<Key>
{
private:
  void insertBranch_(NodeB<Key> *&, const Key &);
  bool searchBranch_(NodeB<Key> *, const Key &) const;

public:
  ABB();
  ~ABB();
  void insert(const Key &) override;
  virtual bool search(const Key &) const override;
};

template <class Key>
ABB<Key>::ABB() : AB<Key>() {}

template <class Key>
void ABB<Key>::insertBranch_(NodeB<Key> *&node, const Key &key)
{
  if (node == NULL)
    node = new NodeB<Key>(key);
  else if (key < node->getData())
    insertBranch_(node->getLeft(), key);
  else
    insertBranch_(node->getRight(), key);
}

template <class Key>
ABB<Key>::~ABB() {}

template <class Key>
void ABB<Key>::insert(const Key &key)
{
  insertBranch_(this->root_, key);
}

template <class Key>
bool ABB<Key>::searchBranch_(NodeB<Key> *node, const Key &key) const
{
  if (node == NULL)
    return false;
  else if (key == node->getData())
    return true;
  else if (key < node->getData())
    return searchBranch_(node->getLeft(), key);
  else
    return searchBranch_(node->getRight(), key);
}

template <class Key>
bool ABB<Key>::search(const Key &key) const
{
  return searchBranch_(this->root_, key);
}
```

6.5 include/AVL.h File Reference

This class content AVL Binary tree search is derivated class of ABB.

```
#include "ABB.h"
#include "NodeAVL.h"
```

Include dependency graph for AVL.h:



Classes

- class AVL< Key >

6.5.1 Detailed Description

This class content AVL Binary tree search is derivated class of ABB.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-30

Copyright

Copyright (c) 2023

6.6 AVL.h

Go to the documentation of this file.

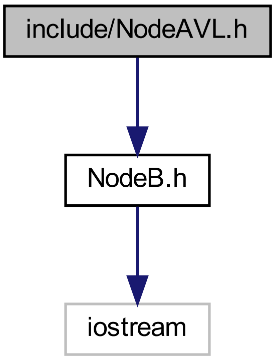
```

#include "ABB.h"
#include "NodeAVL.h"

template <class Key>
class AVL : public ABB<Key>
{
private:
  void rotationLL_(NodeAVL<Key> *&);
  void rotationRR_(NodeAVL<Key> *&);
  void rotationLR_(NodeAVL<Key> *&);
  void rotationRL_(NodeAVL<Key> *&);

  void insertBalanceRight_(NodeAVL<Key> *&, bool &);
  void insertBalanceLeft_(NodeAVL<Key> *&, bool &);
  void insertBalance_(NodeAVL<Key> *&, NodeAVL<Key> *, bool &);

public:
  AVL();
  ~AVL();
  inline NodeAVL<Key> *getRootAVL() const;
  inline NodeAVL<Key> *&getRootAVL();

  void insert(const Key &k) override;
};

template <class Key>
AVL<Key>::AVL() : ABB<Key>() {}

template <class Key>
AVL<Key>::~AVL() {}

template <class Key>
NodeAVL<Key> *AVL<Key>::getRootAVL() const
{
  return reinterpret_cast<NodeAVL<Key> *>(this->root_);
}

template <class Key>
NodeAVL<Key> *&AVL<Key>::getRootAVL()
{
  return reinterpret_cast<NodeAVL<Key> *&>(this->root_);
}

template <class Key>
void AVL<Key>::insert(const Key &key)
{

  bool grow = false;
  insertBalance_(getRootAVL(), new NodeAVL<Key>(key), grow);
}

template <class Key>
void AVL<Key>::rotationLL_(NodeAVL<Key> *&node)
{
  #ifdef TRACE
  std::cout « "\nUnbalance" « std::endl;
  std::cout « *this « std::endl;
  std::cout « "\nRotation Left-Left in [" « node->getData() « "]" « std::endl;
  #endif

  NodeAVL<Key> *node1 = node->getLeftAVL();
  node->getLeftAVL() = node1->getRightAVL();
  node1->getRightAVL() = node;

  if (node1->getBalanceFactor() == 1)
  {
    node->getBalanceFactor() = 0;
    node1->getBalanceFactor() = 0;
  }
  else
  {
    node->getBalanceFactor() = 1;
    node1->getBalanceFactor() = -1;
  }
  node = node1;
}

template <class Key>
void AVL<Key>::rotationRR_(NodeAVL<Key> *&node)
{
  #ifdef TRACE
  std::cout « "\nUnbalance" « std::endl;
  std::cout « *this « std::endl;
  std::cout « "\nRotation Right-Right in [" « node->getData() « "]" « std::endl;
  #endif

  NodeAVL<Key> *node1 = node->getRightAVL();
  node->getRightAVL() = node1->getLeftAVL();
  node1->getLeftAVL() = node;

  if (node1->getBalanceFactor() == -1)
  {
    node->getBalanceFactor() = 0;
    node1->getBalanceFactor() = 0;
  }
  else
  {
    node->getBalanceFactor() = -1;
    node1->getBalanceFactor() = 1;
  }
  node = node1;
}

template <class Key>
void AVL<Key>::rotationLR_(NodeAVL<Key> *&node)
{
  #ifdef TRACE
  std::cout « "\nUnbalance" « std::endl;
  std::cout « *this « std::endl;
  std::cout « "\nRotation Left-Right in [" « node->getData() « "]" « std::endl;
  #endif

  NodeAVL<Key> *node1 = node->getLeftAVL();
  NodeAVL<Key> *node2 = node1->getRightAVL();
  node->getLeftAVL() = node2->getRightAVL();
  node2->getRightAVL() = node;
  node1->getRightAVL() = node2->getLeftAVL();
  node2->getLeftAVL() = node1;

  if (node2->getBalanceFactor() == -1)
  {
    node1->getBalanceFactor() = 1;
  }
  else
  {
    node1->getBalanceFactor() = 0;
  }
  if (node2->getBalanceFactor() == 1)
  {
    node->getBalanceFactor() = -1;
  }
  else
  {
    node->getBalanceFactor() = 0;
  }
  node2->getBalanceFactor() = 0;
  node = node2;
}

template <class Key>
void AVL<Key>::rotationRL_(NodeAVL<Key> *&node)
{
  #ifdef TRACE
  std::cout « "\nUnbalance" « std::endl;
  std::cout « *this « std::endl;
  std::cout « "\nRotation Right-Left in [" « node->getData() « "]" « std::endl;
  #endif

  NodeAVL<Key> *node1 = node->getRightAVL();
  NodeAVL<Key> *node2 = node1->getLeftAVL();
  node->getRightAVL() = node2->getLeftAVL();
  node2->getLeftAVL() = node;
  node1->getLeftAVL() = node2->getRightAVL();
  node2->getRightAVL() = node1;

  if (node2->getBalanceFactor() == 1)
  {
    node1->getBalanceFactor() = -1;
  }
  else
  {
    node1->getBalanceFactor() = 0;
  }
  if (node2->getBalanceFactor() == -1)
  {
    node->getBalanceFactor() = 1;
  }
  else
  {
    node->getBalanceFactor() = 0;
  }
  node2->getBalanceFactor() = 0;
  node = node2;
}

template <class Key>
void AVL<Key>::insertBalanceLeft_(NodeAVL<Key> *&node, bool &grow)
{
  switch (node->getBalanceFactor())
  {
  case -1:
    node->getBalanceFactor() = 0;
    grow = false;
    break;
  case 0:
    node->getBalanceFactor() = 1;
    break;
  case 1:
    NodeAVL<Key> *node1 = node->getLeftAVL();
    if (node1->getBalanceFactor() == 1)
      rotationLL_(node);
    else
      rotationLR_(node);
    grow = false;
    break;
  }
}

template <class Key>
void AVL<Key>::insertBalanceRight_(NodeAVL<Key> *&node, bool &grow)
{
  switch (node->getBalanceFactor())
  {
  case 1:
    node->getBalanceFactor() = 0;
    grow = false;
    break;
  case 0:
    node->getBalanceFactor() = -1;
    break;
  case -1:
    NodeAVL<Key> *node1 = node->getRightAVL();
    if (node1->getBalanceFactor() == -1)
      rotationRR_(node);
    else
      rotationRL_(node);
    grow = false;
    break;
  }
}

template <class Key>
void AVL<Key>::insertBalance_(NodeAVL<Key> *&node, NodeAVL<Key> *newNode, bool &grow)
{
  if (node == NULL)
  {
    node = newNode;
    grow = true;
  }
  if (node->getData() == newNode->getData())
    return;

  if (node->getData() > newNode->getData())
  {
    insertBalance_(node->getLeftAVL(), newNode, grow);
    if (grow)
      insertBalanceLeft_(node, grow);
  }
  else
  {
    insertBalance_(node->getRightAVL(), newNode, grow);
    if (grow)
      insertBalanceRight_(node, grow);
  }
}
```

6.7 include/NodeAVL.h File Reference

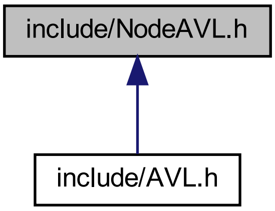
This class content Node AVL Binary tree search is derivated class of NodeB.

```
#include "NodeB.h"
```

Include dependency graph for NodeAVL.h:



This graph shows which files directly or indirectly include this file:



Classes

- class NodeAVL< Key >

6.7.1 Detailed Description

This class content Node AVL Binary tree search is derivated class of NodeB.

Author

Fabrizzio Daniell Perilli Martín `alu0101138589@ull.edu.es`

Version

0.1

Date

2023-04-30

Copyright

6.8 NodeAVL.h

Go to the documentation of this file.

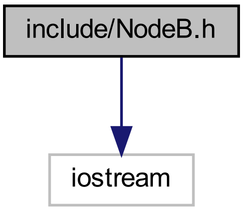
```
#include "NodeB.h"

template <class Key>
class NodeAVL : public NodeB<Key>
{
private:
  int balance_factor_;

public:
  NodeAVL(const Key &, NodeAVL<Key> * = NULL, NodeAVL<Key> * = NULL);
  ~NodeAVL();

  inline int getBalanceFactor() const;
  inline int &getBalanceFactor();

  inline NodeAVL<Key> *getLeftAVL() const;
  inline NodeAVL<Key> *&getLeftAVL();
  inline NodeAVL<Key> *getRightAVL() const;
  inline NodeAVL<Key> *&getRightAVL();
};

template <class Key>
NodeAVL<Key>::NodeAVL(const Key &data, NodeAVL<Key> *left, NodeAVL<Key> *right) : NodeB<Key>(data, left,
    right), balance_factor_(0) {}

template <class Key>
NodeAVL<Key>::~NodeAVL() {}

template <class Key>
int NodeAVL<Key>::getBalanceFactor() const
{
  return balance_factor_;
}

template <class Key>
int &NodeAVL<Key>::getBalanceFactor()
{
  return balance_factor_;
}

template <class Key>
NodeAVL<Key> *NodeAVL<Key>::getLeftAVL() const
{
  return reinterpret_cast<NodeAVL<Key> *>(this->left_);
}

template <class Key>
NodeAVL<Key> *&NodeAVL<Key>::getLeftAVL()
{
  return reinterpret_cast<NodeAVL<Key> *&>(this->left_);
}

template <class Key>
NodeAVL<Key> *NodeAVL<Key>::getRightAVL() const
{
  return reinterpret_cast<NodeAVL<Key> *>(this->right_);
}

template <class Key>
NodeAVL<Key> *&NodeAVL<Key>::getRightAVL()
{
  return reinterpret_cast<NodeAVL<Key> *&>(this->right_);
}
```

6.9 include/NodeB.h File Reference

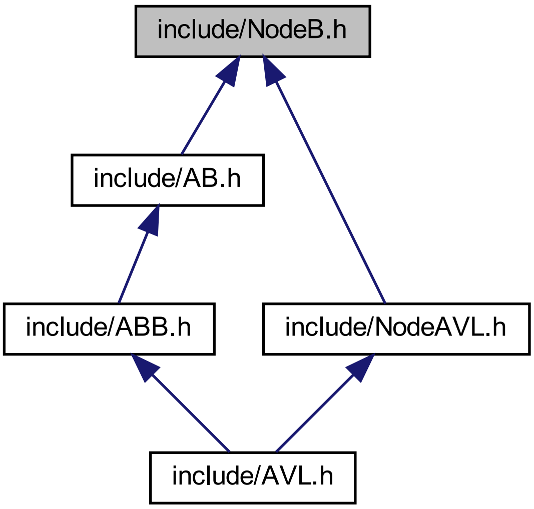
This class contains the node of binary tree.

```
#include <iostream>
```

Include dependency graph for NodeB.h:



This graph shows which files directly or indirectly include this file:



Classes

- class NodeB< Key >

Functions

- template<class Key >
  std::ostream & **operator**<< (std::ostream &os, const NodeB< Key > &node)

6.9.1 Detailed Description

This class contains the node of binary tree.

Author

Fabrizzio Daniell Perilli Martín alu0101138589@ull.edu.es

Version

0.1

Date

2023-04-23

Copyright

Copyright (c) 2023

6.10 NodeB.h

Go to the documentation of this file.

```

#pragma once
#include <iostream>

template <class Key>
class NodeB
{
private:
  Key data_;

protected:
  NodeB<Key> *left_;
  NodeB<Key> *right_;

public:
  NodeB(const Key &, NodeB<Key> * = NULL, NodeB<Key> * = NULL);
  virtual ~NodeB();

  inline Key getData() const;

  inline NodeB<Key> *getLeft() const;
  inline NodeB<Key> *&getLeft();

  inline NodeB<Key> *getRight() const;
  inline NodeB<Key> *&getRight();

  template <class T>
  friend std::ostream &operator«(std::ostream &, const NodeB<T> &);
};

template <class Key>
NodeB<Key>::NodeB(const Key &data, NodeB<Key> *left, NodeB<Key> *right) : data_(data), left_(left),
    right_(right) {}

template <class Key>
NodeB<Key>::~NodeB() {}

template <class Key>
Key NodeB<Key>::getData() const
{
  return data_;
}

template <class Key>
NodeB<Key> *NodeB<Key>::getLeft() const
```

```
{
  return left_;
}

template <class Key>
NodeB<Key> *&NodeB<Key>::getLeft()
{
  return left_;
}

template <class Key>
NodeB<Key> *NodeB<Key>::getRight() const
{
  return right_;
}

template <class Key>
NodeB<Key> *&NodeB<Key>::getRight()
{
  return right_;
}

template <class Key>
std::ostream &operator«(std::ostream &os, const NodeB<Key> &node)
{
  os « node.data_;
  return os;
}
```

Index