Compression task

Generated by Doxygen 1.9.8

1 Class Index

1.1 Class List

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

2 File Index

2.1 File List

Here is a list of all files with brief descriptions:

3 Class Documentation

3.1 ClearClosingTagsComp Class Reference

```
#include <ClearClosingTagsComp.h>
```

Public Member Functions

- ClearClosingTagsComp (const std::string ∗xmlFile)
  C'tor.
- std::string ∗ compress ()
  This function compresses the XML file.

3.1.1 Detailed Description

Definition at line 32 of file ClearClosingTagsComp.h.

3.1.2 Constructor & Destructor Documentation

ClearClosingTagsComp()

```
ClearClosingTagsComp::ClearClosingTagsComp (
            const std::string * xmlFile )  [inline], [explicit]
```

C'tor.

Initializes the XML file.

XML version and encoding line example:

```
  <?xml version="1.0" encoding="UTF-8"?>
*
```

Parameters

the	XML file without the XML version and encoding line.

Definition at line 51 of file ClearClosingTagsComp.h.

3.1.3 Member Function Documentation

compress()

```
std::string * ClearClosingTagsComp::compress ( )
```

This function compresses the XML file.

Operation summary:

- Find the closing tag.
- Delete the closing tag.

Returns

The result string doesn't contain XML version and encoding line.

Warning

use only with social network data.

Definition at line 29 of file ClearClosingTagsComp.cpp.

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

- ClearClosingTagsComp.h
- ClearClosingTagsComp.cpp

3.2 ClearClosingTagsDec Class Reference

```
#include <ClearClosingTagsDec.h>
```

Public Member Functions

- ClearClosingTagsDec (const std::string *xmlFile)
  C'tor that initializes the XML string with provided value, and empty tagsTree, and empty tagsStack.
- ~ClearClosingTagsDec ()
  D'tor.
- std::string * decompress () const
  This function decompresses the XML file.

3.2.1 Detailed Description

Definition at line 32 of file ClearClosingTagsDec.h.

3.2.2 Constructor & Destructor Documentation

ClearClosingTagsDec()

```
ClearClosingTagsDec::ClearClosingTagsDec (
            const std::string * xmlFile )  [inline], [explicit]
```

C'tor that initializes the XML string with provided value, and empty tagsTree, and empty tagsStack.

XML version and encoding line example:

```
  <?xml version="1.0" encoding="UTF-8"?>
*
```

Parameters

the	XML file without the XML version and encoding line.

Definition at line 92 of file ClearClosingTagsDec.h.

~ClearClosingTagsDec()

```
ClearClosingTagsDec::~ClearClosingTagsDec ( )  [inline]
```

D'tor.

Definition at line 98 of file ClearClosingTagsDec.h.

3.2.3 Member Function Documentation

decompress()

```
std::string * ClearClosingTagsDec::decompress ( ) const
```

This function decompresses the XML file.

Operation summary:

- collect each opening tag in the stack.
- when reaching a new tag, check from the tree if the current tag (in top of the stack) contains the next tag as a child this don't close the current tag, and add the next tag into the stack.
- if it wasn't a child, add a closing tag for the current tag, then check the same with the next value in the stack.
- At the end add all the remaining tags in the stack in their order.

Returns

The result string doesn't contain XML version and encoding line.

Warning

use only with social network data.

Definition at line 59 of file ClearClosingTagsDec.cpp.

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

- ClearClosingTagsDec.h
- ClearClosingTagsDec.cpp

3.3 HuffmanComp Class Reference

`#include <HuffmanComp.h>`

Public Member Functions

- HuffmanComp (std::string *fileC)
  C'tor.
- ~HuffmanComp ()
  D'tor.
- std::string * compress ()
  Compresses the file content using the Huffman tree.

3.3.1 Detailed Description

Definition at line 19 of file HuffmanComp.h.

3.3.2 Constructor & Destructor Documentation

HuffmanComp()

```
HuffmanComp::HuffmanComp (
            std::string * fileC )  [inline], [explicit]
```

C'tor.

It will construct the tree from the HuffmanTree::generateTreeFromText() static method.

Parameters

fileC	file to compress

Definition at line 36 of file HuffmanComp.h.

~HuffmanComp()

```
HuffmanComp::~HuffmanComp ( )  [inline]
```

D'tor.

Definition at line 43 of file HuffmanComp.h.

3.3.3 Member Function Documentation

compress()

```
std::string * HuffmanComp::compress ( )
```

Compresses the file content using the Huffman tree.

This function compresses the file content using the Huffman tree's encoded representation. It encodes each character in the file content using the Huffman tree and returns a pointer to the compressed string.

Returns

A pointer to the compressed string of zeros and ones according to the encoding of each char.

Definition at line 16 of file HuffmanComp.cpp.

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

- HuffmanComp.h
- HuffmanComp.cpp

3.4 HuffmanDec Class Reference

```
#include <HuffmanDec.h>
```

Public Member Functions

- HuffmanDec (std::string *file)
  C'tor.
- ~HuffmanDec ()
  D'tor.
- std::string * decompress ()
  Decompresses the file content using the Huffman tree.

3.4.1 Detailed Description

Definition at line 17 of file HuffmanDec.h.

3.4.2 Constructor & Destructor Documentation

HuffmanDec()

```
HuffmanDec::HuffmanDec (
            std::string * file )  [explicit]
```

C'tor.

It will construct the tree from the HuffmanTree::rebuildTree() static method, and the encoded tree line (the first line in the compressed file.)

Parameters

fileC	file to decompress

Definition at line 14 of file HuffmanDec.cpp.

~HuffmanDec()

```
HuffmanDec::~HuffmanDec ( )  [inline]
```

D'tor.

Definition at line 42 of file HuffmanDec.h.

3.4.3 Member Function Documentation

decompress()

```
std::string * HuffmanDec::decompress ( )
```

Decompresses the file content using the Huffman tree.

This function decompresses the file content using the Huffman tree's encoded representation. It decodes the bits based on the Huffman tree and returns the decompressed string.

Returns

A pointer to the decompressed string.

Definition at line 36 of file HuffmanDec.cpp.

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

- HuffmanDec.h
- HuffmanDec.cpp

3.5 HuffmanTree Class Reference

```
#include <HuffmanTree.h>
```

Public Member Functions

- HuffmanTree (HuffmanTreeNode *root)
  An empty C'tor. Initialize the root with nullptr.
- ~HuffmanTree ()
  D'tor that deallocates the root. the root will delete all its children.
- std::string getEncodedTree ()
  Retrieves the encoding for a specific character from the tree.
- std::vector< bool > getEncodingFromChar (char c)
  Get the Huffman encoding for a specific character in the tree.
- char getCharFromEncoding (std::vector< bool > encoding)
  Get the character represented by a specific Huffman encoding in the tree.

Static Public Member Functions

- static HuffmanTree * generateTreeFromText (const std::string &text)

  Generates a Huffman tree from the provided text.
- static HuffmanTree * rebuildTree (const std::string &encodedTree)

  Rebuilds a Huffman tree from its encoded representation (The first line in the compressed text that ends with ")\n").

3.5.1 Detailed Description

Definition at line 23 of file HuffmanTree.h.

3.5.2 Constructor & Destructor Documentation

HuffmanTree()

```
HuffmanTree::HuffmanTree (
            HuffmanTreeNode * root )  [inline], [explicit]
```

An empty C'tor. Initialize the root with nullptr.

Definition at line 82 of file HuffmanTree.h.

~HuffmanTree()

```
HuffmanTree::~HuffmanTree ( )  [inline]
```

D'tor that deallocates the root. the root will delete all its children.

Definition at line 87 of file HuffmanTree.h.

3.5.3 Member Function Documentation

generateTreeFromText()

```
HuffmanTree * HuffmanTree::generateTreeFromText (
            const std::string & text )  [static]
```

Generates a Huffman tree from the provided text.

Parameters

text	The input text used to construct the tree.

Returns

A pointer to the generated HuffmanTree object.

Definition at line 90 of file HuffmanTree.cpp.

getCharFromEncoding()

```
char HuffmanTree::getCharFromEncoding (
            std::vector< bool > encoding )
```

Get the character represented by a specific Huffman encoding in the tree.

This function retrieves the character represented by a given Huffman encoding (binary representation) within the Huffman tree. It uses a helper function findCharFromEncodingHelper to traverse the tree and find the character.

Pass the bits in its order from left to right in the vector, i.e., MSB (left most bit) at position zero.

Parameters

encoding	A vector of boolean values representing the binary encoding to decode.

Returns

The character represented by the given encoding.

Definition at line 271 of file HuffmanTree.cpp.

getEncodedTree()

```
std::string HuffmanTree::getEncodedTree ( )
```

Retrieves the encoding for a specific character from the tree.

The string will be in this formate:

- Parentheses-based representation: "((A:01)(B:10)\n", where each node and its encoding are enclosed in parentheses.

Returns

The encoding tree string to add to the compressed file.

Definition at line 198 of file HuffmanTree.cpp.

getEncodingFromChar()

```
std::vector< bool > HuffmanTree::getEncodingFromChar (
            char c )
```

Get the Huffman encoding for a specific character in the tree.

This function retrieves the Huffman encoding (binary representation) for a specific character 'c' within the Huffman tree. It uses a helper function findCharEncodingHelper to traverse the tree and find the encoding.

Parameters

c	The character to find the encoding for.

Returns

A vector of boolean values representing the binary encoding of the character.

Definition at line 254 of file HuffmanTree.cpp.

rebuildTree()

```
HuffmanTree * HuffmanTree::rebuildTree (
            const std::string & encodedTree )  [static]
```

Rebuilds a Huffman tree from its encoded representation (The first line in the compressed text that ends with ")\n").

If will be in this formate:

- Parentheses-based representation: "((A:01)(B:10))\n", where each node and its encoding are enclosed in parentheses.

Parameters

encodedTree	The encoded representation of the tree, without the 1st "(" and ")\n".

Returns

A pointer to the reconstructed HuffmanTree object.

Definition at line 131 of file HuffmanTree.cpp.

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

- HuffmanTree.h
- HuffmanTree.cpp

3.6 HuffmanTreeNode Class Reference

```
#include <HuffmanTreeNode.h>
```

Collaboration diagram for HuffmanTreeNode:

Public Member Functions

- HuffmanTreeNode ()

  An empty C'tor. Initialize the c with '\o', freq with 0, both children with nullptr.
- HuffmanTreeNode (int frequency, char character, HuffmanTreeNode *left, HuffmanTreeNode *right)

  Constructs a HuffmanTreeNode with specified frequency, character, left, and right children.
- HuffmanTreeNode (int frequency, char character)

  Constructs a leaf HuffmanTreeNode with specified frequency and character.
- ~HuffmanTreeNode ()

  D'tor that deallocates the children.
- bool operator< (const HuffmanTreeNode &other) const

  Overloads the less-than operator (<) to compare HuffmanTreeNode objects based on their frequencies. But we need the smallest nodes at the start of the priority_queue, so do the opposite in the implementation.
- bool operator() (const HuffmanTreeNode *x, const HuffmanTreeNode *y)

  Overloaded () operator for comparing HuffmanTreeNode pointers.

Public Attributes

- int freq
- char c
- HuffmanTreeNode * leftChild
- HuffmanTreeNode * rightChild

Friends

- class HuffmanTree

3.6.1 Detailed Description

Definition at line 13 of file HuffmanTreeNode.h.

3.6.2 Constructor & Destructor Documentation

HuffmanTreeNode() [1/3]

```
HuffmanTreeNode::HuffmanTreeNode ( )  [inline], [explicit]
```

An empty C'tor. Initialize the c with '\o', freq with 0, both children with nullptr.

Definition at line 36 of file HuffmanTreeNode.h.

HuffmanTreeNode() [2/3]

```
HuffmanTreeNode::HuffmanTreeNode (
            int frequency,
            char character,
            HuffmanTreeNode * left,
            HuffmanTreeNode * right )  [inline]
```

Constructs a HuffmanTreeNode with specified frequency, character, left, and right children.

Parameters

frequency	The frequency associated with the node.
character	The character associated with the node.
left	A pointer to the left child node.
right	A pointer to the right child node.

Definition at line 46 of file HuffmanTreeNode.h.

HuffmanTreeNode() [3/3]

```
HuffmanTreeNode::HuffmanTreeNode (
            int frequency,
            char character )  [inline]
```

Constructs a leaf HuffmanTreeNode with specified frequency and character.

Parameters

frequency	The frequency associated with the leaf node.
character	The character associated with the leaf node.

Definition at line 54 of file HuffmanTreeNode.h.

~HuffmanTreeNode()

```
HuffmanTreeNode::~HuffmanTreeNode ( )  [inline]
```

D'tor that deallocates the children.

Definition at line 61 of file HuffmanTreeNode.h.

3.6.3 Member Function Documentation

operator()()

```
bool HuffmanTreeNode::operator() (
            const HuffmanTreeNode * x,
            const HuffmanTreeNode * y )  [inline]
```

Overloaded () operator for comparing HuffmanTreeNode pointers.

This operator compares two HuffmanTreeNode pointers based on the comparison of the nodes themselves.

Parameters

x	Pointer to the first HuffmanTreeNode to compare.
y	Pointer to the second HuffmanTreeNode to compare.

Returns

True if the first node has higher priority than the second node.

Definition at line 92 of file HuffmanTreeNode.h.

operator<()

```
bool HuffmanTreeNode::operator< (
            const HuffmanTreeNode & other ) const  [inline]
```

Overloads the less-than operator (<) to compare HuffmanTreeNode objects based on their frequencies. But we need the smallest nodes at the start of the priority_queue, so do the opposite in the implementation.

Parameters

other	The HuffmanTreeNode object to compare with.

Returns

True if the frequency of this node is more than the frequency of the other node, false otherwise.

Definition at line 80 of file HuffmanTreeNode.h.

3.6.4 Friends And Related Symbol Documentation

HuffmanTree

```
friend class HuffmanTree  [friend]
```

Definition at line 15 of file HuffmanTreeNode.h.

3.6.5 Member Data Documentation

c

```
char HuffmanTreeNode::c
```

Definition at line 23 of file HuffmanTreeNode.h.

freq

```
int HuffmanTreeNode::freq
```

Definition at line 18 of file HuffmanTreeNode.h.

leftChild

```
HuffmanTreeNode* HuffmanTreeNode::leftChild
```

Definition at line 26 of file HuffmanTreeNode.h.

rightChild

```
HuffmanTreeNode* HuffmanTreeNode::rightChild
```

Definition at line 27 of file HuffmanTreeNode.h.

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

- HuffmanTreeNode.h

3.7 Map Class Reference

```
#include <Map.h>
```

Public Member Functions

- Map ()
  C'tor. Initializes empty map with an empty dynamic array.
- Map (const std::string *tagMapBlock)
  C'tor. Initialize the map from a <TagMap> block.
- ~Map ()
- int add (std::string *key)
  Adds the key to the map.
- int getValue (const std::string *key) const
  The value that the key is mapped to.
- const std::string * getKey (int value) const
  Get the key from the value that the key was mapped to.
- bool containKey (const std::string *key) const
  Checks if the map contains that key.
- int getSize ()
- std::string * toString ()
  Returns the <TagMap> block so it can be added to the compressed XML file.

3.7.1 Detailed Description

Definition at line 21 of file Map.h.

3.7.2 Constructor & Destructor Documentation

Map() [1/2]

```
Map::Map ( )  [inline], [explicit]
```

C'tor. Initializes empty map with an empty dynamic array.

Definition at line 39 of file Map.h.

Map() [2/2]

```
Map::Map (
            const std::string * tagMapBlock )  [explicit]
```

C'tor. Initialize the map from a <TagMap> block.

The file must start with <TagMap> and ends with </TagMap> otherwise the file is considered defected.

Parameters

tagMapBlock.	

Exceptions

runtime_error	if the file is defected.

Definition at line 22 of file Map.cpp.

~Map()

```
Map::~Map ( )  [inline]
```

D'tor.

Warning

It will delete all the keys string assigned to it.

Definition at line 54 of file Map.h.

3.7.3 Member Function Documentation

add()

```
int Map::add (
            std::string * key )
```

Adds the key to the map.

Parameters

key	To add.

Returns

The value that the key is mapped to.

Definition at line 63 of file Map.cpp.

containKey()

```
bool Map::containKey (
            const std::string * key ) const
```

Checks if the map contains that key.

Parameters

key.	

Returns

true if the key is available in the map, false otherwise.

Definition at line 92 of file Map.cpp.

getKey()

```
const std::string * Map::getKey (
            int value ) const
```

Get the key from the value that the key was mapped to.

Parameters

value	

Returns

The key.

Exceptions

runtime	error if the value is not in the map.

Definition at line 81 of file Map.cpp.

getSize()

```
int Map::getSize ( )  [inline]
```

Returns

the size of the map.

Definition at line 97 of file Map.h.

getValue()

```
int Map::getValue (
            const std::string * key ) const
```

The value that the key is mapped to.

Parameters

key.	

Returns

The value if the key is found, -1 otherwise.

Definition at line 69 of file Map.cpp.

toString()

```
std::string * Map::toString ( )
```

Returns the <TagMap> block so it can be added to the compressed XML file.

Exceptions

runtime	error if the map is empty.

Definition at line 96 of file Map.cpp.

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

- Map.h
- Map.cpp

3.8 MinifyingXML Class Reference

```
#include <MinifyingXML.h>
```

Public Member Functions

- MinifyingXML (const std::string *xmlFile)

  C'tor.
- std::string * minifyString ()

  This function deletes any spaces and new lines from the XML File.
- void skipFromBeginning (std::string *result) const

  This function clears all the skip Chars except some spaces.
- void skipFromEnd (std::string *result) const

  This function clears the extra spaces left from the prev step.
- const std::string * getXMLFile () const
- void setXMLFile (const std::string *xmlFileNew)

Static Public Member Functions

- static bool isSkipChar (const char c)

  function checks if the char is one the located characters.

3.8.1 Detailed Description

Definition at line 29 of file MinifyingXML.h.

3.8.2 Constructor & Destructor Documentation

MinifyingXML()

```
MinifyingXML::MinifyingXML (
            const std::string * xmlFile )  [inline], [explicit]
```

C'tor.

See also

D'tor, this class will not deallocate the XML file string.

Parameters

xmlFile	

Definition at line 45 of file MinifyingXML.h.

3.8.3 Member Function Documentation

getXMLFile()

```
const std::string * MinifyingXML::getXMLFile ( ) const  [inline]
```

Definition at line 121 of file MinifyingXML.h.

isSkipChar()

```
bool MinifyingXML::isSkipChar (
            const char c )  [static]
```

function checks if the char is one the located characters.

Parameters

c	-The character to check.

Returns

- True if it's a skip char.
  - False otherwise.

See also

MinifyingXML::charToSkip array.

Definition at line 130 of file MinifyingXML.cpp.

minifyString()

```
std::string * MinifyingXML::minifyString ( )
```

This function deletes any spaces and new lines from the XML File.

It removes any charToSkip from the file string, except spaces in the tags values (leading and trailing spaces are removed from the value too).

See also

MinifyingXML::charToSkip array.

Returns

The result string from minifying function.

Definition at line 29 of file MinifyingXML.cpp.

setXMLFile()

```
void MinifyingXML::setXMLFile (
            const std::string * xmlFileNew )  [inline]
```

Definition at line 125 of file MinifyingXML.h.

skipFromBeginning()

```
void MinifyingXML::skipFromBeginning (
            std::string * result ) const
```

This function clears all the skip Chars except some spaces.

Operation:

- For all the string, skip (don't add it into the result) all charToskip elements except spaces.
- For space cases: -> Starting from the beginning, skip any spaces until reaching the first closing tag '>'. -> After the closing tab, skip any spaces until reaching the first non skip value (any chars not in charToSkip array). -> Add all spaces until reaching the next opening tag '<'.

Example: " <name> Ahmed Ali \n </name>" --> "<name>Ahmed Ali </name>"

Parameters

result	an empty string to store the result of this function.

Definition at line 54 of file MinifyingXML.cpp.

skipFromEnd()

```
void MinifyingXML::skipFromEnd (
            std::string * result ) const
```

This function clears the extra spaces left from the prev step.

Operation: Starting from the last element.

- Clear all the spaces before any opening tag '<' till the prev last non skip char. -> If the current element was the opening tag, skip spaces. -> If the current element is any non skip char, stop skipping spaces.

Example: "<name>Ahmed Ali </name>" --> "<name>Ahmed Ali</name>"

- Other skip chars are eliminated from the prev step.

See also

void MinifyingXML::skipFromBeginning(std::string* result).

Parameters

result	The result string from the prev step to modify it.

Definition at line 98 of file MinifyingXML.cpp.

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

- MinifyingXML.h
- MinifyingXML.cpp

3.9 TagsMapComp Class Reference

`#include <TagsMapComp.h>`

Public Member Functions

- TagsMapComp (const std::string *xmlFile)
  C'tor.
- ~TagsMapComp ()
  D'tor.
- void mapTags ()
  Reads the XML file and create the map with all the tags.
- std::string * compress (bool addMapTable=false)
  This function compresses the XML file.

3.9.1 Detailed Description

Definition at line 37 of file TagsMapComp.h.

3.9.2 Constructor & Destructor Documentation

TagsMapComp()

```
TagsMapComp::TagsMapComp (
            const std::string * xmlFile )  [inline], [explicit]
```

C'tor.

Initializes the XML file, reads it and create a map with all the tags.

Parameters

the	XML file without the XML version and encoding line.

Definition at line 53 of file TagsMapComp.h.

~TagsMapComp()

```
TagsMapComp::~TagsMapComp ( )  [inline]
```

D'tor.

Definition at line 61 of file TagsMapComp.h.

3.9.3 Member Function Documentation

compress()

```
std::string * TagsMapComp::compress (
            bool addMapTable = false )
```

This function compresses the XML file.

Operation:

- If addMapTable is true, it adds the <TagMap> block in the first line in the string. Will not added otherwise (is false by default).
- It replaces all the tags (closing and opening) with there mapped value in the map. Example: <TagEg> --> <0>, </TagEg> --> </0>

Parameters

addMapTable	if true, then a <TagMap> block will be added in the 1st line in the result string.

Returns

A string contains the XML file after the compression.

Note

The result string doesn't contain XML version and encoding line.

Definition at line 68 of file TagsMapComp.cpp.

mapTags()

```
void TagsMapComp::mapTags ( )
```

Reads the XML file and create the map with all the tags.

Explanation:

- Find the next tag.
- If the tag is in the map, do nothing.
- If the tag is not in the map add it.

Definition at line 39 of file TagsMapComp.cpp.

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

- TagsMapComp.h
- TagsMapComp.cpp

3.10 TagsMapDec Class Reference

`#include <TagsMapDec.h>`

Public Member Functions

- TagsMapDec (const std::string *xmlFile)
  C'tor. -Initialize the Map with the TagMap block.
- ~TagsMapDec ()
  D'tor.
- std::string * decompress ()
  This method decompresses the XML file.

3.10.1 Detailed Description

Definition at line 36 of file TagsMapDec.h.

3.10.2 Constructor & Destructor Documentation

TagsMapDec()

```
TagsMapDec::TagsMapDec (
            const std::string * xmlFile )  [inline], [explicit]
```

C'tor. -Initialize the Map with the TagMap block.

Parameters

the	XML file without the XML version and encoding line.

Definition at line 73 of file TagsMapDec.h.

~TagsMapDec()

```
TagsMapDec::~TagsMapDec ( )  [inline]
```

D'tor.

Definition at line 83 of file TagsMapDec.h.

3.10.3 Member Function Documentation

decompress()

```
std::string * TagsMapDec::decompress ( )
```

This method decompresses the XML file.

See also

TagMapComp::compress() for the functionality.

Returns

the file data after decompression.

Definition at line 74 of file TagsMapDec.cpp.

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

- TagsMapDec.h
- TagsMapDec.cpp

3.11 Tree Class Reference

`#include <Tree.h>`

Public Member Functions

- Tree ()
  Initialize the Tree in that arrange for social network system:
- ~Tree ()
  D'tor.
- TreeNode * getRoot ()
- void print () const

3.11.1 Detailed Description

Definition at line 28 of file Tree.h.

3.11.2 Constructor & Destructor Documentation

Tree()

`Tree::Tree ( )  [explicit]`

Initialize the Tree in that arrange for social network system:

- users –children--> {user}
- user --> {id,name,posts,followers}
- posts --> {post}
- post --> {body, topics}
- topics --> {topic}
- followers --> {follower}
- follower --> {id}
- not mentioned: doesn't have a child.

Definition at line 41 of file Tree.cpp.

~Tree()

```
Tree::~Tree ( )  [inline]
```

D'tor.

Definition at line 51 of file Tree.h.

3.11.3 Member Function Documentation

getRoot()

```
TreeNode * Tree::getRoot ( )  [inline]
```

Definition at line 56 of file Tree.h.

print()

```
void Tree::print ( ) const  [inline]
```

Definition at line 58 of file Tree.h.

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

- Tree.h
- Tree.cpp

3.12 TreeNode Class Reference

```
#include <TreeNode.h>
```

Public Member Functions

- TreeNode (const TreeNode *parentNode, std::vector< TreeNode * > *children, std::string *value)
  C'tor.
- ~TreeNode ()
  D'tor.
- const TreeNode * getChild (const std::string *value) const
  Returns The child with the assigned value.
- const TreeNode * getParent () const
  Returns the parent of this node.
- bool isChild (const std::string *value) const
  check if the value is for a child or not.
- std::string getValue () const

Friends

- class Tree

3.12.1 Detailed Description

Definition at line 14 of file TreeNode.h.

3.12.2 Constructor & Destructor Documentation

TreeNode()

```
TreeNode::TreeNode (
            const TreeNode * parentNode,
            std::vector< TreeNode * > * children,
            std::string * value )  [inline], [explicit]
```

C'tor.

Parameters

parentNode	
children	
value	

Definition at line 30 of file TreeNode.h.

~TreeNode()

```
TreeNode::~TreeNode ( )  [inline]
```

D'tor.

Definition at line 36 of file TreeNode.h.

3.12.3 Member Function Documentation

getChild()

```
const TreeNode * TreeNode::getChild (
            const std::string * value ) const
```

Returns The child with the assigned value.

Parameters

value	

Returns

child TreeNode, nullptr if not found.

Definition at line 12 of file TreeNode.cpp.

getParent()

```
const TreeNode * TreeNode::getParent ( ) const  [inline]
```

Returns the parent of this node.

Definition at line 55 of file TreeNode.h.

getValue()

```
std::string TreeNode::getValue ( ) const  [inline]
```

Definition at line 65 of file TreeNode.h.

isChild()

```
bool TreeNode::isChild (
            const std::string * value ) const  [inline]
```

check if the value is for a child or not.

Parameters

value	

Returns

true if found, false otherwise.

Definition at line 62 of file TreeNode.h.

3.12.4 Friends And Related Symbol Documentation

Tree

```
friend class Tree  [friend]
```

Definition at line 16 of file TreeNode.h.

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

- TreeNode.h
- TreeNode.cpp

4 File Documentation

4.1 ClearClosingTagsComp.cpp File Reference

The source file of ClearClosingTagsComp class.

```
#include "pch.h"
#include "ClearClosingTagsComp.h"
```

Include dependency graph for ClearClosingTagsComp.cpp:

4.2 ClearClosingTagsComp.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00026 #include "pch.h"
00027 #include "ClearClosingTagsComp.h"
00028
00029 std::string* ClearClosingTagsComp::compress()
00030 {
00031     // to store the result.
00032     std::string* result = new std::string();
00033     //length of the original file.
00034     int length = this->xmlFile->size();
00035
00036     // The max size of the result string is the same of the entered string.
00037     result->reserve(length);
00038
00039     /*
00040     * Loop for all the original string.
00041     * - If the current string is '</'
00042     *       1.skip that tag (increment i till the end of the tag).
00043     *       2.Don't add it to the result string.
00044     * - For other characters, add them to the result.
00045     */
00046     char currentChar = 0;
00047     for (int i = 0; i < length; i++) {
00048         // get current char
00049         currentChar = this->xmlFile->at(i);
00050         if (currentChar == '<' && this->xmlFile->at(i + 1) == '/') {
00051             //skip the closing tag
00052             i = this->xmlFile->find('>', i);
00053             continue;
00054         }
00055         result->append(1, currentChar);
00056     }
00057
00058     // Free the extra allocated memory locations.
00059     result->shrink_to_fit();
00060     return result;
00061 }// compress()
```

4.3 ClearClosingTagsComp.h File Reference

The header file of ClearClosingTagsComp class.

```
#include <string>
```

Include dependency graph for ClearClosingTagsComp.h: This graph shows which files directly or indirectly include this file:

Classes

- class ClearClosingTagsComp

Macros

- #define CLEAR_CLOSING_TAGS_COMP_H

4.3.1 Detailed Description

The header file of ClearClosingTagsComp class.

- This algorithm of compression is based on deleting all the ending tags to reduce the space of the xmlFile.
- It requires knowing what tags comes after other to know how to decompress the file.

Example:

- File before: <tag0><tag1><tag2>d1</tag2><tag2>d2</tag2></tag1></tag0>
- File after: <tag0><tag1><tag2>d1<tag2>d2

@TODO update the file to work with any type of XML data. Use Trees to recored the order of the tags.

Warning

This implementation only works for Social network system, needs an update.

Author

eslam

Date

December 2023

Definition in file ClearClosingTagsComp.h.

4.3.2 Macro Definition Documentation

CLEAR_CLOSING_TAGS_COMP_H

```
#define CLEAR_CLOSING_TAGS_COMP_H
```

Definition at line 28 of file ClearClosingTagsComp.h.

4.4 ClearClosingTagsComp.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00026 #pragma once
00027 #ifndef CLEAR_CLOSING_TAGS_COMP_H
00028 #define CLEAR_CLOSING_TAGS_COMP_H
00029
00030 #include <string>
00031
00032 class ClearClosingTagsComp
00033 {
00034 private:
00035     const std::string* xmlFile;
00036
00037 public:
00051     explicit ClearClosingTagsComp(const std::string* xmlFile) : xmlFile(xmlFile) {}
00052
00064     std::string* compress();
00065 }; // class ClearClosingTagsComp
00066 #endif // !CLEAR_CLOSING_TAGS_COMP_H
```

4.5 ClearClosingTagsComp_unittest.cpp File Reference

Unit test code for ClearClosingTagsComp class.

```
#include "pch.h"
#include "gtest/gtest.h"
#include "ClearClosingTagsComp.h"
```

Include dependency graph for ClearClosingTagsComp_unittest.cpp:

4.5.1 Detailed Description

Unit test code for ClearClosingTagsComp class.

Author

eslam

Date

December 2023

Definition in file ClearClosingTagsComp_unittest.cpp.

4.6 ClearClosingTagsComp_unittest.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00009 #include "pch.h"
00010 #include "gtest/gtest.h"
00011 #include "ClearClosingTagsComp.h"
00012
00013 namespace {
00014     class ClearClosingTagsCompTest : public::testing::Test {
00015     public:
00016         ClearClosingTagsComp* c;
00017         std::string* input;
00018         std::string* output;
00019     protected:
00020         void SetUp() override {
```

```
00021             input = new std::string(R"(<users><user><id>1</id><name>Ahmed
      Ali</name><posts><post><body>Lorem ipsum dolor sit ametffsjkn &alt;
      </body><topics><topic>economy</topic></topics></post></posts><followers><follower><id>2</id></follower></followers></use
00022
00023             output = new std::string(R"(<users><user><id>1<name>Ahmed Ali<posts><post><body>Lorem
      ipsum dolor sit ametffsjkn &alt; <topics><topic>economy<followers><follower><id>2)");
00024
00025             c = new ClearClosingTagsComp(input);
00026         }
00027
00028         void TearDown() override {
00029             delete c;
00030             c = nullptr;
00031             delete input;
00032             input = nullptr;
00033             delete output;
00034             output = nullptr;
00035         }
00036     };
00037
00038     TEST_F(ClearClosingTagsCompTest, compressTest) {
00039         std::string* s = c->compress();
00040         EXPECT_EQ(*s, *output);
00041
00042         delete s;
00043         s = nullptr;
00044     }
00045 }// namespace
```

4.7 ClearClosingTagsDec.cpp File Reference

The source file of ClearClosingTagsDec class.

```
#include "pch.h"
#include "ClearClosingTagsDec.h"
```

Include dependency graph for ClearClosingTagsDec.cpp:

4.7.1 Detailed Description

The source file of ClearClosingTagsDec class.

- The decompression algorithm is based on returning the removed tags from compression.
- It requires knowing what tags comes after other to know how to decompress the file. Example:
- File before: <tag0><tag1><tag2>d1<tag2>d2
- File after: <tag0><tag1><tag2>d1</tag2><tag2>d2</tag2></tag1></tag0>

@TODO update the file to work with any type of XML data. Use Trees to recored the order of the tags.

Warning

This implementation only works for Social network system, needs an update.

Author

eslam

Date

December 2023

Definition in file ClearClosingTagsDec.cpp.

4.8 ClearClosingTagsDec.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00025 #include "pch.h"
00026 #include "ClearClosingTagsDec.h"
00027
00028 std::string ClearClosingTagsDec::getTag(int& tagPosition) const
00029 {
00030     //get the '<'
00031     char currentChar = this->xmlFile->at(tagPosition);
00032     //append '<' to the result string.
00033     std::string tag = std::string(1, currentChar);
00034     //loop and add all chars to the result until '>'
00035     while (currentChar != '>') {
00036         tagPosition++;
00037         currentChar = this->xmlFile->at(tagPosition);
00038         tag += currentChar;
00039     }
00040     return tag;
00041 }
00042
00043 bool ClearClosingTagsDec::needClosingTag(std::string& tag) const
00044 {
00045     if (tagsStack->empty()) {
00046         tagsStack->push(tagsTree->getRoot());
00047         return false;
00048     }
00049     bool isChild = tagsStack->top()->isChild(&tag.substr(1, tag.length() - 2));
00050     if (isChild) {
00051         tagsStack->push(tagsStack->top()->getChild(&tag.substr(1, tag.length() - 2)));
00052         return false;
00053     }
00054     else {
00055         return true;
00056     }
00057 }
00058
00059 std::string* ClearClosingTagsDec::decompress() const
00060 {
00061     // to store the result.
00062     std::string* result = new std::string();
00063     //length of the original file.
00064     int length = this->xmlFile->size();
00065
00066     // Reserve double the space of the original string.
00067     result->reserve(2 * length);
00068
00069     /*
00070     * Loop for all the original string.
00071     * - If the current string is '<' collect that tag then:
00072     *       1.If the stack is empty, add the tag's node into the stack.
00073     *       2.else check the top tag on the stack
00074     *           a.If the current is a child of the top tag:
00075     *             add the current tag to the stack, and to the result string.
00076     *           b.if it's not pop the op tag and append it to the result
00077     *             as a closing tag.
00078     *             Repeat that till the current tag is a child of the top tag,
00079     *             then do the same as in a.
00080     *       3.At the end of the file add all the remaining tags in the stack as
00081     *         a closing tags in order.
00082     * - For other characters, add them to the result.
00083     */
00084     char currentChar = 0;
00085     for (int i = 0; i < length; i++) {
00086         // get current char
00087         currentChar = this->xmlFile->at(i);
00088         if (currentChar == '<') {
00089             //get the tag
00090             std::string tag = getTag(i);
00091             //check if the prev tag needs a closing tag.
00092             bool addClosingTag = needClosingTag(tag);
00093             //loop and add closing tags for the top of the stacks until addClosingTag == false.
00094             while (addClosingTag) {
00095                 const TreeNode* temp = tagsStack->top();
00096                 //get the closing tag
00097                 std::string value = std::string(temp->getValue());
00098                 std::string closingTag("</");
00099                 closingTag.append(value);
00100                 closingTag.append(">");
00101                 result->append(closingTag);
00102
00103                 tagsStack->pop();
00104
00105                 //check the next top of stack
00106                 addClosingTag = needClosingTag(tag);
```

```
00107             }
00108             //append the new tag.
00109             result->append(tag);
00110         } // if (currentChar == '<')
00111         else {
00112             result->append(1, currentChar);
00113         }
00114     }
00115
00116     //add the remaining tags in the stack
00117     while (!tagsStack->empty()) {
00118         const TreeNode* temp = tagsStack->top();
00119
00120         // get the closing tag and add it to the result
00121         std::string value = std::string(temp->getValue());
00122         std::string closingTag("</");
00123         closingTag.append(value);
00124         closingTag.append(">");
00125
00126         result->append(closingTag);
00127
00128         tagsStack->pop();
00129     }
00130
00131     // Free the extra allocated memory locations.
00132     result->shrink_to_fit();
00133     return result;
00134 }
```

4.9 ClearClosingTagsDec.h File Reference

The header file of ClearClosingTagsDec class.

```
#include "Tree.h"
#include <stack>
```

Include dependency graph for ClearClosingTagsDec.h: This graph shows which files directly or indirectly include this file:

Classes

- class ClearClosingTagsDec

Macros

- #define CLEAR_CLOSING_TAGS_DEC_H

4.9.1 Detailed Description

The header file of ClearClosingTagsDec class.

- The decompression algorithm is based on returning the removed tags from compression.
- It requires knowing what tags comes after other to know how to decompress the file.

Example:

- File before: <tag0><tag1><tag2>d1<tag2>d2
- File after: <tag0><tag1><tag2>d1</tag2><tag2>d2</tag2></tag1></tag0>

@TODO update the file to work with any type of XML data. Use Trees to recored the order of the tags.

Warning

This implementation only works for Social network system, needs an update.

Author

eslam

Date

December 2023

Definition in file ClearClosingTagsDec.h.

4.9.2 Macro Definition Documentation

CLEAR_CLOSING_TAGS_DEC_H

```
#define CLEAR_CLOSING_TAGS_DEC_H
```

Definition at line 27 of file ClearClosingTagsDec.h.

4.10 ClearClosingTagsDec.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00025 #pragma once
00026 #ifndef CLEAR_CLOSING_TAGS_DEC_H
00027 #define CLEAR_CLOSING_TAGS_DEC_H
00028
00029 #include "Tree.h"
00030 #include <stack>
00031
00032 class ClearClosingTagsDec
00033 {
00034 private:
00035     //Holds the known arrange of tags.
00036     Tree* tagsTree;
00037     //To know which tag we are inside.
00038     std::stack<const TreeNode*>* tagsStack;
00039     //The file needed to be decompressed.
00040     const std::string* xmlFile;
00041
00042     //helper methods
00050     std::string getTag(int& tagPosition) const;
00078     bool needClosingTag(std::string& tag) const;
00079
00080 public:
00092     explicit ClearClosingTagsDec(const std::string* xmlFile) : xmlFile(xmlFile),
00093         tagsTree(new Tree()), tagsStack(new std::stack<const TreeNode*>()) {}
00098     ~ClearClosingTagsDec() { delete tagsTree; delete tagsStack; }
00099
00116     std::string* decompress() const;
00117 };
00118
00119 #endif // !CLEAR_CLOSING_TAGS_DEC_H
```

4.11 ClearClosingTagsDec_unittest.cpp File Reference

Unit test code for ClearClosingTagsDec class.

```
#include "pch.h"
#include "gtest/gtest.h"
#include "ClearClosingTagsDec.h"
```

Include dependency graph for ClearClosingTagsDec_unittest.cpp:

4.11.1 Detailed Description

Unit test code for ClearClosingTagsDec class.

Author

eslam

Date

December 2023

Definition in file ClearClosingTagsDec_unittest.cpp.

4.12 ClearClosingTagsDec_unittest.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00009 #include "pch.h"
00010 #include "gtest/gtest.h"
00011 #include "ClearClosingTagsDec.h"
00012
00013 namespace {
00014     class ClearClosingTagsDecTest : public::testing::Test {
00015     public:
00016         ClearClosingTagsDec* c;
00017         std::string* input;
00018         std::string* output;
00019     protected:
00020         void SetUp() override {
00021             output = new std::string(R"(<users><user><id>1</id><name>Ahmed
      Ali</name><posts><post><body>Lorem ipsum dolor sit ametffsjkn &alt;
      </body><topics><topic>economy</topic></topics></post></posts><followers><follower><id>2</id></follower></followers></use
00022
00023             input = new std::string(R"(<users><user><id>1<name>Ahmed Ali<posts><post><body>Lorem ipsum
      dolor sit ametffsjkn &alt; <topics><topic>economy<followers><follower><id>2)");
00024
00025             c = new ClearClosingTagsDec(input);
00026         }
00027
00028         void TearDown() override {
00029             delete c;
00030             c = nullptr;
00031             delete input;
00032             input = nullptr;
00033             delete output;
00034             output = nullptr;
00035         }
00036     };
00037
00038     TEST_F(ClearClosingTagsDecTest, compressTest) {
00039         std::string* s = c->decompress();
00040         EXPECT_EQ(*s, *output);
00041
00042         delete s;
00043         s = nullptr;
00044     }
00045 } // namespace
```

4.13 Tree.cpp File Reference

A simple Tree DS implementation.

```
#include "pch.h"
#include "Tree.h"
```

Include dependency graph for Tree.cpp:

4.13.1 Detailed Description

A simple Tree DS implementation.

This tree is for arranging social network system tags. it will be in this order:

- users –children--> {user}
- user --> {id,name,posts,followers}
- posts --> {post}
- post --> {body, topics}
- topics --> {topic}
- followers --> {follower}
- follower --> {id}
- not mentioned: doesn't have a child.

Author

eslam

Date

December 2023

Definition in file Tree.cpp.

4.14 Tree.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00021 #include "pch.h"
00022 #include "Tree.h"
00023
00024 void Tree::printTreeNode(const TreeNode* node, int depth) const
00025 {
00026     if (node == nullptr) {
00027         return;
00028     }
00029
00030     for (int i = 0; i < depth; ++i) {
00031         std::cout « "   ";
00032     }
00033
00034     std::cout « "|-- " « *node->value « std::endl;
00035
00036     for (const TreeNode* child : *node->children) {
00037         printTreeNode(child, depth + 1);
00038     }
00039 }
00040
00041 Tree::Tree() {
00042     // Creating nodes for the social network system tags
00043     root = new TreeNode(nullptr, new std::vector<TreeNode*>(), new std::string("users"));
00044
00045     // Add child nodes for 'users'
00046     root->children->push_back(
00047         new TreeNode(root, new std::vector<TreeNode*>(), new std::string("user"))
00048     );
00049
00050     // Add child nodes for 'user'
```

```
00051     (*root->children)[0]->children->push_back(
00052         new TreeNode((*root->children)[0], new std::vector<TreeNode*>(), new std::string("id"))
00053     );
00054     (*root->children)[0]->children->push_back(
00055         new TreeNode((*root->children)[0], new std::vector<TreeNode*>(), new std::string("name"))
00056     );
00057     (*root->children)[0]->children->push_back(
00058         new TreeNode((*root->children)[0], new std::vector<TreeNode*>(), new std::string("posts"))
00059     );
00060     (*root->children)[0]->children->push_back(
00061         new TreeNode((*root->children)[0], new std::vector<TreeNode*>(), new std::string("followers"))
00062     );
00063
00064     // Add child nodes for 'posts'
00065     (*(*root->children)[0]->children)[2]->children->push_back(
00066         new TreeNode((*(*root->children)[0]->children)[2], new std::vector<TreeNode*>(), new
      std::string("post"))
00067     );
00068
00069     // Add child nodes for 'post'
00070     (*(*(*root->children)[0]->children)[2]->children)[0]->children->push_back(
00071         new TreeNode((*(*(*root->children)[0]->children)[2]->children)[0], new
      std::vector<TreeNode*>(), new std::string("body"))
00072     );
00073
00074     (*(*(*root->children)[0]->children)[2]->children)[0]->children->push_back(
00075         new TreeNode((*(*(*root->children)[0]->children)[2]->children)[0], new
      std::vector<TreeNode*>(), new std::string("topics"))
00076     );
00077
00078     // Add child nodes for 'topics'
00079     (*(*(*(*root->children)[0]->children)[2]->children)[0]->children)[1]->children->push_back(
00080         new TreeNode((*(*(*(*root->children)[0]->children)[2]->children)[0]->children)[1], new
      std::vector<TreeNode*>(), new std::string("topic"))
00081     );
00082
00083     // Add child nodes for 'followers'
00084     (*(*root->children)[0]->children)[3]->children->push_back(
00085         new TreeNode((*(*root->children)[0]->children)[3], new std::vector<TreeNode*>(), new
      std::string("follower"))
00086     );
00087
00088     // Add child nodes for 'follower'
00089     (*(*(*root->children)[0]->children)[3]->children)[0]->children->push_back(
00090         new TreeNode((*(*(*root->children)[0]->children)[3]->children)[0], new
      std::vector<TreeNode*>(), new std::string("id"))
00091     );
00092 }
```

4.15 Tree.h File Reference

A simple Tree DS implementation.

```
#include "TreeNode.h"
#include <iostream>
```

Include dependency graph for Tree.h: This graph shows which files directly or indirectly include this file:

Classes

- class Tree

Macros

- #define TREE_H

4.15.1 Detailed Description

A simple Tree DS implementation.

This tree is for arranging social network system tags. it will be in this order:

- users –children--> {user}
- user --> {id,name,posts,followers}
- posts --> {post}
- post --> {body, topics}
- topics --> {topic}
- followers --> {follower}
- follower --> {id}
- not mentioned: doesn't have a child.

Author

eslam

Date

December 2023

Definition in file Tree.h.

4.15.2 Macro Definition Documentation

TREE_H

```
#define TREE_H
```

Definition at line 23 of file Tree.h.

4.16 Tree.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00021 #pragma once
00022 #ifndef TREE_H
00023 #define TREE_H
00024
00025 #include "TreeNode.h"
00026 #include <iostream>
00027
00028 class Tree
00029 {
00030 private:
00031     TreeNode* root;
00032     //for debugging
00033     void printTreeNode(const TreeNode* node, int depth) const;
00034 public:
00046     explicit Tree();
00051     ~Tree() {
00052         delete root;
00053     }
00054
00055     //getter
00056     TreeNode* getRoot() { return root; }
00057     //for debugging
00058     void print() const {
00059         printTreeNode(root, 0);
00060     }
00061 };
00062
00063 #endif // !TREE_H
```

4.17 TreeNode.cpp File Reference

A simple Tree Node for the tree data structure.

```
#include "pch.h"
#include "TreeNode.h"
```

Include dependency graph for TreeNode.cpp:

4.17.1 Detailed Description

A simple Tree Node for the tree data structure.

Author

eslam

Date

December 2023

Definition in file TreeNode.cpp.

4.18 TreeNode.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00009 #include "pch.h"
00010 #include "TreeNode.h"
00011
00012 const TreeNode* TreeNode::getChild(const std::string* value) const
00013 {
00014     //loop for all the vector until finding the needed child with the needed value.
00015     for (TreeNode* child : *children) {
00016         if (*child->value == *value) {
00017             return child;
00018         }
00019     }
00020     //if not found.
00021     return nullptr;
00022 }
```

4.19 TreeNode.h File Reference

A simple Tree Node for the tree data structure.

```
#include <string>
#include <vector>
```

Include dependency graph for TreeNode.h: This graph shows which files directly or indirectly include this file:

Classes

- class TreeNode

Macros

- #define TREE_NODE_H

4.19.1 Detailed Description

A simple Tree Node for the tree data structure.

Author

eslam

Date

December 2023

Definition in file TreeNode.h.

4.19.2 Macro Definition Documentation

TREE_NODE_H

```
#define TREE_NODE_H
```

Definition at line 10 of file TreeNode.h.

4.20 TreeNode.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00008 #pragma once
00009 #ifndef TREE_NODE_H
00010 #define TREE_NODE_H
00011
00012 #include <string>
00013 #include <vector>
00014 class TreeNode
00015 {
00016     friend class Tree;
00017 private:
00018     const TreeNode* parentNode;
00019     std::vector<TreeNode*>* children;
00020     std::string* value;
00021
00022 public:
00030     explicit TreeNode(const TreeNode* parentNode, std::vector<TreeNode*>* children, std::string*
      value)
00031         : parentNode(parentNode), children(children), value(value) {}
00032
00036     ~TreeNode() {
00037         for (TreeNode* child : *children) {
00038             delete child;
00039         }
00040         delete children;
00041         delete value;
00042     }
00043
00044     //methods.
00051     const TreeNode* getChild(const std::string* value) const;
00055     const TreeNode* getParent()const { return this->parentNode; }
00062     bool isChild(const std::string* value) const { return getChild(value) != nullptr; }
00063
00064     //getter
00065     std::string getValue() const { return *value; }
00066 };
00067 #endif // !TREE_NODE_H
```

4.21 HuffmanComp.cpp File Reference

Source file for Huffman Compression algorithm.

```
#include "pch.h"
#include "HuffmanComp.h"
```

Include dependency graph for HuffmanComp.cpp:

4.21.1 Detailed Description

Source file for Huffman Compression algorithm.

This file implements the HuffmanComp class, which implements Huffman compression. It utilizes HuffmanTree for encoding and generates compressed output based on the Huffman tree's encoded representation.

Author

eslam

Date

December 2023

Definition in file HuffmanComp.cpp.

4.22 HuffmanComp.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00013 #include "pch.h"
00014 #include "HuffmanComp.h"
00015
00016 std::string* HuffmanComp::compress()
00017 {
00018     // add the encoding table in the 1st line of the compressed file.
00019     std::string* compressedString = new std::string(tree->getEncodedTree());
00020
00021     // Encode each character in the file content using the Huffman tree
00022     std::string bits = "";
00023     for (char c : *file) {
00024         std::vector<bool> encoding = tree->getEncodingFromChar(c);
00025         for (bool bit : encoding) {
00026             bits += (bit ? '1' : '0');
00027         }
00028     }
00029
00030     // Add the total number of bits so it can be retrieved at decompression.
00031     *compressedString += std::to_string(bits.size()) + "\n";
00032
00033     //add the bits
00034     *compressedString += bits;
00035
00036     return compressedString;
00037 }
```

4.23 HuffmanComp.h File Reference

Header file for Huffman Compression algorithm.

```
#include "HuffmanTree.h"
```

Include dependency graph for HuffmanComp.h: This graph shows which files directly or indirectly include this file:

Classes

- class HuffmanComp

Macros

- #define HUFFMAN_COMP_H

4.23.1 Detailed Description

Header file for Huffman Compression algorithm.

This file defines the HuffmanComp class, which implements Huffman compression. It utilizes HuffmanTree for encoding and generates compressed output based on the Huffman tree's encoded representation.

Author

eslam

Date

December 2023

Definition in file HuffmanComp.h.

4.23.2 Macro Definition Documentation

HUFFMAN_COMP_H

```
#define HUFFMAN_COMP_H
```

Definition at line 16 of file HuffmanComp.h.

4.24 HuffmanComp.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00014 #pragma once
00015 #ifndef HUFFMAN_COMP_H
00016 #define HUFFMAN_COMP_H
00017 #include "HuffmanTree.h"
00018
00019 class HuffmanComp
00020 {
00021 private:
00022     // tree holds the encoding.
00023     HuffmanTree* tree;
00024     //the file to compress.
00025     std::string* file;
00026
00027 public:
00036     explicit HuffmanComp(std::string* fileC) : file(fileC) {
00037         this->tree = HuffmanTree::generateTreeFromText(*fileC);
00038     }
00039
00043     ~HuffmanComp() { delete tree; delete file; }
00044
00055     std::string* compress();
00056 };
00057
00058 #endif // !HUFFMAN_COMP_H
```

4.25 HuffmanDec.cpp File Reference

Source file for Huffman Decompression algorithm.

```
#include "pch.h"
#include "HuffmanDec.h"
```

Include dependency graph for HuffmanDec.cpp:

4.25.1 Detailed Description

Source file for Huffman Decompression algorithm.

This file implements the HuffmanDec class, which implements Huffman decompression. It provides functionality to decompress data using a Huffman tree's encoded representation.

Author

eslam

Date

December 2023

Definition in file HuffmanDec.cpp.

4.26 HuffmanDec.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00011 #include "pch.h"
00012 #include "HuffmanDec.h"
00013
00014 HuffmanDec::HuffmanDec(std::string* file) : file(file) {
00015     if (!file->empty()) {
00016         // Find the end of the encoded tree.
00017         int endPos = file->find(")\n");
00018
00019         //get the number of bits.
00020         int endNumPos = file->find("\n", endPos);
00021         std::string bitsNum = file->substr(endPos + 2, endNumPos - endPos - 3);
00022         this->bitsLength = std::stoi(bitsNum);
00023
00024         //build the tree.
00025         if (endPos != std::string::npos) {
00026             // Exclude the ")\n" at the end and the first "("
00027             std::string encodedTree = file->substr(1, endPos - 1);
00028             tree = HuffmanTree::rebuildTree(encodedTree);
00029         }
00030     }
00031     else {
00032         throw std::runtime_error("Defected file.");
00033     }
00034 }
00035
00036 std::string* HuffmanDec::decompress() {
00037     if (!tree || !file || file->empty()) {
00038         // Cannot decompress without a tree or compressed data
00039         return nullptr;
00040     }
00041
00042     // Find the end of the encoded tree
00043     int startPos = file->find(")\n") + 2;
00044     // Find the end of the bits number.
00045     startPos = file->find("\n", startPos) + 1;
00046
```

```
00047     //get the bits after the number of bits line, loop only for the requited bits
00048     //(total as the number of bits in the second line.) and skip the extra added bits in the last.
00049     if (startPos != std::string::npos && startPos < file->size()) {
00050         // Extract compressed bits
00051         std::string compressedBits = file->substr(startPos);
00052 
00053         std::string* decompressedString = new std::string();
00054         std::vector<bool> encoding;
00055 
00056         // Loop through the bits and reconstruct the original string
00057         for (int i = 0; i < this->bitsLength; i++) {
00058             //get the current bit.
00059             char bit = compressedBits.at(i);
00060             //push to the encoding vector as a boolean value.
00061             encoding.push_back(bit == '1' ? true : false);
00062             //- Get the char from its encoding using the tree, if the returned was void, that
00063             //  means the length of the encoding vector didn't reach any leaf, so add the next
00064             //  bit and try again.
00065             //- If it wasn't a void char, than add the char into the result string, and empty
00066             //  the encoding vector.
00067             char character = tree->getCharFromEncoding(encoding);
00068             if (character != '\0') {
00069                 *decompressedString += character;
00070                 encoding.clear();
00071             }
00072         }
00073         //the result after decompressing.
00074         return decompressedString;
00075     }
00076 
00077     // Unable to decompress properly
00078     return nullptr;
00079 }
```

4.27 HuffmanDec.h File Reference

Header file for Huffman Decompression algorithm.

`#include "HuffmanTree.h"`
Include dependency graph for HuffmanDec.h: This graph shows which files directly or indirectly include this file:

Classes

- class HuffmanDec

Macros

- #define HUFFMAN_DEC_H

4.27.1 Detailed Description

Header file for Huffman Decompression algorithm.

This file defines the HuffmanDec class, which implements Huffman decompression. It provides functionality to decompress data using a Huffman tree's encoded representation.

Author

eslam

Date

December 2023

Definition in file HuffmanDec.h.

4.27.2 Macro Definition Documentation

HUFFMAN_DEC_H

```
#define HUFFMAN_DEC_H
```

Definition at line 14 of file HuffmanDec.h.

4.28 HuffmanDec.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00012 #pragma once
00013 #ifndef HUFFMAN_DEC_H
00014 #define HUFFMAN_DEC_H
00015 #include "HuffmanTree.h"
00016
00017 class HuffmanDec
00018 {
00019 private:
00020     // tree holds the encoding.
00021     HuffmanTree* tree;
00022     // the file to decompress.
00023     std::string* file;
00024     // number of bits in the compressed file
00025     long long bitsLength;
00026
00027 public:
00036     explicit HuffmanDec(std::string* file);
00037
00042     ~HuffmanDec() {
00043         delete tree;
00044         if (file != nullptr) {
00045             delete file;
00046         }
00047     }
00048
00057     std::string* decompress();
00058 };
00059
00060 #endif // !HUFFMAN_DEC_H
```

4.29 HuffmanTree.cpp File Reference

Source file implementing the HuffmanTree class responsible for managing the Huffman tree construction.

```
#include "pch.h"
#include "HuffmanTree.h"
```

Include dependency graph for HuffmanTree.cpp:

4.29.1 Detailed Description

Source file implementing the HuffmanTree class responsible for managing the Huffman tree construction.

This file contains the declaration of the HuffmanTree class, which handles the construction of the Huffman tree based on character frequencies derived from input text. The class provides functionalities to generate the tree, get its encoded representation, and rebuild the tree from its encoded form.

To use the tree, use the suitable static function to create the tree, then use the methods of encoding or decoding according to the task.

Author

eslam

Date

December 2023

Definition in file HuffmanTree.cpp.

4.30 HuffmanTree.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00016 #include "pch.h"
00017 #include "HuffmanTree.h"
00018
00019 std::vector<int> HuffmanTree::calculateFrequencies(const std::string& text)
00020 {
00021     // initialize a vector with 256 position all with zeros as a value.
00022     // each position represents a char in the ASCII code.
00023     std::vector<int> frequencies(256, 0);
00024
00025     int length = text.length();
00026
00027     // Loop for all the chars in the text, and increment its position.
00028     for (int i = 0; i < length; i++) {
00029         char c = text.at(i);
00030         frequencies[static_cast<unsigned char>(c)]++;
00031     }
00032     // return the freq vector.
00033     return frequencies;
00034 }
00035
00036 bool HuffmanTree::findCharEncodingHelper(HuffmanTreeNode* node, char c, std::vector<bool>&
      currentEncoding,
00037     std::vector<bool>& encoding) {
00038     if (!node) {
00039         // If the node is null, the character is not found in this branch.
00040         return false;
00041     }
00042     if (node->c == c) {
00043         // Found the character's encoding, updating 'encoding'.
00044         encoding = currentEncoding;
00045         return true;
00046     }
00047
00048     // Appending 'false' to the encoding path.
00049     currentEncoding.push_back(false);
00050     if (findCharEncodingHelper(node->leftChild, c, currentEncoding, encoding)) {
00051         // If 'c' is found in the left subtree, return true.
00052         return true;
00053     }
00054     // If 'c' is not in the left subtree, backtrack.
00055     currentEncoding.pop_back();
00056
00057     // Appending 'true' to the encoding path.
00058     currentEncoding.push_back(true);
00059     if (findCharEncodingHelper(node->rightChild, c, currentEncoding, encoding)) {
00060         // If 'c' is found in the right subtree, return true.
00061         return true;
00062     }
00063     // If 'c' is not in the right subtree, backtrack.
00064     currentEncoding.pop_back();
00065
00066     // 'c' is not found in this branch of the tree.
00067     return false;
00068 }
00069
00070 char HuffmanTree::findCharFromEncodingHelper(HuffmanTreeNode* node, const std::vector<bool>& encoding,
      int index)
00071 {
00072     // Return null character if node is null
00073     if (!node) {
00074         return '\0';
00075     }
00076     // Return the character when the end of encoding is reached
00077     if (index == encoding.size()) {
00078         return node->c;
00079     }
00080
00081     // Traverse left or right based on the bit value in encoding
00082     if (encoding[index] == false) {
00083         return findCharFromEncodingHelper(node->leftChild, encoding, index + 1);
00084     }
00085     else {
00086         return findCharFromEncodingHelper(node->rightChild, encoding, index + 1);
00087     }
00088 }
00089
00090 HuffmanTree* HuffmanTree::generateTreeFromText(const std::string& text)
00091 {
00092     // get the frequency of each char in the text.
00093     std::vector<int> frequencies = calculateFrequencies(text);
00094     // To sort the char's freq from the least to the highest.
00095     // The queue will use the overridden operator() method in the HuffmanTreeNode for
```

```
00096     // the comparing.
00097     std::priority_queue<HuffmanTreeNode*, std::vector<HuffmanTreeNode*>, HuffmanTreeNode> minHeap;
00098
00099     //loop for all char frequencies, create nodes for each char that occurred in the text
00100     // then, add nodes to the minHeap.
00101     // The frequency of the node will be provided from the freq vector, and the char is
00102     // the position of the freq vector.
00103     for (int i = 0; i < 256; ++i) {
00104         if (frequencies[i] > 0) {
00105             minHeap.push(new HuffmanTreeNode(frequencies[i], static_cast<char>(i)));
00106         }
00107     }
00108     /*
00109     * - Loop until only the root is left alone in the minHeap
00110     * - The 2 nodes with the lowest freq will be combined into one node,
00111     *       - The smallest will be the right child of the parent node.
00112     *       - The other will be the left child.
00113     * - The parent node will be constructed with the sum of freq of the 2 nodes, and char '\0'.
00114     * - remove the 2 children and push the parent in the heap.
00115     */
00116     while (minHeap.size() > 1) {
00117         HuffmanTreeNode* left = minHeap.top();
00118         minHeap.pop();
00119         HuffmanTreeNode* right = minHeap.top();
00120         minHeap.pop();
00121
00122         HuffmanTreeNode* newNode = new HuffmanTreeNode(left->freq + right->freq, '\0', left, right);
00123         minHeap.push(newNode);
00124     }
00125     //The left node in the heap will be the root of the tree.
00126
00127     return new HuffmanTree(minHeap.top());
00128 }
00129
00130
00131 HuffmanTree* HuffmanTree::rebuildTree(const std::string& encodedTree)
00132 {
00133     //length of the encoded tree;
00134     int length = encodedTree.length();
00135
00136     // Construct the frequency vector from the encoded tree.
00137     std::vector<int> frequencies(256, 0);
00138
00139     for (int i = 0; i < length; i++) {
00140         char currentChar = encodedTree.at(i);
00141         if (currentChar == '(') {
00142             //get the char
00143             i++;
00144             char c = encodedTree.at(i);
00145             i += 2;
00146             //get the freq number
00147             std::string freq = "";
00148             while (encodedTree.at(i) != ')') {
00149                 freq += encodedTree.at(i);
00150                 i++;
00151             }
00152             //add the freq to the freq vector.
00153             int f = std::stoi(freq);
00154             frequencies[static_cast<unsigned char>(c)] = f;
00155         }
00156         else {
00157             throw std::runtime_error("Invalid Tree Encode");
00158         }
00159     }
00160
00161     // After constructing the frequency vector, if we iterate it in the same way as in
00162     // generateTreeFromText(), we can get the same tree.
00163
00164     // To sort the char's freq from the least to the highest.
00165     std::priority_queue<HuffmanTreeNode*, std::vector<HuffmanTreeNode*>, HuffmanTreeNode> minHeap;
00166
00167     //loop for all char frequencies, create nodes for each char that occurred in the text
00168     // then, add nodes to the minHeap.
00169     // The frequency of the node will be provided from the freq vector, and the char is
00170     // the position of the freq vector.
00171     for (int i = 0; i < 256; ++i) {
00172         if (frequencies[i] > 0) {
00173             minHeap.push(new HuffmanTreeNode(frequencies[i], static_cast<char>(i)));
00174         }
00175     }
00176     /*
00177     * - Loop until only the root is left alone in the minHeap
00178     * - The 2 nodes with the lowest freq will be combined into one node,
00179     *       - The smallest will be the right child of the parent node.
00180     *       - The other will be the left child.
00181     * - The parent node will be constructed with the sum of freq of the 2 nodes, and char '\0'.
00182     * - remove the 2 children and push the parent in the heap.
```

```
00183     */
00184     while (minHeap.size() > 1) {
00185         HuffmanTreeNode* left = minHeap.top();
00186         minHeap.pop();
00187         HuffmanTreeNode* right = minHeap.top();
00188         minHeap.pop();
00189 
00190         HuffmanTreeNode* newNode = new HuffmanTreeNode(left->freq + right->freq, '\0', left, right);
00191         minHeap.push(newNode);
00192     }
00193     //The left node in the heap will be the root of the tree.
00194 
00195     return new HuffmanTree(minHeap.top());
00196 }
00197 
00198 std::string HuffmanTree::getEncodedTree()
00199 {
00200     // If the tree is empty, return an empty string
00201     if (!root) {
00202         return "";
00203     }
00204     // Vector to get all the leaf nodes.
00205     std::vector<HuffmanTreeNode*> leafNodes;
00206     // Queue to hold nodes for traversal.
00207     std::queue<HuffmanTreeNode*> nodeQueue;
00208     // Add the root node to start traversal
00209     nodeQueue.push(root);
00210 
00211     // Traverse the tree to collect all leaf nodes (breadth-first search (BFS) traversal).
00212     while (!nodeQueue.empty()) {
00213         // Dequeue the front node for exploration
00214         HuffmanTreeNode* current = nodeQueue.front();
00215         nodeQueue.pop();
00216 
00217         if (current) {
00218             // Check if the current node is a leaf node
00219             if (!current->leftChild && !current->rightChild) {
00220                 // Collect leaf nodes
00221                 leafNodes.push_back(current);
00222             }
00223             else {
00224                 // If not a leaf node, enqueue its non-null children for further exploration
00225                 if (current->leftChild) {
00226                     nodeQueue.push(current->leftChild);
00227                 }
00228                 if (current->rightChild) {
00229                     nodeQueue.push(current->rightChild);
00230                 }
00231             }
00232         }
00233         // Continue until all nodes are explored
00234     }
00235 
00236     // Construct the encoded tree string from the leaf nodes vector
00237     std::string encodedTree = "(";
00238     for (const HuffmanTreeNode* leaf : leafNodes) {
00239         if (leaf->c != '\0') {
00240             encodedTree += '(';
00241             //Add char.
00242             encodedTree += leaf->c;
00243             encodedTree += ',';
00244 
00245             //add freq
00246             encodedTree += std::to_string(leaf->freq);
00247             encodedTree += ')';
00248         }
00249     }
00250     encodedTree += ")\n";
00251     return encodedTree;
00252 }
00253 
00254 std::vector<bool> HuffmanTree::getEncodingFromChar(char c)
00255 {
00256     // Vector to store the character's encoding
00257     std::vector<bool> encoding;
00258     // Return an empty vector if the tree is empty
00259     if (!root) {
00260         return encoding;
00261     }
00262 
00263     // Temporary vector to store the current path
00264     std::vector<bool> currentEncoding;
00265     // Call helper function to find encoding
00266     findCharEncodingHelper(root, c, currentEncoding, encoding);
00267 
00268     return encoding;
00269 }
```

```
00270
00271 char HuffmanTree::getCharFromEncoding(std::vector<bool> encoding)
00272 {
00273     // Return null character if tree is empty or encoding is empty
00274     if (!root || encoding.empty()) {
00275         return '\0';
00276     }
00277
00278     return findCharFromEncodingHelper(root, encoding, 0);
00279 }
```

4.31 HuffmanTree.h File Reference

Header file defining the HuffmanTree class responsible for managing the Huffman tree construction.

```
#include "HuffmanTreeNode.h"
#include <vector>
#include <queue>
#include <algorithm>
```

Include dependency graph for HuffmanTree.h: This graph shows which files directly or indirectly include this file:

Classes

- class HuffmanTree

Macros

- #define HUFFMAN_TREE_H

4.31.1 Detailed Description

Header file defining the HuffmanTree class responsible for managing the Huffman tree construction.

This file contains the declaration of the HuffmanTree class, which handles the construction of the Huffman tree based on character frequencies derived from input text. The class provides functionalities to generate the tree, get its encoded representation, and rebuild the tree from its encoded form.

To use the tree, use the suitable static function to create the tree, then use the methods of encoding or decoding according to the task.

Author

eslam

Date

December 2023

Definition in file HuffmanTree.h.

4.31.2 Macro Definition Documentation

HUFFMAN_TREE_H

```
#define HUFFMAN_TREE_H
```

Definition at line 17 of file HuffmanTree.h.

4.32 HuffmanTree.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00015 #pragma once
00016 #ifndef HUFFMAN_TREE_H
00017 #define HUFFMAN_TREE_H
00018 #include "HuffmanTreeNode.h"
00019 #include <vector>
00020 #include <queue>
00021 #include <algorithm>
00022
00023 class HuffmanTree
00024 {
00025 private:
00026     HuffmanTreeNode* root;
00027
00028     // Helper methods.
00040     static std::vector<int> calculateFrequencies(const std::string& text);
00041
00058     bool findCharEncodingHelper(HuffmanTreeNode* node, char c,
00059         std::vector<bool>& currentEncoding, std::vector<bool>& encoding);
00060
00073     char findCharFromEncodingHelper(HuffmanTreeNode* node, const std::vector<bool>& encoding, int
      index);
00074
00075 public:
00076
00082     explicit HuffmanTree(HuffmanTreeNode* root) : root(root) {}
00087     ~HuffmanTree() { delete root; }
00088
00089     //Static methods for building the tree
00096     static HuffmanTree* generateTreeFromText(const std::string& text);
00097
00109     static HuffmanTree* rebuildTree(const std::string& encodedTree);
00110
00111     //methods
00112     //for encoding
00122     std::string getEncodedTree();
00123
00134     std::vector<bool> getEncodingFromChar(char c);
00135
00136     //for decoding
00150     char getCharFromEncoding(std::vector<bool> encoding);
00151 };
00152 #endif // !HUFFMAN_TREE_H
```

4.33 HuffmanTree_unittest.cpp File Reference

A quick unit test for the HuffmanTree using gtest framework.

```
#include "pch.h"
#include "gtest/gtest.h"
#include "HuffmanTree.h"
```

Include dependency graph for HuffmanTree_unittest.cpp:

4.33.1 Detailed Description

A quick unit test for the HuffmanTree using gtest framework.

Author

eslam

Date

December 2023

Definition in file HuffmanTree_unittest.cpp.

4.34 HuffmanTree_unittest.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00009 #include "pch.h"
00010 #include "gtest/gtest.h"
00011 #include "HuffmanTree.h"
00012
00013 namespace {
00014     class HuffmanTreeTest : public ::testing::Test {
00015     protected:
00016         // Set up the test fixture
00017         void SetUp() override {
00018             // Create a test string
00019             testString = "This is a test string for Huffman Tree implementation.";
00020             // Create a HuffmanTree object from the test string
00021             tree = HuffmanTree::generateTreeFromText(testString);
00022         }
00023
00024         // Tear down the test fixture
00025         void TearDown() override {
00026             delete tree;
00027             tree = nullptr;
00028         }
00029
00030         // Variables for testing
00031         HuffmanTree* tree;
00032         std::string testString;
00033     };
00034
00035     // Test case for checking generation of Huffman Tree from text
00036     TEST_F(HuffmanTreeTest, GenerateTreeFromText) {
00037         // Check if tree is not null
00038         EXPECT_NE(tree, nullptr);
00039     }
00040
00041     // Test case for checking encoding retrieval for a character
00042     TEST_F(HuffmanTreeTest, GetEncodingFromChar) {
00043         char testChar = 's'; // Character to test encoding
00044         std::vector<bool> encoding = tree->getEncodingFromChar(testChar);
00045         // Check if encoding is not empty
00046         EXPECT_FALSE(encoding.empty());
00047     }
00048
00049     // Test case for checking decoding of character from encoding
00050     TEST_F(HuffmanTreeTest, GetCharFromEncoding) {
00051         char testChar = ' '; // Character to test decoding
00052         std::vector<bool> encoding = tree->getEncodingFromChar(testChar);
00053         char decodedChar = tree->getCharFromEncoding(encoding);
00054         // Check if decoded character matches the original character
00055         EXPECT_EQ(decodedChar, testChar);
00056     }
00057
00058     // Test case for checking rebuilding tree from encoded string
00059     TEST_F(HuffmanTreeTest, RebuildTreeFromEncodedString) {
00060         std::string encodedTree = tree->getEncodedTree();
00061         int endPos = encodedTree.find(")\n"); // Find the end of the encoded tree
00062         encodedTree = encodedTree.substr(1, endPos - 1); // Exclude the ")\n" at the end
00063
00064         HuffmanTree* rebuiltTree = HuffmanTree::rebuildTree(encodedTree);
00065         // Check if rebuilt tree is not null
00066         EXPECT_NE(rebuiltTree, nullptr);
00067         delete rebuiltTree;
00068         rebuiltTree = nullptr;
00069     }
00070 }// namespace
```

4.35 HuffmanTreeNode.h File Reference

A Tree Node created for the Huffman Encoding Tree.

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

Classes

- class HuffmanTreeNode

Macros

- #define HUFFMAN_TREE_NODE_H

4.35.1 Detailed Description

A Tree Node created for the Huffman Encoding Tree.

Author

eslam

Date

December 2023

Definition in file HuffmanTreeNode.h.

4.35.2 Macro Definition Documentation

HUFFMAN_TREE_NODE_H

```
#define HUFFMAN_TREE_NODE_H
```

Definition at line 11 of file HuffmanTreeNode.h.

4.36 HuffmanTreeNode.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00009 #pragma once
00010 #ifndef HUFFMAN_TREE_NODE_H
00011 #define  HUFFMAN_TREE_NODE_H
00012
00013 class HuffmanTreeNode
00014 {
00015     friend class HuffmanTree;
00016 public:
00017     //freq of the node.
00018     int freq;
00019     //
00020     // Holds the character associated with the node.
00021     // Have a value only for the leaf Node, other wise the char will be '\0'.
00022     //
00023     char c;
00024     //children of the node, nullptr if it was a leaf.
00025
00026     HuffmanTreeNode* leftChild;
00027     HuffmanTreeNode* rightChild;
00028
00029 public:
00030     //C'tor s
00036     explicit HuffmanTreeNode::HuffmanTreeNode()
00037         : freq(0), c('\0'), leftChild(nullptr), rightChild(nullptr) {}
00038
00046     HuffmanTreeNode::HuffmanTreeNode(int frequency, char character, HuffmanTreeNode* left,
      HuffmanTreeNode* right)
00047         : freq(frequency), c(character), leftChild(left), rightChild(right) {}
00048
00054     HuffmanTreeNode::HuffmanTreeNode(int frequency, char character)
00055         : freq(frequency), c(character), leftChild(nullptr), rightChild(nullptr) {}
00056
00061     HuffmanTreeNode::~HuffmanTreeNode() {
00062         // Deallocate left and right child nodes if they exist
00063         if (leftChild != nullptr) {
00064             delete leftChild;
00065             leftChild = nullptr;
00066         }
00067         if (rightChild != nullptr) {
00068             delete rightChild;
00069             rightChild = nullptr;
00070         }
00071     }
00072
00080     bool HuffmanTreeNode::operator<(const HuffmanTreeNode& other) const { return freq > other.freq; }
00081
00092     bool operator () (const HuffmanTreeNode* x, const HuffmanTreeNode* y) {
00093         return *x < *y;
00094     }
00095 };
00096
00097 #endif // !HUFFMAN_TREE_NODE_H
```

4.37 Huffman_unittest.cpp File Reference

```
#include "pch.h"
#include "gtest/gtest.h"
#include "HuffmanComp.h"
#include "HuffmanDec.h"
```

Include dependency graph for Huffman_unittest.cpp:

4.37.1 Detailed Description

Author

eslam

Date

December 2023

Definition in file Huffman_unittest.cpp.

4.38 Huffman_unittest.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00010 #include "pch.h"
00011 #include "gtest/gtest.h"
00012 #include "HuffmanComp.h"
00013 #include "HuffmanDec.h"
00014
00015 namespace {
00016     TEST(HuffmanCompression, CompressionDecompression) {
00017         std::string* inputString = new std::string(R"(<t0><t1><t2>1<t3>Ahmed Ali<t4><t5><t6>Lorem
      ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et
      dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut
      aliquip ex ea commodo consequat.<t7><t8>economy<t8>finance<t5><t6>Lorem ipsum dolor sit amet,
      consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut
      enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo
      consequat.<t7><t8>solar_energy<t9><t10><t2>2<t10><t2>3<t1><t2>2<t3>Yasser Ahmed<t4><t5><t6>Lorem ipsum
      dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
      magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
      commodo consequat.<t7><t8>education<t9><t10><t2>1<t1><t2>3<t3>Mohamed Sherif<t4><t5><t6>Lorem ipsum
      dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore
      magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea
      commodo consequat.<t7><t8>sports<t9><t10><t2>1)");
00018
00019         // Compress the input string
00020         HuffmanComp* compressor = new HuffmanComp(inputString);
00021         std::string* compressedString = compressor->compress();
00022
00023         // Decompress the compressed string
00024         HuffmanDec* decompressor = new HuffmanDec(compressedString);
00025         std::string* decompressedString = decompressor->decompress();
00026
00027         // Check if the decompressed string matches the original input
00028         EXPECT_EQ(*decompressedString, *inputString);
00029
00030         // Clean up memory
00031         delete compressor;
00032         delete decompressor;
00033         compressedString = nullptr;
00034         decompressedString = nullptr;
00035         compressor = nullptr;
00036         decompressor = nullptr;
00037     }
00038 }
```

4.39 MinifyingXML.cpp File Reference

The source file of class MinifyingXML.

```
#include "pch.h"
#include "MinifyingXML.h"
```

Include dependency graph for MinifyingXML.cpp:

4.39.1 Detailed Description

The source file of class MinifyingXML.

Minifying is one of the required functions in the data structure and algorithms course's project. Minifying is a way of decreasing the size of the file by deleting all spaces, tabs, new lines.

This class will minify any flawless XML file.

Operation summary:

- Using the array charToSkip.
- All charToSkip (except the space : ' ') will not be added into the result array.
- Spaces will be added to the result string only if it occurred inside the tag's value, not before or after the value. i.e., "<tagg> value with spaces </tagg>" –apply minifying--> "<tagg>value with spaces</tagg>", on other words, the value will be trimmed from spaces before or after it.

Author

eslam

Date

December 2023

Definition in file MinifyingXML.cpp.

4.40 MinifyingXML.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00023 #include "pch.h"
00024 #include "MinifyingXML.h"
00025
00026  // char To skip in minifying.
00027 const char MinifyingXML::charToSkip[5] = { ' ', '\n', '\t','\v','\f' };
00028
00029 std::string* MinifyingXML::minifyString()
00030 {
00031     // To store the result.
00032     std::string* result = new std::string();
00033     // Length of the original string
00034     int length = this->xmlFile->size();
00035     //
00036     // The max size of the result string is the same of the entered string.
00037     // That happens when the original doesn't contain any extra spaces or
00038     // other charToSkip elements.
00039     //
00040     result->reserve(length);
00041
00042     this->skipFromBeginning(result);
00043     this->skipFromEnd(result);
00044
00045     // Free the extra allocated memory locations.
00046     result->shrink_to_fit();
00047     //return the result.
00048     return result;
00049 }
00050
00051 // TODO: check whether to add the new line too or not if it was in the body value.
00052 // TODO: check whether to delete comments or not.
00053
00054 void MinifyingXML::skipFromBeginning(std::string* result) const
00055 {
00056     // Length of the original string
00057     int length = this->xmlFile->size();
00058     // Flag for skipping spaces.
00059     bool skipSpaces = true;
00060
00061     /*
00062     * Loop for all values starting form 0.
00063     * That will help removing any charToSkip after tags, but it will
00064     * miss the spaces after values and the next tag (starting or ending).
00065     */
00066     // @TODO: check whether to add the new line too or not if it was in the body value.
00067
00068     // To store the value of the current char on this loop.
00069     char currentChar = 0;
00070     for (int i = 0; i < length; i++) {
00071         //get the current element
00072         currentChar = this->xmlFile->at(i);
00073
00074         //check if it was a skip char
00075         if (MinifyingXML::isSkipChar(currentChar)) {
00076             // @TODO if we should  add new spaces to, change the condition here.
00077             // If it was a space and skipSpaces is false, add the space to the result string.
00078             if (currentChar == ' ' && !skipSpaces) {
00079                 result->append(1, currentChar);
00080             }
00081         }
00082         else {
00083             // If not add too the result
00084             result->append(1, currentChar);
00085             // If it was a '>' or  '<',
```

```
00086             // skip the next spaces.
00087             if (currentChar == '<' || currentChar == '>') {
00088                 skipSpaces = true;
00089             }
00090             // else if it was any char, don't skip after it.
00091             else {
00092                 skipSpaces = false;
00093             }
00094         }
00095     }
00096 }
00097
00098 void MinifyingXML::skipFromEnd(std::string* result) const
00099 {
00100     // Length of the original string
00101     int length = result->size();
00102     // Flag for skipping spaces.
00103     bool skipSpaces = true;
00104
00105     /*
00106     * Loop for all values starting form the end (length - 1).
00107     */
00108     // To store the value of the current char on this loop.
00109     char currentChar = 0;
00110     for (int i = length - 1; i >= 0; i--) {
00111         //get the current element
00112         currentChar = result->at(i);
00113         //if a skip space delete it.
00114         if (currentChar == ' ' && skipSpaces) {
00115             result->erase(i, 1);
00116         }
00117
00118         //if it is a '<', set skip to true.
00119         else if (currentChar == '<') {
00120             skipSpaces = true;
00121         }
00122
00123         // if any other char, set skip to false.
00124         else {
00125             skipSpaces = false;
00126         }
00127     }
00128 }
00129
00130 bool MinifyingXML::isSkipChar(const char c)
00131 {
00132     for (char ch : MinifyingXML::charToSkip) {
00133         if (c == ch) {
00134             return true;
00135         }
00136     }
00137     return false;
00138 }
```

4.41 MinifyingXML.h File Reference

Header file of the MinifyingXML class.

```
#include <string>
#include <stdexcept>
```

Include dependency graph for MinifyingXML.h: This graph shows which files directly or indirectly include this file:

Classes

- class MinifyingXML

Macros

- #define MINIFYING_XML_H

4.41.1 Detailed Description

Header file of the MinifyingXML class.

Minifying is one of the required functions in the data structure and algorithms course's project. Minifying is a way of decreasing the size of the file by deleting all spaces, tabs, new lines.

This class will minify any flawless XML file.

Operation summary:

- Using the array charToSkip.
- All charToSkip (except the space : ' ') will not be added into the result array.
- Spaces will be added to the result string only if it occurred inside the tag's value, not before or after the value. i.e., "<tagg> value with spaces </tagg>" –apply minifying--> "<tagg>value with spaces</tagg>", on other words, the value will be trimmed from spaces before or after it.

Author

eslam

Date

December 2023

Definition in file MinifyingXML.h.

4.41.2 Macro Definition Documentation

MINIFYING_XML_H

```
#define MINIFYING_XML_H
```

Definition at line 24 of file MinifyingXML.h.

4.42 MinifyingXML.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00022 #pragma once
00023 #ifndef MINIFYING_XML_H
00024 #define MINIFYING_XML_H
00025 #include <string>
00026
00027 #include <stdexcept>
00028
00029 class MinifyingXML
00030 {
00031 private:
00032     // the file that neads to be minified.
00033     const std::string* xmlFile;
00034
00035     // char To skip in minifying.
00036     static const char charToSkip[5];
00037
00038 public:
```

```
00045     explicit MinifyingXML(const std::string* xmlFile) : xmlFile(xmlFile) {
00046         // check adding a null ptr.
00047         if (xmlFile == nullptr) {
00048             throw std::logic_error("Null pointer exception: Accessing null pointer!");
00049         }
00050     }
00051
00052     //methods
00053
00064     std::string* minifyString();
00065
00075     static bool isSkipChar(const char c);
00076
00077     //helper methods
00078
00095     void skipFromBeginning(std::string* result)const;
00096
00115     void skipFromEnd(std::string* result) const;
00116
00117     //getters and setters, used for debugging
00118     //getters.
00119
00120     //XML file getter.
00121     const std::string* getXMLFile() const { return this->xmlFile; }
00122
00123     //setters
00124     //XML file setter.
00125     void setXMLFile(const std::string* xmlFileNew) {
00126         // check adding a null ptr.
00127         if (xmlFileNew == nullptr) {
00128             throw std::logic_error("Null pointer exception: Accessing null pointer!");
00129         }
00130         this->xmlFile = xmlFileNew;
00131     }
00132 }; //class MinifyingXML
00133
00134 #endif // !MINIFYING_XML_H
```

4.43 MinifyingXML_unittest.cpp File Reference

Unit test code for MinifyingXML class.

```
#include "gtest/gtest.h"
#include "pch.h"
#include "MinifyingXML.h"
```

Include dependency graph for MinifyingXML_unittest.cpp:

4.43.1 Detailed Description

Unit test code for MinifyingXML class.

It includes a test for each member method in the class using gtest framework.

Author

eslam

Date

December 2023

Definition in file MinifyingXML_unittest.cpp.

4.44 MinifyingXML_unittest.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00010 #include "gtest/gtest.h"
00011 #include "pch.h"
00012 #include "MinifyingXML.h"
00013
00014 namespace {
00015     class MinifyingXML_Test_essintials : public ::testing::Test {
00016     protected:
00017
00018         MinifyingXML* m;
00019
00020         void SetUp() override {
00021             m = nullptr;  // Initialize m to nullptr in SetUp
00022             init_m(new std::string(""));
00023         }
00024
00025         void TearDown()override { clearVar(); }
00026
00027     public:
00028         // methods to help with C'tor tests.
00029         void clearVar() {
00030             delete m;  // Safe delete, checks if m is nullptr before deletion
00031             m = nullptr;  // Reset m to nullptr after deletion
00032         }
00033         void init_m(const std::string* s) {
00034             clearVar();
00035             m = new MinifyingXML(s);
00036         }
00037     }; // class MinifyingXML_Test_essintials
00038
00039     //Getters and C'tor tests.
00040     TEST_F(MinifyingXML_Test_essintials, ConstructorAndGettersTest) {
00041         EXPECT_EQ(*m->getXMLFile(), "");
00042
00043         // Test handling null pointer in initialization
00044         std::string* s = nullptr;
00045         EXPECT_THROW(init_m(s), std::logic_error);
00046
00047         // Initialize with a valid string and verify the state
00048         s = new std::string("this is a new string.");
00049         init_m(s);
00050         EXPECT_EQ(*m->getXMLFile(), "this is a new string.");
00051         // Clean up memory after testing
00052         delete s;
00053         s = nullptr;
00054     }
00055
00056     //Setters Test
00057     TEST_F(MinifyingXML_Test_essintials, SettersTest) {
00058         std::string* s = nullptr;
00059         EXPECT_THROW(m->setXMLFile(s), std::logic_error);
00060
00061         //empty string
00062         s = new std::string("");
00063         m->setXMLFile(s);
00064         EXPECT_EQ(*m->getXMLFile(), "");
00065
00066         delete s;
00067         s = nullptr;
00068
00069         // any string
00070         s = new std::string("this is a new string.");
00071         m->setXMLFile(s);
00072         EXPECT_EQ(*m->getXMLFile(), "this is a new string.");
00073
00074         delete s;
00075         s = nullptr;
00076     }
00077
00078     //isChar test
00079     TEST_F(MinifyingXML_Test_essintials, isSkipCharTest) {
00080         //true cases.
00081         EXPECT_TRUE(MinifyingXML::isSkipChar(' '));
00082         EXPECT_TRUE(MinifyingXML::isSkipChar('\t'));
00083         EXPECT_TRUE(MinifyingXML::isSkipChar('\v'));
00084         EXPECT_TRUE(MinifyingXML::isSkipChar('\n'));
00085         EXPECT_TRUE(MinifyingXML::isSkipChar('\f'));
00086
00087         //some false cases
00088         EXPECT_FALSE(MinifyingXML::isSkipChar('p'));
00089         EXPECT_FALSE(MinifyingXML::isSkipChar('a'));
00090         EXPECT_FALSE(MinifyingXML::isSkipChar('0'));
00091         EXPECT_FALSE(MinifyingXML::isSkipChar('3'));
```

```
00092         EXPECT_FALSE(MinifyingXML::isSkipChar('8'));
00093     }
00094
00095     class MinifyingXML_Test_Functionality : public ::testing::Test {
00096     protected:
00097         const std::string* input1;
00098         const std::string* expectedResult;
00099         const std::string* afterMinifying;
00100         MinifyingXML* m;
00101         void SetUp() override {
00102             input1 = new std::string(R"(      <users>
00103         <user>
00104                   <id>        1       </id>
00105             <name>  Ahmed  Ali  </name>
00106             <posts>
00107                 <post>
00108                     <body>  Lorem ipsum dolor sit ametffsjkn</body>
00109                     <topics>
00110                         <topic>     economy</topic>
00111                     </topics>
00112                 </post>
00113             </posts>
00114             <followers>
00115                 <follower>
00116                     <id>2           </id>
00117                 </follower>
00118             </followers>
00119         </user>
00120     </users>     )");
00121
00122             expectedResult = new std::string(R"(<users><user><id>1       </id><name>Ahmed  Ali
      </name><posts><post><body>Lorem ipsum dolor sit
      ametffsjkn</body><topics><topic>economy</topic></topics></post></posts><followers><follower><id>2
      </id></follower></followers></user></users>)");
00123
00124             afterMinifying = new std::string(R"(<users><user><id>1</id><name>Ahmed
      Ali</name><posts><post><body>Lorem ipsum dolor sit
      ametffsjkn</body><topics><topic>economy</topic></topics></post></posts><followers><follower><id>2</id></follower></follo
00125
00126             m = new MinifyingXML(input1);
00127         }
00128
00129         void TearDown() override {
00130             delete input1;
00131             delete expectedResult;
00132             delete afterMinifying;
00133
00134             input1 = nullptr;
00135             expectedResult = nullptr;
00136             afterMinifying = nullptr;
00137         }
00138     }; // class MinifyingXML_Test_Functionality
00139
00140     //helper functions test
00141     TEST_F(MinifyingXML_Test_Functionality, skipFromBeginningTest) {
00142         //action
00143         std::string* output = new std::string();
00144         m->skipFromBeginning(output);
00145
00146         //test
00147         EXPECT_EQ(*output, *expectedResult);
00148
00149         //deallocate
00150         delete output;
00151         output = nullptr;
00152     }
00153
00154     TEST_F(MinifyingXML_Test_Functionality, skipFromEndTest) {
00155         //action
00156         std::string* output = new std::string(*expectedResult);
00157         m->skipFromEnd(output);
00158
00159         //test
00160         EXPECT_EQ(*output, *afterMinifying);
00161
00162         //deallocate
00163         delete output;
00164         output = nullptr;
00165     }
00166
00167     TEST_F(MinifyingXML_Test_Functionality, minifyStringTest) {
00168         //action
00169         const std::string* output = m->minifyString();
00170
00171         //test
00172         EXPECT_EQ(*output, *afterMinifying);
00173
```

```
00174         //deallocate
00175         delete output;
00176         output = nullptr;
00177     }
00178 } // namespace
```

4.45 TagsMapComp.cpp File Reference

The source file of TagsMapComp class.

```
#include "pch.h"
#include "TagsMapComp.h"
```

Include dependency graph for TagsMapComp.cpp:

4.45.1 Detailed Description

The source file of TagsMapComp class.

A compression algorithm that maps tags into numbers. By applying this algorithm, the size file decrease, as many characters in tags will be getting red off, so theses char will not repeated over and over again.

To now the mapping values, a <TagsMap> block will be added to the start of the XML file.

Example: -> File before: <tag0><tag1><tag2></tag2><tag2></tag2></tag1></tag0>

-> File after: <TagMap>tag0,tag1,tag2<Tag/Map> <0><1><2></2><2></2></1></0>

Note

: <TagMap> block is optional, Will not be added to the social network file, is tags are constant there.

: if <TagMap> is added, this algorithm will be efficient only if it contains lots of long tags.

all methods in this class assumes that the input file is flawless.

Author

eslam

Date

December 2023

Definition in file TagsMapComp.cpp.

4.46 TagsMapComp.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00031 #include "pch.h"
00032 #include "TagsMapComp.h"
00033
00034  //initialize defaultTagMapBlock
00035 const std::string* TagsMapComp::defualtTagMapBlock = new std::string(
00036     "<TagMap>users,user,id,name,posts,post,body,topics,topic,followers,follower</TagMap>"
00037 );
00038
00039 void TagsMapComp::mapTags()
00040 {
00041     std::stringstream ss(*this->xmlFile);
00042     std::string tag;
00043     std::string line;
00044
00045     while (std::getline(ss, line, '<')) {
00046         // Trim leading spaces
00047         line.erase(0, line.find_first_not_of(" \t\n\r"));
00048         // Extract the tag name between '<' and '>'
00049         int pos = line.find('>');
00050         if (pos == -1) {
00051             continue;
00052         }
00053         int start = 0;
00054         //closing tag
00055         if (line.at(0) == '/') {
00056             start = 1;
00057         }
00058         int length = pos - start;
00059         tag = line.substr(start, length);
00060
00061         // If the tag wasn't in the map, add it
00062         if (!map->containKey(&std::string(tag))) {
00063             map->add(new std::string(tag));
00064         }
00065     }
00066 }
00067
00068 std::string* TagsMapComp::compress(bool addMapTable)
00069 {
00070     //to store the result.
00071     std::string* result = new std::string();
00072     //length of the original file.
00073     int length = this->xmlFile->size();
00074
00075     //
00076     // The max size of the result string is the same of the entered string.
00077     // // the added 60 is for the mapTable.
00078     //
00079     result->reserve(length + 60);
00080
00081     //add MapTable if required
00082     if (addMapTable) {
00083         std::string* mapTags = map->toString();
00084         result->append(*mapTags);
00085         //result->append(1, '\n');
00086
00087         delete mapTags;
00088         mapTags = nullptr;
00089     }
00090     else {
00091         //  Reinitialize the map to the default Tag map for
00092         //  social network system.
00093         delete map;
00094         map = new Map(TagsMapComp::defualtTagMapBlock);
00095     }
00096
00097     /*
00098     * Loop for all the original string.
00099     * - If the current string is '<'
00100     *       1.Collect the tag after it.
00101     *       2.Map that tag.
00102     *       3.Add the mapped tag to the result string.
00103     * - For other characters, add them to the result.
00104     */
00105     char currentChar = 0;
00106     for (int i = 0; i < length; i++) {
00107         // get current char
00108         currentChar = this->xmlFile->at(i);
00109
00110         //current char is '<' --> map the tag.
00111         if (currentChar == '<') {
00112             // increment the counter to get the next char.
```

```
00113             i++;
00114             // get the next char
00115             currentChar = this->xmlFile->at(i);
00116 
00117             // to know it is an opening or closing tag.
00118             bool openingTag = true;
00119             if (currentChar == '/') {
00120                 openingTag = false;
00121                 // increment the counter to get the next char.
00122                 i++;
00123                 // get the next char
00124                 currentChar = this->xmlFile->at(i);
00125             }
00126 
00127             // To store the tag.
00128             std::string tag = std::string();
00129             //loop to get the full tag
00130             while (currentChar != '>') {
00131                 // append it to the tag string
00132                 tag.append(1, currentChar);
00133                 // increment the counter.
00134                 i++;
00135                 // get current char
00136                 currentChar = this->xmlFile->at(i);
00137             }
00138 
00139             //map the tag
00140             std::string afterMaping = std::string("<");
00141             if (!openingTag) {
00142                 afterMaping.append("/");
00143             }
00144             afterMaping.append(std::to_string(map->getValue(&tag)));
00145             afterMaping.append(1, '>');
00146 
00147             //append to the result.
00148             result->append(afterMaping);
00149         } // if current char == '<'
00150 
00151         else {
00152             result->append(1, currentChar);
00153         }
00154     }
00155 
00156     // Free the extra allocated memory locations.
00157     result->shrink_to_fit();
00158     return result;
00159 }// compress()
```

4.47 TagsMapComp.h File Reference

The header file of TagsMapComp class.

```
#include <string>
#include "Map.h"
```

Include dependency graph for TagsMapComp.h: This graph shows which files directly or indirectly include this file:

Classes

- class TagsMapComp

Macros

- #define TAGS_MAP_Comp_H

4.47.1 Detailed Description

The header file of TagsMapComp class.

A compression algorithm that maps tags into numbers. By applying this algorithm, the size file decrease, as many characters in tags will be getting red off, so theses char will not repeated over and over again.

To now the mapping values, a <TagsMap> block will be added to the start of the XML file.

Example: -> File before: <tag0><tag1><tag2></tag2><tag2></tag2></tag1></tag0>

-> File after: <TagMap>tag0,tag1,tag2<Tag/Map> <0><1><2></2><2></2></1></0>

Note

: <TagMap> block is optional, Will not be added to the social network file, is tags are constant there.

: if <TagMap> is added, this algorithm will be efficient only if it contains lots of long tags.

all methods in this class assumes that the input file is flawless.

Author

eslam

Date

December 2023

Definition in file TagsMapComp.h.

4.47.2 Macro Definition Documentation

TAGS_MAP_Comp_H

```
#define TAGS_MAP_Comp_H
```

Definition at line 32 of file TagsMapComp.h.

4.48 TagsMapComp.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00030 #pragma once
00031 #ifndef TAGS_MAP_Comp_H
00032 #define TAGS_MAP_Comp_H
00033
00034 #include <string>
00035 #include "Map.h"
00036
00037 class TagsMapComp
00038 {
00039 private:
00040     const std::string* xmlFile;
00041     const static std::string* defualtTagMapBlock;
00042     //Map of tag values.
00043     Map* map;
00044 public:
00053     explicit TagsMapComp(const std::string* xmlFile) : xmlFile(xmlFile),
00054         map(new Map()) {
00055         mapTags();
00056     }
00061     ~TagsMapComp() { delete map; }
00070     void mapTags();
00071
00087     std::string* compress(bool addMapTable = false);
00088 };
00089
00090 #endif // !TAGS_MAP_Comp_H
```

4.49 TagsMapComp_unittest.cpp File Reference

Unit test code for TagsMapComp class.

```
#include "gtest/gtest.h"
#include "pch.h"
#include "TagsMapComp.h"
```

Include dependency graph for TagsMapComp_unittest.cpp:

4.49.1 Detailed Description

Unit test code for TagsMapComp class.

Author

eslam

Date

December 2023

Definition in file TagsMapComp_unittest.cpp.

4.50 TagsMapComp_unittest.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00009 #include "gtest/gtest.h"
00010 #include "pch.h"
00011 #include "TagsMapComp.h"
00012
00013 namespace {
00014     class TagsMapCompTest : public::testing::Test {
00015     public:
00016         TagsMapComp* t;
00017         std::string* input;
00018         std::string* result;
00019         std::string* resultWithMap;
00020     protected:
00021         void SetUp() {
00022             input = new std::string(R"(      <users>
00023         <user>
00024                   <id>        1       </id>
00025             <name>  Ahmed  Ali  </name>
00026             <posts>
00027                 <post>
00028                     <body>  Lorem ipsum dolor sit ametffsjkn</body>
00029                     <topics>
00030                         <topic>     economy</topic>
00031                     </topics>
00032                 </post>
00033             </posts>
00034             <followers>
00035                 <follower>
00036                     <id>2           </id>
00037                 </follower>
00038             </followers>
00039         </user>
00040     </users>     )");
00041
00042             t = new TagsMapComp(input);
00043
00044             result = new std::string(R"(      <0>
00045         <1>
00046                   <2>        1       </2>
00047             <3>  Ahmed  Ali  </3>
```

```
00048             <4>
00049                 <5>
00050                     <6>  Lorem ipsum dolor sit ametffsjkn</6>
00051                     <7>
00052                         <8>     economy</8>
00053                     </7>
00054                 </5>
00055             </4>
00056             <9>
00057                 <10>
00058                     <2>2           </2>
00059                 </10>
00060             </9>
00061         </1>
00062     </0>     )");
00063
00064             resultWithMap = new
      std::string(R"(<TagMap>users,user,id,name,posts,post,body,topics,topic,followers,follower</TagMap>
      <0>
00065         <1>
00066                   <2>        1       </2>
00067             <3>  Ahmed  Ali  </3>
00068             <4>
00069                 <5>
00070                     <6>  Lorem ipsum dolor sit ametffsjkn</6>
00071                     <7>
00072                         <8>     economy</8>
00073                     </7>
00074                 </5>
00075             </4>
00076             <9>
00077                 <10>
00078                     <2>2           </2>
00079                 </10>
00080             </9>
00081         </1>
00082     </0>     )");
00083         }
00084
00085         void TearDown() {
00086             delete t;
00087             delete input;
00088             delete result;
00089             delete resultWithMap;
00090             t = nullptr;
00091             input = nullptr;
00092             result = nullptr;
00093             resultWithMap = nullptr;
00094         }
00095     }; // TagsMapCompTest
00096
00097     TEST_F(TagsMapCompTest, noMap) {
00098         std::string* s = t->compress(false);
00099         EXPECT_EQ(*s, *result);
00100
00101         delete s;
00102         s = nullptr;
00103     }
00104
00105     TEST_F(TagsMapCompTest, withMap) {
00106         std::string* s = t->compress(true);
00107         EXPECT_EQ(*s, *resultWithMap);
00108
00109         delete s;
00110         s = nullptr;
00111     }
00112 }
```

4.51 TagsMapDec.cpp File Reference

The header file of TagsMapDec class.

```
#include "pch.h"
#include "TagsMapDec.h"
```

Include dependency graph for TagsMapDec.cpp:

4.51.1 Detailed Description

The header file of TagsMapDec class.

The decompression algorithm of TagsMap compression algorithm. The decompression will re-map the tags to their original value.

The file might contain a TagsMap tag at the beginning, from that tag we can get the mapping numbers.

If the file doesn't contain this tag, then it will be assumed to be: <TagMap>users,user,id,name,posts,post,body,topics,topic,followers,fo
TagMap>. which will be used for social network system only.

Example: -> File before: <TagMap>tag0,tag1,tag2<Tag/Map> <0><1><2></2><2></2></1></0>

-> File after: <tag0><tag1><tag2></tag2><tag2></tag2></tag1></tag0>

See also

TagsMapComp

Author

eslam

Date

December 2023

Definition in file TagsMapDec.cpp.

4.52 TagsMapDec.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00028 #include "pch.h"
00029 #include "TagsMapDec.h"
00030
00031 void TagsMapDec::getMapTags()
00032 {
00033     const std::string* tagMapLine = this->getTagsMapBlock();
00034     this->map = new Map(tagMapLine);
00035     if (tagMapLine != defualtTagMapBlock) {
00036         delete tagMapLine;
00037         tagMapLine = nullptr;
00038     }
00039 }
00040
00041 const std::string* TagsMapDec::getTagsMapBlock()
00042 {
00043     //Minify the file
00044     MinifyingXML* m = new MinifyingXML(this->xmlFile);
00045     std::string* afterMinifying = m->minifyString();
00046     //deallocate m
00047     delete m;
00048     m = nullptr;
00049
00050     //get the position of both the opening and the closing tags
00051     int start = afterMinifying->find("<TagMap>");
00052     int end = afterMinifying->find("</TagMap>");
00053     //if any was not found, then the file is assumed to be for
00054     //social network system --> return the default line
00055     if (start == std::string::npos && end == std::string::npos) {
00056         return defualtTagMapBlock;
00057     }
00058
```

```
00059     //if tagMap wasn't in the first position, then the file is defected
00060     else if (start != 0) {
00061         throw std::runtime_error("Defected file.");
00062     }
00063
00064     //get the line and return it.
00065     const std::string* result = new std::string(
00066         afterMinifying->substr(start, end + 9 - start)
00067     );
00068     //deallocate after minifying string.
00069     delete afterMinifying;
00070     afterMinifying = nullptr;
00071     return result;
00072 }// getTagsMapBlock()
00073
00074 std::string* TagsMapDec::decompress()
00075 {
00076     //to store the result.
00077     std::string* result = new std::string();
00078     //length of the original file.
00079     int length = this->xmlFile->size();
00080     // Assume that the worst case will be triple the size.
00081     result->reserve(length * 3);
00082
00083     //skip the TagMap block
00084     int i = this->xmlFile->find("</TagMap>");
00085     // if the block is not found, start from the beginning.
00086     if (i == std::string::npos) {
00087         i = 0;
00088     }
00089     else {
00090         i += 9;
00091     }
00092
00093     /*
00094     * Loop for all the original string.
00095     * - If the current string is '<'
00096     *       1.Collect the tag after it.
00097     *       2.Map that tag.
00098     *       3.Add the mapped tag to the result string.
00099     * - For other characters, add them to the result.
00100     */
00101
00102     char currentChar = 0;
00103     for (i; i < length; i++) {
00104         // get current char
00105         currentChar = this->xmlFile->at(i);
00106
00107         //current char is '<' --> map the tag.
00108         if (currentChar == '<') {
00109             // increment the counter to get the next char.
00110             i++;
00111             // get the next char
00112             currentChar = this->xmlFile->at(i);
00113
00114             // to know it is an opening or closing tag.
00115             bool openingTag = true;
00116             if (currentChar == '/') {
00117                 openingTag = false;
00118                 // increment the counter to get the next char.
00119                 i++;
00120                 // get the next char
00121                 currentChar = this->xmlFile->at(i);
00122             }
00123
00124             // To store the tag number.
00125             std::string tag = std::string();
00126             //loop to get the full tag
00127             while (currentChar != '>') {
00128                 // append it to the tag string
00129                 tag.append(1, currentChar);
00130                 // increment the counter.
00131                 i++;
00132                 // get current char
00133                 currentChar = this->xmlFile->at(i);
00134             }
00135             //get the number from the tag
00136             int value = std::stoi(tag);
00137
00138             //map the tag
00139             std::string afterMaping = std::string("<");
00140             if (!openingTag) {
00141                 afterMaping.append("/");
00142             }
00143
00144             afterMaping.append(*map->getKey(value));
00145             afterMaping.append(1, '>');
```

```
00146
00147             //append to the result.
00148             result->append(afterMaping);
00149         } // if current char == '<'
00150
00151         else {
00152             result->append(1, currentChar);
00153         }
00154     }
00155
00156     // Free the extra allocated memory locations.
00157     result->shrink_to_fit();
00158     return result;
00159 }// decompress()
```

4.53 TagsMapDec.h File Reference

The header file of TagsMapDec class.

```
#include "Map.h"
#include "MinifyingXML.h"
#include <string>
```

Include dependency graph for TagsMapDec.h: This graph shows which files directly or indirectly include this file:

Classes

- class TagsMapDec

Macros

- #define TAGS_MAP_DEC_H

4.53.1 Detailed Description

The header file of TagsMapDec class.

The decompression algorithm of TagsMap compression algorithm. The decompression will re-map the tags to their original value.

The file might contain a TagsMap tag at the beginning, from that tag we can get the mapping numbers.

If the file doesn't contain this tag, then it will be assumed to be: <TagMap>users,user,id,name,posts,post,body,topics,topic,followers,fo TagMap>. which will be used for social network system only.

Example: -> File before: <TagMap>tag0,tag1,tag2<Tag/Map> <0><1><2></2><2></2></1></0>

-> File after: <tag0><tag1><tag2></tag2><tag2></tag2></tag1></tag0>

See also

TagsMapComp

Author

eslam

Date

December 2023

Definition in file TagsMapDec.h.

4.53.2 Macro Definition Documentation

TAGS_MAP_DEC_H

```
#define TAGS_MAP_DEC_H
```

Definition at line 30 of file TagsMapDec.h.

4.54 TagsMapDec.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00028 #pragma once
00029 #ifndef TAGS_MAP_DEC_H
00030 #define TAGS_MAP_DEC_H
00031
00032 #include "Map.h"
00033 #include "MinifyingXML.h"
00034 #include <string>
00035
00036 class TagsMapDec
00037 {
00038 private:
00039     const std::string* xmlFile;
00040
00041     std::string* defualtTagMapBlock;
00042
00043     //Map of tag values.
00044     Map* map;
00045
00046     //helper methods
00047
00051     void getMapTags();
00065     const std::string* getTagsMapBlock();
00066 public:
00073     explicit TagsMapDec(const std::string* xmlFile) : xmlFile(xmlFile) {
00074         defualtTagMapBlock = new std::string(
00075             "<TagMap>users,user,id,name,posts,post,body,topics,topic,followers,follower</TagMap>"
00076         );
00077         getMapTags();
00078     }
00083     ~TagsMapDec() {
00084         delete map;
00085         map = nullptr;
00086
00087         if (defualtTagMapBlock != nullptr) {
00088             delete defualtTagMapBlock;
00089
00090             defualtTagMapBlock = nullptr;
00091         }
00092     }
00099     std::string* decompress();
00100 };
00101 #endif // !TAGS_MAP_DEC_H
```

4.55 TagsMapDec_unittest.cpp File Reference

Unit test code for TagsMapDec class.

```
#include "gtest/gtest.h"
#include "pch.h"
#include "TagsMapDec.h"
```

Include dependency graph for TagsMapDec_unittest.cpp:

4.55.1 Detailed Description

Unit test code for TagsMapDec class.

Author

eslam

Date

December 2023

Definition in file TagsMapDec_unittest.cpp.

4.56 TagsMapDec_unittest.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00009 #include "gtest/gtest.h"
00010 #include "pch.h"
00011 #include "TagsMapDec.h"
00012
00013 namespace {
00014     class TagsMapDecTest : public::testing::Test {
00015     public:
00016         TagsMapDec* t;
00017         std::string* inputWithMap;
00018         std::string* inputWithOutMap;
00019         std::string* result;
00020     protected:
00021         void SetUp() override {
00022             inputWithMap = new
      std::string(R"(<TagMap>users,user,id,name,posts,post,body,topics,topic,followers,follower</TagMap>
      <0>
00023         <1>
00024                   <2>        1       </2>
00025             <3>  Ahmed  Ali  </3>
00026             <4>
00027                 <5>
00028                     <6>  Lorem ipsum dolor sit ametffsjkn</6>
00029                     <7>
00030                         <8>     economy</8>
00031                     </7>
00032                 </5>
00033             </4>
00034             <9>
00035                 <10>
00036                     <2>2           </2>
00037                 </10>
00038             </9>
00039         </1>
00040     </0>     )");
00041
00042             inputWithOutMap = new std::string(R"(      <0>
00043         <1>
00044                   <2>        1       </2>
00045             <3>  Ahmed  Ali  </3>
00046             <4>
00047                 <5>
00048                     <6>  Lorem ipsum dolor sit ametffsjkn</6>
00049                     <7>
00050                         <8>     economy</8>
00051                     </7>
00052                 </5>
00053             </4>
00054             <9>
00055                 <10>
00056                     <2>2           </2>
00057                 </10>
00058             </9>
00059         </1>
00060     </0>     )");
00061
```

```
00062             result = new std::string(R"(      <users>
00063         <user>
00064                   <id>        1       </id>
00065             <name>  Ahmed  Ali  </name>
00066             <posts>
00067                 <post>
00068                     <body>  Lorem ipsum dolor sit ametffsjkn</body>
00069                     <topics>
00070                         <topic>     economy</topic>
00071                     </topics>
00072                 </post>
00073             </posts>
00074             <followers>
00075                 <follower>
00076                     <id>2           </id>
00077                 </follower>
00078             </followers>
00079         </user>
00080     </users>     )");
00081         }
00082
00083         void TearDown() {
00084             delete t;
00085             t = nullptr;
00086             delete inputWithMap;
00087             inputWithMap = nullptr;
00088             delete inputWithOutMap;
00089             inputWithOutMap = nullptr;
00090             delete result;
00091             result = nullptr;
00092         }
00093     };
00094
00095     TEST_F(TagsMapDecTest, withMap) {
00096         t = new TagsMapDec(inputWithMap);
00097         std::string* s = t->decompress();
00098         EXPECT_EQ(*s, *result);
00099
00100         delete s;
00101         s = nullptr;
00102     }
00103
00104     TEST_F(TagsMapDecTest, withOutMap) {
00105         t = new TagsMapDec(inputWithOutMap);
00106         std::string* s = t->decompress();
00107         EXPECT_EQ(*s, *result);
00108
00109         delete s;
00110         s = nullptr;
00111     }
00112 }
```

4.57 Map.cpp File Reference

The source file of the simple Map.

```
#include "pch.h"
#include "Map.h"
```

Include dependency graph for Map.cpp:

4.57.1 Detailed Description

The source file of the simple Map.

This a simple implementation of Map data structure that will help Mapping tags into numbers. Each tag will mapped into the value of its position in the vector.

Author

eslam

Date

December 2023

Definition in file Map.cpp.

4.58 Map.cpp

Go to the documentation of this file.

```
00001 /*****************************************************************/
00013 #include "pch.h"
00014 #include "Map.h"
00015
00016 void Map::trimString(std::string& str)
00017 {
00018     str.erase(0, str.find_first_not_of(' ')); // Remove leading spaces
00019     str.erase(str.find_last_not_of(' ') + 1); // Remove trailing spaces
00020 }
00021
00022 Map::Map(const std::string* tagMapBlock)
00023 {
00024     this->arr = new std::vector<std::string*>();
00025     //clear the spaces of the file.
00026     MinifyingXML* m = new MinifyingXML(tagMapBlock);
00027     std::string* afterMini = m->minifyString();
00028     delete m;
00029     m = nullptr;
00030
00031     // Get the positions of the opening and closing tags to remove them
00032     int openingTagPos = afterMini->find("<TagMap>");
00033     int closingTagPos = afterMini->find("</TagMap>");
00034
00035     //check that the tag is available.
00036     if (openingTagPos == std::string::npos
00037         || closingTagPos == std::string::npos) {
00038         throw std::runtime_error("Defected TagMAp block");
00039     }
00040
00041     //erase the tag
00042     // Erase the opening tag "<TagMap>"
00043     afterMini->erase(openingTagPos, 8);
00044     // Erase the closing tag "</TagMap>"
00045     afterMini->erase(afterMini->size() - 9, 9);
00046
00047     // add the values between ',' into the arr vector.
00048     std::stringstream ss(*afterMini);
00049     std::string* token = new std::string();
00050
00051     while (std::getline(ss, *token, ',')) {
00052         Map::trimString(*token);
00053         this->add(token);
00054         token = new std::string();
00055     }
00056     delete token;
00057     token = nullptr;
00058
00059     delete afterMini;
00060     afterMini = nullptr;
00061 }
00062
00063 int Map::add(std::string* key)
00064 {
00065     arr->push_back(key);
00066     return arr->size() - 1;
00067 }
00068
00069 int Map::getValue(const std::string* key) const
00070 {
00071     int counter = -1;
00072     for (int i = 0; i < arr->size(); i++) {
00073         std::string* k = arr->at(i);
00074         if (*k == *key) {
00075             return i;
00076         }
00077     }
00078     return -1;
00079 }
00080
00081 const std::string* Map::getKey(int value) const
00082 {
00083     if (arr->size() == 0) {
00084         throw std::runtime_error("array out of bound exception");
00085     }
00086     if (value < 0 || value> arr->size() - 1) {
00087         throw std::runtime_error("array out of bound exception");
00088     }
00089     return arr->at(value);
00090 }
00091
00092 bool Map::containKey(const std::string* key) const {
00093     return (this->getValue(key) == -1) ? false : true;
00094 }
```

```
00095
00096 std::string* Map::toString()
00097 {
00098     if (arr->size() == 0) {
00099         throw std::runtime_error("No value are being mapped");
00100     }
00101     std::string* result = new std::string("<TagMap>");
00102     for (std::string* s : *arr) {
00103         result->append(*s);
00104         result->append(",");
00105     }
00106     result->erase(result->size() - 1);
00107     result->append("</TagMap>");
00108     return result;
00109 }
```

4.59 Map.h File Reference

The header file of the simple Map.

```
#include <vector>
#include "MinifyingXML.h"
#include <sstream>
```

Include dependency graph for Map.h: This graph shows which files directly or indirectly include this file:

Classes

- class Map

Macros

- #define MAP_H

4.59.1 Detailed Description

The header file of the simple Map.

This a simple implementation of Map data structure that will help Mapping tags into numbers. Each tag will mapped into the value of its position in the vector.

Author

eslam

Date

December 2023

Definition in file Map.h.

4.59.2 Macro Definition Documentation

MAP_H

```
#define MAP_H
```

Definition at line 15 of file Map.h.

4.60 Map.h

Go to the documentation of this file.

```
00001 /*****************************************************************/
00013 #pragma once
00014 #ifndef MAP_H
00015 #define MAP_H
00016
00017 #include <vector>
00018 #include "MinifyingXML.h"
00019 #include <sstream>
00020
00021 class Map
00022 {
00023 private:
00024     std::vector<std::string*>* arr;
00025
00026     //helper method
00032     static void trimString(std::string& str);
00033
00034 public:
00039     explicit Map() :arr(new std::vector<std::string*>()) {}
00049     explicit Map(const std::string* tagMapBlock);
00054     ~Map() {
00055         for (std::string* s : *arr) {
00056             delete s;
00057         }
00058         delete arr;
00059     }
00060
00061     //methods
00062
00069     int add(std::string* key);
00076     int getValue(const std::string* key) const;
00084     const std::string* getKey(int value) const;
00092     bool containKey(const std::string* key) const;
00093
00097     int getSize() { return arr->size(); }
00098
00105     std::string* toString();
00106 };
00107
00108 #endif // !MAP_H
```

4.61 Map_unittest.cpp File Reference

Unit test code for Map class.

```
#include "gtest/gtest.h"
#include "pch.h"
#include "Map.h"
```

Include dependency graph for Map_unittest.cpp:

4.61.1 Detailed Description

Unit test code for Map class.

Author

eslam

Date

December 2023

Definition in file Map_unittest.cpp.

4.62 Map_unittest.cpp

Go to the documentation of this file.

```
00001 /*************************************************************/
00008 #include "gtest/gtest.h"
00009 #include "pch.h"
00010 #include "Map.h"
00011
00012 namespace {
00013     class Map_Test : public ::testing::Test {
00014     public:
00015         Map* m;
00016         std::string* s0;
00017         std::string* s1;
00018         std::string* s2;
00019         std::string* s3;
00020
00021     protected:
00022         void SetUp() override {
00023             m = new Map();
00024             s0 = new std::string("v0");
00025             s1 = new std::string("v1");
00026             s2 = new std::string("v2");
00027             s3 = new std::string("v3");
00028         }
00029
00030         void add() {
00031             m->add(s0);
00032             m->add(s1);
00033             m->add(s2);
00034             m->add(s3);
00035         }
00036         void TearDown() override {
00037             delete m;
00038             m = nullptr;
00039         }
00040     };
00041
00042     TEST_F(Map_Test, emptyMap) {
00043         EXPECT_EQ(m->getSize(), 0);
00044         std::string* s = new std::string("any");
00045         EXPECT_EQ(m->getValue(s), -1);
00046
00047         EXPECT_THROW(m->getKey(0), std::runtime_error);
00048         EXPECT_THROW(m->getKey(-1), std::runtime_error);
00049         EXPECT_THROW(m->getKey(5), std::runtime_error);
00050
00051         EXPECT_FALSE(m->containKey(s));
00052
00053         EXPECT_THROW(m->toString(), std::runtime_error);
00054
00055         delete s;
00056         s = nullptr;
00057     }
00058
00059     TEST_F(Map_Test, AddToTheMap) {
00060         add();
00061
00062         EXPECT_EQ(m->getSize(), 4);
00063
00064         EXPECT_EQ(m->getValue(s0), 0);
00065         EXPECT_EQ(m->getValue(s1), 1);
00066         EXPECT_EQ(m->getValue(s2), 2);
00067         EXPECT_EQ(m->getValue(s3), 3);
00068
00069         EXPECT_THROW(m->getKey(5), std::runtime_error);
00070
00071         EXPECT_EQ(m->getKey(0), s0);
00072         EXPECT_EQ(m->getKey(1), s1);
00073         EXPECT_EQ(m->getKey(2), s2);
00074         EXPECT_EQ(m->getKey(3), s3);
00075
00076         EXPECT_TRUE(m->containKey(s0));
00077         EXPECT_TRUE(m->containKey(s1));
00078         EXPECT_TRUE(m->containKey(s2));
00079         EXPECT_TRUE(m->containKey(s3));
00080
00081         std::string eOutput = "<TagMap>v0,v1,v2,v3</TagMap>";
00082         std::string* output = m->toString();
00083         EXPECT_EQ(*output, eOutput);
00084         delete output;
00085         output = nullptr;
00086     }
00087
00088     class Map_Test2 : public::testing::Test {
00089     public:
```

```
00090         Map* m;
00091         std::string* TagMapBlock;
00092         std::string* output;
00093     protected:
00094         void SetUp() override {
00095             TagMapBlock = new std::string(R"(     <TagMap>   v0,v1,
00096        v2,   v3
00097        </TagMap>
00098        )");
00099             output = new std::string(R"(<TagMap>v0,v1,v2,v3</TagMap>)");
00100             m = new Map(TagMapBlock);
00101         }
00102         void TearDown() override {
00103             delete m;
00104             m = nullptr;
00105             delete TagMapBlock;
00106             TagMapBlock = nullptr;
00107             delete output;
00108             output = nullptr;
00109         }
00110     }; // Map_Test2
00111
00112     TEST_F(Map_Test2, MapInitConstrucotr) {
00113         EXPECT_EQ(m->getSize(), 4);
00114         EXPECT_EQ(*m->getKey(0), "v0");
00115         EXPECT_EQ(*m->getKey(1), "v1");
00116         EXPECT_EQ(*m->getKey(2), "v2");
00117         EXPECT_EQ(*m->getKey(3), "v3");
00118         std::string* s = m->toString();
00119         EXPECT_EQ(*s, *output);
00120         delete s;
00121         s = nullptr;
00122     }
00123 } // namespace
```

Index