ChunkyString-1

HW 7

Generated by Doxygen 1.8.7

Wed Mar 28 2018 20:25:39

Contents

Chapter 1

HW 07: ChunkyString Testing

1.1 Introduction

The C++ standard `string` is convenient for most applications, but vague about asymptotic complexity for its operations. While its often a waste of time to reimplement STL structures, in this case we want a string that can ensure efficiency in the following areas:

- Memory Usage
- Insertion of Characters
- Deletion of Characters

To satisfy this, we write a compromise between a linked-list of characters and an array of characters called ChunkyString. Previous implementations defined a class called ChunkyString directly, and for testing we use the interface and (compiled) implementations of that class. Next week, we'll define a ChunkyList that is templated, and will instantiate the template to define a ChunkyString as a list of characters.

1.2 Usage

The ChunkyString class can be used by including `chunkystring.hpp` in any file. A test suite to assert its correctness is provided in stringtest.cpp. The ChunkyList class can be used by including `chunkylist.hpp` in any file. A test suite to assert its correctness is provided in stringtest.cpp.

Chapter 2

Class Index

2.1 Class List

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

Chapter 3

File Index

3.1 File List

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

Chapter 4

Class Documentation

4.1 TestingLogger::AssertInfo Struct Reference

Public Attributes

- int **asserts_** = 0
- int **failures_** = 0

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

- testing-logger.hpp

4.2 ChunkyString::Chunk Struct Reference

Holds part of a ChunkyString.

Public Attributes

- size_t length_

  Number of characters occupying this chunk.
- char chars_ [CHUNKSIZE]

  Contents of this chunk.

Static Public Attributes

- static const size_t CHUNKSIZE = 12

  Maximum size of a chunk.

4.2.1 Detailed Description

Holds part of a ChunkyString.

4.2.2 Member Data Documentation

4.2.2.1 const size_t ChunkyString::Chunk::CHUNKSIZE = 12 [static]

Maximum size of a chunk.

Remarks

Although we set the value of CHUNKSIZE here to be 12, that's an implementation detail. We're allowed to change it, and user's code (as well as our own implementation code) shouldn't depend on CHUNKSIZE having a particular value.

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

- chunkystring-past.hpp

4.3 ChunkyList< ELEMENT >::Chunk Struct Reference

Holds part of a ChunkyList.

Public Attributes

- size_t length_

  Number of characters occupying this chunk.
- ELEMENT elements_ [CHUNKSIZE]

  Contents of this chunk.

Static Public Attributes

- static const size_t CHUNKSIZE = 12

  Maximum size of a chunk.

4.3.1 Detailed Description

template<typename ELEMENT>struct ChunkyList< ELEMENT >::Chunk

Holds part of a ChunkyList.

4.3.2 Member Data Documentation

4.3.2.1 template<typename ELEMENT> const size_t ChunkyList< ELEMENT >::Chunk::CHUNKSIZE = 12 [static]

Maximum size of a chunk.

Remarks

Although we set the value of CHUNKSIZE here to be 12, that's an implementation detail. We're allowed to change it, and user's code (as well as our own implementation code) shouldn't depend on CHUNKSIZE having a particular value.

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

- chunkylist.hpp

4.4 ChunkyList< ELEMENT > Class Template Reference

Efficiently represents strings where insert and erase are constant-time operations.

`#include <chunkylist.hpp>`

Classes

- struct Chunk

  Holds part of a ChunkyList.
- class Iterator

  Iterator for ChunkyList.

Public Types

- using **iterator** = Iterator

Public Member Functions

- ChunkyList ()

  Default constructor.
- iterator begin ()

  Return an iterator to the first element in the ChunkyList.
- iterator end ()

  Return an iterator to "one past the end".
- void push_back (ELEMENT e)

  Inserts an element at the end of the ChunkyList.
- size_t size () const

  List size.
- ChunkyList< ELEMENT > & operator+= (const ChunkyList< ELEMENT > &rhs)

  List concatenation.
- bool operator== (const ChunkyList< ELEMENT > &rhs) const

  List equality.
- bool operator!= (const ChunkyList< ELEMENT > &rhs) const

  List inequality.
- std::ostream & print (std::ostream &out) const

  List printing.
- bool operator< (const ChunkyList< ELEMENT > &rhs) const

List comparison.

- iterator insert (iterator i, ELEMENT e)

  Insert an element before the element at i.

- iterator erase (iterator i)

  Erase an element at i.

- double utilization () const

  Average capacity of each chunk, as a fraction.

Private Attributes

- size_t size_

  Length of the string.

- std::list< Chunk > chunks_

  Linked list of chunks.

4.4.1 Detailed Description

template<typename ELEMENT>class ChunkyList< ELEMENT >

Efficiently represents strings where insert and erase are constant-time operations.

This class is comparable to a linked-list of elements, but more space efficient.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 template<typename ELEMENT> ChunkyList< ELEMENT >::ChunkyList ( )

Default constructor.

Note

constant time

4.4.3 Member Function Documentation

4.4.3.1 template<typename ELEMENT> iterator ChunkyList< ELEMENT >::erase (iterator *i*)

Erase an element at i.

What makes ChunkyList special is its ability to insert and erase elements quickly while remaining space efficient.

Parameters

i	iterator pointing to the element to erase

Returns

an iterator pointing to the element after the one that was deleted.

Note

constant time

Warning

invalidates all iterators except the returned iterator
erasing from an empty list is undefined behavior

4.4.3.2 template<typename ELEMENT> iterator ChunkyList< ELEMENT >::insert (iterator *i*, ELEMENT *e*)

Insert an element before the element at i.

What makes ChunkyList special is its ability to insert and erase elements quickly while remaining space efficient.

Parameters

i	iterator to specify insertion point
e	element to insert

Returns

an iterator pointing to the newly inserted element.

Note

constant time

Warning

invalidates all iterators except the returned iterator

4.4.3.3 template<typename ELEMENT> bool ChunkyList< ELEMENT >::operator< (const ChunkyList< ELEMENT > & *rhs*) const

List comparison.

Remarks

For lists of chars, comparison is lexicographic, which means:

- Two lists `s1` and `s2` are compared character by character.
- The first characters that aren't equal determine the order. If the character value from `s1` is smaller than the corresponding one from `s2`, then `s1 < s2`, and vice-versa.
- If `s1` is a prefix of `s2`, then `s1 < s2`, and vice-versa.
- If `s1` and `s2` have exactly the same character, then neither is less than the other.
- An empty string is less than any other string, except the empty string.

See also

http://www.cplusplus.com/reference/string/string/compare/
http://en.cppreference.com/w/cpp/algorithm/lexicographical_compare

4.4.3.4 template<typename ELEMENT> void ChunkyList< ELEMENT >::push_back (ELEMENT *e*)

Inserts an element at the end of the ChunkyList.

Parameters

e	Element to insert

Note

constant time

4.4.3.5 template<typename ELEMENT> size_t ChunkyList< ELEMENT >::size ( ) const

List size.

Note

constant time

4.4.3.6 template<typename ELEMENT> double ChunkyList< ELEMENT >::utilization ( ) const

Average capacity of each chunk, as a fraction.

This function computes the fraction of the ChunkyList's element cells that are in use. It is defined as

$$\frac{\text{number of elements in the list}}{\text{number of chunks} \times \text{CHUNKSIZE}}$$

For reasonably sized lists (i.e., those with more than one or two elements), utilization should never fall to near one element per chunk; otherwise the data structure would be wasting too much space.

The utilization for an empty list is undefined (i.e., any value is acceptable).

Note

constant time

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

- chunkylist.hpp

4.5 ChunkyString Class Reference

Efficiently represents strings where insert and erase are constant-time operations.

`#include <chunkystring-past.hpp>`

Classes

- struct Chunk

  Holds part of a ChunkyString.
- class Iterator

  Iterator for ChunkyString.

Public Types

- using **iterator** = Iterator

Public Member Functions

- ChunkyString ()

  Default constructor.
- iterator begin ()

  Return an iterator to the first character in the ChunkyString.
- iterator end ()

  Return an iterator to "one past the end".
- void push_back (char c)

  Inserts a character at the end of the ChunkyString.
- size_t size () const

  String size.
- ChunkyString & operator+= (const ChunkyString &rhs)

  String concatenation.
- bool operator== (const ChunkyString &rhs) const

  String equality.
- bool operator!= (const ChunkyString &rhs) const

  String inequality.
- std::ostream & print (std::ostream &out) const

  String printing.
- bool operator< (const ChunkyString &rhs) const

  String comparison.
- iterator insert (iterator i, char c)

  Insert a character before the character at i.
- iterator erase (iterator i)

  Erase a character at i.
- double utilization () const

  Average capacity of each chunk, as a fraction.

Private Attributes

- size_t size_

  Length of the string.
- std::list< Chunk > chunks_

  Linked list of chunks.

4.5.1 Detailed Description

Efficiently represents strings where insert and erase are constant-time operations.

This class is comparable to a linked-list of characters, but more space efficient.

4.5.2 Constructor & Destructor Documentation

4.5.2.1 ChunkyString::ChunkyString ( )

Default constructor.

Note

constant time

4.5.3 Member Function Documentation

4.5.3.1 iterator ChunkyString::erase (iterator *i*)

Erase a character at i.

What makes ChunkyString special is its ability to insert and erase characters quickly while remaining space efficient.

Parameters

i	iterator pointing to the character to erase

Returns

an iterator pointing to the character after the one that was deleted.

Note

constant time

Warning

invalidates all iterators except the returned iterator
erasing from an empty string is undefined behavior

4.5.3.2 iterator ChunkyString::insert (iterator *i*, char *c*)

Insert a character before the character at i.

What makes ChunkyString special is its ability to insert and erase characters quickly while remaining space efficient.

Parameters

i	iterator to specify insertion point
c	character to insert

Returns

an iterator pointing to the newly inserted character.

Note

constant time

Warning

invalidates all iterators except the returned iterator

4.5.3.3 bool ChunkyString::operator< (const ChunkyString & *rhs*) const

String comparison.

Remarks

String comparison is lexicographic, which means:

- Two strings `s1` and `s2` are compared character by character.
- The first characters that aren't equal determine the order. If the character value from `s1` is smaller than the corresponding one from `s2`, then `s1` < `s2`, and vice-versa.
- If `s1` is a prefix of `s2`, then `s1` < `s2`, and vice-versa.
- If `s1` and `s2` have exactly the same character, then neither is less than the other.
- An empty string is less than any other string, except the empty string.

See also

http://www.cplusplus.com/reference/string/string/compare/
http://en.cppreference.com/w/cpp/algorithm/lexicographical_compare

4.5.3.4 void ChunkyString::push_back (char *c*)

Inserts a character at the end of the ChunkyString.

Parameters

c	Character to insert

Note

constant time

4.5.3.5 size_t ChunkyString::size ( ) const

String size.

Note

constant time

4.5.3.6 double ChunkyString::utilization ( ) const

Average capacity of each chunk, as a fraction.

This function computes the fraction of the ChunkyString's character cells that are in use. It is defined as

$$\frac{\text{number of characters in the string}}{\text{number of chunks} \times \text{CHUNKSIZE}}$$

For reasonably sized strings (i.e., those with more than one or two characters), utilization should never fall to near one character per chunk; otherwise the data structure would be wasting too much space.

The utilization for an empty string is undefined (i.e., any value is acceptable).

Note

constant time

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

- chunkystring-past.hpp

4.6 GenericIterator Class Reference

Public Types

- typedef char **value_type**
- typedef value_type & **reference**
- typedef value_type * **pointer**
- typedef ptrdiff_t **difference_type**
- typedef std::bidirectional_iterator_tag **iterator_category**

Public Member Functions

- **GenericIterator** (IteratorWrapper *iter)
- GenericIterator & **operator=** (const GenericIterator &rhs)
- GenericIterator & **operator++** ()
- GenericIterator & **operator--** ()
- reference **operator*** () const
- pointer **operator->** () const
- bool **operator==** (const GenericIterator &rhs) const
- bool **operator!=** (const GenericIterator &rhs) const
- IteratorWrapper * **underlyingIterator** () const

Private Attributes

- UnsharedPointer< IteratorWrapper > **iter_**

Friends

- class **GenericString**

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

- string-wrapper.hpp
- string-wrapper.cpp

4.7 GenericString Class Reference

Public Types

- typedef char **value_type**
- typedef size_t **size_type**
- typedef int **difference_type**
- typedef value_type & **reference**
- typedef const value_type & **const_reference**
- typedef GenericIterator **iterator**
- typedef GenericIterator **const_iterator**
- typedef StringWrapper ∗(∗ **factoryPtr**)()

Public Member Functions

- iterator **begin** () const
- iterator **end** () const
- void **push_back** (char c)
- size_t **size** () const
- GenericString & **operator=** (const GenericString &rhs)
- GenericString & **operator+=** (const GenericString &rhs)
- bool **operator==** (const GenericString &rhs) const
- bool **operator!=** (const GenericString &rhs) const
- bool **operator**< (const GenericString &rhs) const
- iterator **insert** (iterator i, char c)
- iterator **erase** (iterator i)
- double **utilization** () const
- std::ostream & **print** (std::ostream &out) const

Static Public Member Functions

- static void **setFactory** (factoryPtr f)
- static void **loadImplementation** (const char ∗pluginFile)

Private Member Functions

- **GenericString** (StringWrapper ∗str)

Private Attributes

- UnsharedPointer< StringWrapper > **strptr_**

Static Private Attributes

- static factoryPtr **factory** = nullptr

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

- string-wrapper.hpp
- string-wrapper.cpp

4.8 ChunkyList< ELEMENT >::Iterator Class Reference

Iterator for ChunkyList.

Public Member Functions

- Iterator ()

  Default constructor, to be STL-compliant.
- Iterator & **operator++** ()
- Iterator & **operator--** ()
- char & **operator*** () const
- bool **operator==** (const Iterator &rhs) const
- bool **operator!=** (const Iterator &rhs) const

4.8.1 Detailed Description

template<typename ELEMENT>class ChunkyList< ELEMENT >::Iterator

Iterator for ChunkyList.

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

- chunkylist.hpp

4.9 ChunkyString::Iterator Class Reference

Iterator for ChunkyString.

Public Member Functions

- Iterator ()

  Default constructor, to be STL-compliant.
- Iterator & **operator++** ()
- Iterator & **operator--** ()
- char & **operator*** () const
- bool **operator==** (const Iterator &rhs) const
- bool **operator!=** (const Iterator &rhs) const

4.9.1 Detailed Description

Iterator for ChunkyString.

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

- chunkystring-past.hpp

4.10 IteratorWrapper Class Reference

Public Types

- typedef char **value_type**
- typedef value_type & **reference**
- typedef value_type * **pointer**
- typedef ptrdiff_t **difference_type**
- typedef std::bidirectional_iterator_tag **iterator_category**

Public Member Functions

- virtual IteratorWrapper * **heapcopy** ()=0
- virtual void **operator=** (const IteratorWrapper &rhs)=0
- virtual void **operator++** ()=0
- virtual void **operator--** ()=0
- virtual reference **operator*** () const =0
- virtual bool **operator==** (const IteratorWrapper &rhs) const =0
- virtual bool **operator!=** (const IteratorWrapper &rhs) const =0

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

- string-wrapper.hpp

4.11 StringWrapper Class Reference

Public Types

- typedef char **value_type**
- typedef size_t **size_type**
- typedef int **difference_type**
- typedef value_type & **reference**
- typedef const value_type & **const_reference**
- typedef GenericIterator **iterator**
- typedef GenericIterator **const_iterator**

Public Member Functions

- virtual StringWrapper * **heapcopy** ()=0
- virtual iterator **begin** () const =0
- virtual iterator **end** () const =0
- virtual void **push_back** (char c)=0
- virtual size_t **size** () const =0
- virtual void **operator=** (const StringWrapper &rhs)=0
- virtual void **operator+=** (const StringWrapper &rhs)=0
- virtual bool **operator==** (const StringWrapper &rhs) const =0
- virtual bool **operator!=** (const StringWrapper &rhs) const =0

- virtual bool **operator**< (const StringWrapper &rhs) const =0
- virtual iterator **insert** (iterator i, char c)=0
- virtual iterator **erase** (iterator i)=0
- virtual double **utilization** () const =0
- virtual std::ostream & **print** (std::ostream &out) const =0

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

- string-wrapper.hpp

4.12 TestingLogger Class Reference

Classes

- struct AssertInfo

Public Member Functions

- **TestingLogger** (std::string name)
- bool **summarize** (bool verbose=false)
- void **clear** ()
- void **abortOnFail** ()

Static Public Member Functions

- static void **check** (bool assertion, std::string description)
- template<typename Function >
  static void **checkSafely** (Function assertionFn, std::string description)

Static Public Attributes

- static TestingLogger * **currentLogger** = nullptr

Private Types

- using **const_iter** = std::map< std::string, AssertInfo >::const_iterator
- using **iter** = std::map< std::string, AssertInfo >::iterator

Private Attributes

- std::map< std::string, AssertInfo > **assertions_**
- std::string **testName_**
- bool **failedSome_**
- bool **abortOnFail_**
- TestingLogger * **previousLogger_**

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

- testing-logger.hpp
- testing-logger.cpp

4.13 UnsharedPointer< T > Class Template Reference

Public Member Functions

- **UnsharedPointer** (T *ptr=nullptr)
- **UnsharedPointer** (const UnsharedPointer &uptr)
- UnsharedPointer & **operator=** (UnsharedPointer uptr)
- T * **operator->** () const
- T & **operator*** () const
- void **swap** (UnsharedPointer< T > &other)

Private Attributes

- T * **ptr_**

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

- upointer.hpp
- upointer-private.hpp

Chapter 5

File Documentation

5.1 chunkylist.hpp File Reference

Declares the ChunkyList class.

```
#include <cstddef>
#include <list>
#include <iostream>
```

Classes

- class ChunkyList< ELEMENT >

  Efficiently represents strings where insert and erase are constant-time operations.
- struct ChunkyList< ELEMENT >::Chunk

  Holds part of a ChunkyList.
- class ChunkyList< ELEMENT >::Iterator

  Iterator for ChunkyList.

5.1.1 Detailed Description

Declares the ChunkyList class.

Authors

CS 70 given code, with additions by ... your aliases here ...

5.2 chunkystring.hpp File Reference

Declares the ChunkyString class.

```
#include <cstddef>
#include "chunkylist.hpp"
```

Typedefs

- using **ChunkyString** = ChunkyList< char >

5.2.1 Detailed Description

Declares the ChunkyString class.

Authors

CS 70 given code, with additions by ... your aliases here ...

5.3 stringtest.cpp File Reference

Tests a ChunkyList for correctness.

```
#include "testing-logger.hpp"
#include "string-wrapper.hpp"
#include <string>
#include <sstream>
#include <stdexcept>
#include <cstddef>
#include <cstdlib>
#include <cassert>
#include "signal.h"
#include "unistd.h"
```

Macros

- #define **LOAD_GENERIC_STRING** 1

Typedefs

- using **TestingString** = GenericString

Functions

- void checkUtilization (const TestingString &test, size_t divisor)

  Assuming chunks are supposed to be at least an average of 1/divisor full, checks for the lowest allowable utilization for the input string.
- bool **exampleTest** ()
- bool **constructorTest** ()
- bool **insertEmptyTest** ()
- bool **pushBackEmptyTest** ()
- bool **eraseOneCharTest** ()
- bool **eraseMultipleCharTest** ()
- bool **plusEqualTest** ()
- bool **pushBackTest** ()

- bool **iteratorBeginTest** ()
- bool **iteratorBeginOneCharTest** ()
- bool **iteratorIncrementOneCharTest** ()
- bool **iteratorIncrementTwoCharTest** ()
- bool **iteratorEndTest** ()
- bool **sandboxTest** ()
- bool **checkSizeEmpty** ()
- bool **checkSizeOne** ()
- bool **checkSizeMultiple** ()
- int main (int argc, char ∗∗argv)

  Run tests.

5.3.1 Detailed Description

Tests a ChunkyList for correctness.

5.3.2 Function Documentation

5.3.2.1 void checkUtilization (const TestingString & *test*, size_t *divisor*)

Assuming chunks are supposed to be at least an average of 1/divisor full, checks for the lowest allowable utilization for the input string.

Remarks

A helper function for affirming a TestingString's utilization is at least 1/divisor. E.g., to check for adherence to 1/2, divisor would be 2. The function does so by calculating the lowest allowable utilization for a string the length of the input string, including handling the edge cases of small strings. Since checkUtilization is not a test on its own, but rather a helper function to be used in other tests, it doesn't create its own TestingLogger object. Instead, its affirms will be associated with the TestingLogger of the calling function.

Parameters

test	TestingString to check
divisor	Fullness of chunk = 1/divisor