

NourUtilString

1.0.0

Generated by Doxygen 1.9.5



<b>1 Namespace Index</b>	<b>1</b>
1.1 Namespace List	1
<b>2 Class Index</b>	<b>3</b>
2.1 Class List	3
<b>3 File Index</b>	<b>5</b>
3.1 File List	5
<b>4 Namespace Documentation</b>	<b>7</b>
4.1 MyString Namespace Reference	7
4.1.1 Function Documentation	7
4.1.1.1 operator+()	7
4.1.1.2 operator<<()	7
4.1.1.3 operator==()	8
4.2 util Namespace Reference	8
4.2.1 Function Documentation	8
4.2.1.1 concat()	8
4.2.1.2 deepCopy()	9
4.2.1.3 operator!=() [1/2]	9
4.2.1.4 operator!=() [2/2]	9
4.2.1.5 operator<<()	9
4.2.1.6 operator==() [1/2]	9
4.2.1.7 operator==() [2/2]	10
4.2.1.8 printHeader()	10
4.2.1.9 printSubHeader()	10
<b>5 Class Documentation</b>	<b>11</b>
5.1 MyString::It< T > Class Template Reference	11
5.1.1 Detailed Description	11
5.1.2 Constructor & Destructor Documentation	11
5.1.2.1 It()	11
5.1.3 Member Function Documentation	12
5.1.3.1 operator!=()	12
5.1.3.2 operator*()	12
5.1.3.3 operator++()	12
5.1.3.4 operator==()	12
5.2 MyString::String Class Reference	12
5.2.1 Detailed Description	13
5.2.2 Constructor & Destructor Documentation	13
5.2.2.1 String() [1/3]	13
5.2.2.2 String() [2/3]	13
5.2.2.3 String() [3/3]	13
5.2.2.4 ~String()	14

5.2.3 Member Function Documentation	14
5.2.3.1 add()	14
5.2.3.2 begin()	14
5.2.3.3 compare()	14
5.2.3.4 end()	14
5.2.3.5 get()	14
5.2.3.6 getLength()	15
5.2.3.7 getText()	15
5.2.3.8 operator[]()	15
5.2.3.9 setLength()	15
5.2.3.10 setText()	15
5.2.4 Friends And Related Function Documentation	15
5.2.4.1 operator<<	15
5.3 util::string Class Reference	16
5.3.1 Detailed Description	16
5.3.2 Constructor & Destructor Documentation	17
5.3.2.1 string() [1/4]	17
5.3.2.2 string() [2/4]	17
5.3.2.3 string() [3/4]	17
5.3.2.4 string() [4/4]	17
5.3.2.5 ~string()	18
5.3.3 Member Function Documentation	18
5.3.3.1 c_str()	18
5.3.3.2 clear()	18
5.3.3.3 compare() [1/2]	18
5.3.3.4 compare() [2/2]	18
5.3.3.5 deepCopy()	19
5.3.3.6 initialize_string()	19
5.3.3.7 length()	19
5.3.3.8 operator!=( ) [1/3]	19
5.3.3.9 operator!=( ) [2/3]	20
5.3.3.10 operator!=( ) [3/3]	20
5.3.3.11 operator+( ) [1/2]	20
5.3.3.12 operator+( ) [2/2]	20
5.3.3.13 operator=( ) [1/3]	20
5.3.3.14 operator=( ) [2/3]	21
5.3.3.15 operator=( ) [3/3]	21
5.3.3.16 operator==( ) [1/3]	21
5.3.3.17 operator==( ) [2/3]	21
5.3.3.18 operator==( ) [3/3]	21
5.3.3.19 operator[]()	21
5.3.3.20 rawSize()	22

5.3.3.21 size()	22
5.3.3.22 substr()	22
5.3.4 Friends And Related Function Documentation	22
5.3.4.1 operator!= [1/2]	22
5.3.4.2 operator!= [2/2]	22
5.3.4.3 operator<<	23
5.3.4.4 operator== [1/2]	23
5.3.4.5 operator== [2/2]	23
<b>6 File Documentation</b>	<b>25</b>
6.1 main.cpp File Reference	25
6.1.1 Function Documentation	25
6.1.1.1 main()	25
6.2 main.cpp	25
6.3 main.h File Reference	28
6.3.1 Detailed Description	29
6.3.2 an invalid index by using operator []	29
6.4 main.h	29
6.5 out/build/x64-Debug/CMakeFiles/3.20.21032501-MSVC_2/CompilerIdC/CMakeCCompilerId.c File Reference	29
6.5.1 Macro Definition Documentation	30
6.5.1.1 ARCHITECTURE_ID	30
6.5.1.2 C_DIALECT	30
6.5.1.3 COMPILER_ID	30
6.5.1.4 DEC	30
6.5.1.5 HEX	31
6.5.1.6 PLATFORM_ID	31
6.5.1.7 STRINGIFY	31
6.5.1.8 STRINGIFY_HELPER	31
6.5.2 Function Documentation	31
6.5.2.1 main()	31
6.5.3 Variable Documentation	32
6.5.3.1 info_arch	32
6.5.3.2 info_compiler	32
6.5.3.3 info_language_dialect_default	32
6.5.3.4 info_platform	32
6.6 CMakeCCompilerId.c	33
6.7 out/build/x64-Debug/CMakeFiles/3.20.21032501-MSVC_2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference	41
6.7.1 Macro Definition Documentation	42
6.7.1.1 ARCHITECTURE_ID	42
6.7.1.2 COMPILER_ID	42
6.7.1.3 CXX_STD	42

6.7.1.4 DEC	42
6.7.1.5 HEX	43
6.7.1.6 PLATFORM_ID	43
6.7.1.7 STRINGIFY	43
6.7.1.8 STRINGIFY_HELPER	43
6.7.2 Function Documentation	43
6.7.2.1 main()	43
6.7.3 Variable Documentation	44
6.7.3.1 info_arch	44
6.7.3.2 info_compiler	44
6.7.3.3 info_language_dialect_default	44
6.7.3.4 info_platform	44
6.8 CMakeCXXCompilerId.cpp	45
6.9 out/build/x64-Debug/CMakeFiles/ShowIncludes/foo.h File Reference	53
6.10 foo.h	53
6.11 out/build/x64-Debug/CMakeFiles/ShowIncludes/main.c File Reference	53
6.11.1 Function Documentation	54
6.11.1.1 main()	54
6.12 main.c	54
6.13 String.cpp File Reference	54
6.14 String.cpp	54
6.15 String.h File Reference	56
6.16 String.h	56
6.17 utilstring.cpp File Reference	57
6.17.1 Detailed Description	58
6.17.2 std::string and this string class are compatible.	58
6.18 utilstring.cpp	58
6.19 utilstring.h File Reference	62
6.19.1 Detailed Description	63
6.19.2 @createdOn : 23.11.2022	63
6.19.3 Macro Definition Documentation	63
6.19.3.1 INITIAL_SIZE	63
6.20 utilstring.h	64

# Chapter 1

## Namespace Index

### 1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

<a href="#">MyString</a>	.....	<a href="#">7</a>
<a href="#">util</a>	.....	<a href="#">8</a>





## Chapter 2

# Class Index

### 2.1 Class List

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

<a href="#">MyString::lt&lt; T &gt;</a>	<a href="#">11</a>
<a href="#">MyString::String</a>	<a href="#">12</a>
<a href="#">util::string</a>	<a href="#">16</a>



## Chapter 3

# File Index

### 3.1 File List

Here is a list of all files with brief descriptions:

<a href="#">main.cpp</a>	25
<a href="#">main.h</a>	28
<a href="#">String.cpp</a>	54
<a href="#">String.h</a>	56
<a href="#">utilstring.cpp</a>	57
<a href="#">utilstring.h</a>	
Implementation of own string class	62
out/build/x64-Debug/CMakeFiles/3.20.21032501-MSVC_2/CompilerIdC/CMakeCCompilerId.c	29
out/build/x64-Debug/CMakeFiles/3.20.21032501-MSVC_2/CompilerIdCXX/CMakeCXXCompilerId.cpp	41
out/build/x64-Debug/CMakeFiles/ShowIncludes/foo.h	53
out/build/x64-Debug/CMakeFiles/ShowIncludes/main.c	53



## Chapter 4

# Namespace Documentation

### 4.1 MyString Namespace Reference

#### Classes

- class [It](#)
- class [String](#)

#### Functions

- [String operator+](#) (const [String](#) &string1, const [String](#) &string2)
- bool [operator==](#) (const [String](#) &string1, const [String](#) &string2)
- std::ostream & [operator<<](#) (std::ostream &iostream, const [MyString::String](#) &string)

#### 4.1.1 Function Documentation

##### 4.1.1.1 operator+()

```
String MyString::operator+ (  
    const String & string1,  
    const String & string2 )
```

Free operator methods

##### 4.1.1.2 operator<<()

```
std::ostream & MyString::operator<< (  
    std::ostream & iostream,  
    const MyString::String & string )
```

Friendship to implement the ostream operator <<

#### 4.1.1.3 operator==()

```
bool MyString::operator== (
    const String & string1,
    const String & string2 )
```

## 4.2 util Namespace Reference

### Classes

- class [string](#)

### Functions

- [std::ostream](#) & [operator<<](#) ([std::ostream](#) &[ostream](#), const [util::string](#) &[myString](#))
- bool [operator==](#) (const [std::string](#) &[lhsString](#), const [util::string](#) &[rhsString](#))
- bool [operator==](#) (const char \*[lhsCharArray](#), const [util::string](#) &[rhsString](#))
- bool [operator!=](#) (const [std::string](#) &[lhsString](#), const [util::string](#) &[rhsString](#))
- bool [operator!=](#) (const char \*[lhsCharArray](#), const [util::string](#) &[rhsString](#))
- void [concat](#) (char \*[rawCharTarget](#), char \*[rawCharSource](#), [size\\_t](#) [startPosition](#))
- void [deepCopy](#) (char \*[rawCharTarget](#), const char \*[rawCharSource](#), [size\\_t](#) [destStartPosition](#), [size\\_t](#) [srcEndPosition](#))
- void [printHeader](#) (const char \*[text](#))
- void [printSubHeader](#) (const char \*[text](#))

### 4.2.1 Function Documentation

#### 4.2.1.1 concat()

```
void util::concat (
    char * rawCharTarget,
    char * rawCharSource,
    size\_t startPosition )
```

fill [rawCharTarget](#) with [rawCharSource](#) starting from [startPosition](#)

Notes:

[rawCharTarget](#) contents will be changed the [rawCharTarget](#) is assumed to be big enough to hold the [rawCharSource](#) (i.e., its size is larger than or equal to that of the [rawCharSource](#))

Definition at line [357](#) of file [utilstring.cpp](#).

#### 4.2.1.2 deepCopy()

```
void util::deepCopy (
    char * rawCharTarget,
    const char * rawCharSource,
    size_t destStartPosition,
    size_t srcEndPosition )
```

Definition at line 378 of file [utilstring.cpp](#).

#### 4.2.1.3 operator"!="() [1/2]

```
bool util::operator!= (
    const char * lhsCharArray,
    const util::string & rhsString )
```

Definition at line 342 of file [utilstring.cpp](#).

#### 4.2.1.4 operator"!="() [2/2]

```
bool util::operator!= (
    const std::string & lhsString,
    const util::string & rhsString )
```

Definition at line 337 of file [utilstring.cpp](#).

#### 4.2.1.5 operator<<()

```
std::ostream & util::operator<< (
    std::ostream & iostream,
    const util::string & myString )
```

Definition at line 319 of file [utilstring.cpp](#).

#### 4.2.1.6 operator==( ) [1/2]

```
bool util::operator== (
    const char * lhsCharArray,
    const util::string & rhsString )
```

Definition at line 331 of file [utilstring.cpp](#).

#### 4.2.1.7 operator==( ) [2/2]

```
bool util::operator== (
    const std::string & lhsString,
    const util::string & rhsString )
```

Definition at line [326](#) of file [utilstring.cpp](#).

#### 4.2.1.8 printHeader()

```
void util::printHeader (
    const char * text )
```

Definition at line [394](#) of file [utilstring.cpp](#).

#### 4.2.1.9 printSubHeader()

```
void util::printSubHeader (
    const char * text )
```

Definition at line [408](#) of file [utilstring.cpp](#).



## Chapter 5

# Class Documentation

### 5.1 MyString::It< T > Class Template Reference

```
#include <String.h>
```

#### Public Member Functions

- [It](#) (T \*data)
- T & [operator\\*](#) ()
- [It](#)< T > & [operator++](#) ()
- bool [operator==](#) (const [It](#)< T > &a)
- bool [operator!=](#) (const [It](#)< T > &a)

#### 5.1.1 Detailed Description

```
template<typename T>  
class MyString::It< T >
```

Definition at line 9 of file [String.h](#).

#### 5.1.2 Constructor & Destructor Documentation

##### 5.1.2.1 It()

```
template<typename T >  
MyString::It< T >::It (  
    T * data ) [inline]
```

Definition at line 13 of file [String.h](#).

### 5.1.3 Member Function Documentation

#### 5.1.3.1 operator"!=()"

```
template<typename T >
bool MyString::It< T >::operator!= (
    const It< T > & a ) [inline]
```

Definition at line 30 of file [String.h](#).

#### 5.1.3.2 operator\*()

```
template<typename T >
T & MyString::It< T >::operator* ( ) [inline]
```

Definition at line 17 of file [String.h](#).

#### 5.1.3.3 operator++()

```
template<typename T >
It< T > & MyString::It< T >::operator++ ( ) [inline]
```

Definition at line 21 of file [String.h](#).

#### 5.1.3.4 operator==()

```
template<typename T >
bool MyString::It< T >::operator== (
    const It< T > & a ) [inline]
```

Definition at line 26 of file [String.h](#).

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

- [String.h](#)

## 5.2 MyString::String Class Reference

```
#include <String.h>
```

## Public Member Functions

- [String](#) (const char \*data)
- [String](#) (size\_t init\_size=20)
- [String](#) (const [String](#) &string)
- [~String](#) ()
- char \* [getText](#) ()
- void [setText](#) (const char \*text)
- size\_t [getLength](#) () const
- void [setLength](#) (size\_t size)
- void [add](#) (const [String](#) &text)
- char [get](#) (size\_t pos) const
- bool [compare](#) (const [String](#) &string) const
- char [operator\[\]](#) (size\_t pos)
- [It](#)< char > [begin](#) ()
- [It](#)< char > [end](#) ()

## Friends

- std::ostream & [operator<<](#) (std::ostream &iostream, const [MyString::String](#) &string)

### 5.2.1 Detailed Description

Definition at line 36 of file [String.h](#).

### 5.2.2 Constructor & Destructor Documentation

#### 5.2.2.1 String() [1/3]

```
MyString::String::String (  
    const char * data )
```

#### 5.2.2.2 String() [2/3]

```
MyString::String::String (  
    size_t init_size = 20 )
```

#### 5.2.2.3 String() [3/3]

```
MyString::String::String (  
    const String & string )
```

#### 5.2.2.4 ~String()

```
MyString::String::~~String ( )
```

### 5.2.3 Member Function Documentation

#### 5.2.3.1 add()

```
void MyString::String::add (
    const String & text )
```

#### 5.2.3.2 begin()

```
It< char > MyString::String::begin ( ) [inline]
```

Iterator

Definition at line [82](#) of file [String.h](#).

#### 5.2.3.3 compare()

```
bool MyString::String::compare (
    const String & string ) const
```

#### 5.2.3.4 end()

```
It< char > MyString::String::end ( ) [inline]
```

Definition at line [87](#) of file [String.h](#).

#### 5.2.3.5 get()

```
char MyString::String::get (
    size_t pos ) const
```

### 5.2.3.6 getLength()

```
size_t MyString::String::getLength ( ) const
```

Returns the current string length (\0 is the end of the string)

### 5.2.3.7 getText()

```
char * MyString::String::getText ( )
```

### 5.2.3.8 operator[]()

```
char MyString::String::operator[] (
    size_t pos )
```

Operators

### 5.2.3.9 setLength()

```
void MyString::String::setLength (
    size_t size )
```

Set the length of the string If the new length is smaller than its data, everything beyond will be cur If the new length is bigger, all elements will be initialized with \0

### 5.2.3.10 setText()

```
void MyString::String::setText (
    const char * text )
```

## 5.2.4 Friends And Related Function Documentation

### 5.2.4.1 operator<<

```
std::ostream & operator<< (
    std::ostream & iostream,
    const MyString::String & string ) [friend]
```

Friendship to implement the ostream operator <<

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

- [String.h](#)

## 5.3 util::string Class Reference

```
#include <utilstring.h>
```

### Public Member Functions

- [string](#) (size\_t initialSize=INITIAL\_SIZE)
- [string](#) (const char \*)
- [string](#) (const std::string &)
- [string](#) (const [string](#) &)
- [~string](#) (void)
- void [intialize\\_string](#) (size\_t length=0)
- void [deepCopy](#) (const char \*rawChar, size\_t startPosition=0)
- [string](#) [substr](#) (size\_t pos, size\_t length)
- char \* [c\\_str](#) () const
- bool [compare](#) (const char \*charArray) const
- void [clear](#) ()
- size\_t [size](#) () const
- size\_t [length](#) () const
- [string](#) & [operator+](#) (const [string](#) &rhsString)
- [string](#) & [operator+](#) (const char \*strInstance)
- [string](#) & [operator=](#) (const [string](#) &rhsString)
- [string](#) & [operator=](#) (const char \*rhsCharArray)
- [string](#) & [operator=](#) (const std::string &rhsString)
- bool [operator==](#) (const [string](#) &rhsString)
- bool [operator==](#) (const std::string &rhsString)
- bool [operator==](#) (const char \*charArray)
- bool [operator!=](#) (const [string](#) &rhsString)
- bool [operator!=](#) (const std::string &rhsString)
- bool [operator!=](#) (const char \*charArray)
- const char [operator\[\]](#) (size\_t position)

### Static Public Member Functions

- static int [compare](#) (const char \*s1, const char \*s2)
- static size\_t [rawSize](#) (const char \*rawChar)

### Friends

- std::ostream & [operator<<](#) (std::ostream &iostream, const [util::string](#) &myString)
- bool [operator==](#) (const std::string &lhsString, const [util::string](#) &rhsString)
- bool [operator==](#) (const char \*lhsCharArray, const [util::string](#) &rhsString)
- bool [operator!=](#) (const std::string &lhsString, const [util::string](#) &rhsString)
- bool [operator!=](#) (const char \*lhsCharArray, const [util::string](#) &rhsString)

#### 5.3.1 Detailed Description

Definition at line 35 of file [utilstring.h](#).

## 5.3.2 Constructor & Destructor Documentation

### 5.3.2.1 string() [1/4]

```
util::string::string (
    size_t initialSize = INITIAL_SIZE )
```

default constructor with empty initialization

Default Constructor

Definition at line 26 of file [utilstring.cpp](#).

### 5.3.2.2 string() [2/4]

```
util::string::string (
    const char * data )
```

Constructor with parameter const char\*

Constructor with char\*

Definition at line 33 of file [utilstring.cpp](#).

### 5.3.2.3 string() [3/4]

```
util::string::string (
    const std::string & data )
```

Constructor with parameter std::string

Definition at line 40 of file [utilstring.cpp](#).

### 5.3.2.4 string() [4/4]

```
util::string::string (
    const string & data )
```

Copy constructor: Creates a deep copy of a passed string

Definition at line 47 of file [utilstring.cpp](#).

#### 5.3.2.5 ~string()

```
util::string::~~string (
    void )
```

Definition at line 56 of file [utilstring.cpp](#).

### 5.3.3 Member Function Documentation

#### 5.3.3.1 c\_str()

```
char * util::string::c_str ( ) const
```

Definition at line 125 of file [utilstring.cpp](#).

#### 5.3.3.2 clear()

```
void util::string::clear ( )
```

#### 5.3.3.3 compare() [1/2]

```
bool util::string::compare (
    const char * charArray ) const
```

#### 5.3.3.4 compare() [2/2]

```
int util::string::compare (
    const char * lhsCharArray,
    const char * rhsCharArray ) [static]
```

Compares two char \* strings lexicographically This function is my own implementation of the std::strcmp() function. Note this function performs a binary comparison of the ASCII code of the characters.

##### Parameters

<i>str1</i>	primitive C string to be compared.
<i>str2</i>	primitive C string to be compared with.



**Returns**

an integral value indicating the relationship between the strings: <0 : the first character that does not match has a lower value in ptr1 than in ptr2 0 : the contents of both strings are equal >0 : the first character that does not match has a greater value in ptr1 than in ptr2

Definition at line 160 of file [utilstring.cpp](#).

**5.3.3.5 deepCopy()**

```
void util::string::deepCopy (
    const char * rawChar,
    size_t startPosition = 0 )
```

Design and implementation of a concept to extend the internal memory if [util::string](#) has to store more than the default INITIAL\_SIZE characters note that the function copy the passed char array starting from the startPosition (i.e. it can write starting from any position in the internal string buffer) startPosition default is 0

Definition at line 101 of file [utilstring.cpp](#).

**5.3.3.6 intialize\_string()**

```
void util::string::intialize_string (
    size_t length = 0 )
```

Definition at line 65 of file [utilstring.cpp](#).

**5.3.3.7 length()**

```
size_t util::string::length ( ) const
```

Definition at line 78 of file [utilstring.cpp](#).

**5.3.3.8 operator"!="() [1/3]**

```
bool util::string::operator!= (
    const char * charArray )
```

Definition at line 237 of file [utilstring.cpp](#).

#### 5.3.3.9 operator!=( ) [2/3]

```
bool util::string::operator!= (
    const std::string & rhsString )
```

Definition at line 230 of file [utilstring.cpp](#).

#### 5.3.3.10 operator!=( ) [3/3]

```
bool util::string::operator!= (
    const string & rhsString )
```

Definition at line 224 of file [utilstring.cpp](#).

#### 5.3.3.11 operator+( ) [1/2]

```
string & util::string::operator+ (
    const char * rhsString )
```

concatenating [util::string](#) and const char\*

Definition at line 275 of file [utilstring.cpp](#).

#### 5.3.3.12 operator+( ) [2/2]

```
string & util::string::operator+ (
    const string & rhsString )
```

Definition at line 252 of file [utilstring.cpp](#).

#### 5.3.3.13 operator=( ) [1/3]

```
string & util::string::operator= (
    const char * rhsCharArray )
```

Definition at line 191 of file [utilstring.cpp](#).

**5.3.3.14 operator=()** [2/3]

```
string & util::string::operator= (
    const std::string & rhsString )
```

Definition at line 196 of file [utilstring.cpp](#).

**5.3.3.15 operator=()** [3/3]

```
string & util::string::operator= (
    const string & rhsString )
```

Definition at line 186 of file [utilstring.cpp](#).

**5.3.3.16 operator==(** [1/3]

```
bool util::string::operator== (
    const char * charArray )
```

Definition at line 216 of file [utilstring.cpp](#).

**5.3.3.17 operator==(** [2/3]

```
bool util::string::operator== (
    const std::string & rhsString )
```

Definition at line 209 of file [utilstring.cpp](#).

**5.3.3.18 operator==(** [3/3]

```
bool util::string::operator== (
    const string & rhsString )
```

Definition at line 203 of file [utilstring.cpp](#).

**5.3.3.19 operator[]()**

```
const char util::string::operator[] (
    size_t position )
```

Definition at line 245 of file [utilstring.cpp](#).

#### 5.3.3.20 rawSize()

```
size_t util::string::rawSize (
    const char * rawChar ) [static]
```

Definition at line 83 of file [utilstring.cpp](#).

#### 5.3.3.21 size()

```
size_t util::string::size ( ) const
```

Get the length of the string

Definition at line 74 of file [utilstring.cpp](#).

#### 5.3.3.22 substr()

```
string util::string::substr (
    size_t pos,
    size_t length )
```

Definition at line 137 of file [utilstring.cpp](#).

### 5.3.4 Friends And Related Function Documentation

#### 5.3.4.1 operator"!= [1/2]

```
bool operator!= (
    const char * lhsCharArray,
    const util::string & rhsString ) [friend]
```

Definition at line 342 of file [utilstring.cpp](#).

#### 5.3.4.2 operator"!= [2/2]

```
bool operator!= (
    const std::string & lhsString,
    const util::string & rhsString ) [friend]
```

Definition at line 337 of file [utilstring.cpp](#).

#### 5.3.4.3 operator<<

```
std::ostream & operator<< (  
    std::ostream & iostream,  
    const util::string & myString )    [friend]
```

Definition at line 319 of file [utilstring.cpp](#).

#### 5.3.4.4 operator== [1/2]

```
bool operator== (  
    const char * lhsCharArray,  
    const util::string & rhsString )    [friend]
```

Definition at line 331 of file [utilstring.cpp](#).

#### 5.3.4.5 operator== [2/2]

```
bool operator== (  
    const std::string & lhsString,  
    const util::string & rhsString )    [friend]
```

Definition at line 326 of file [utilstring.cpp](#).

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

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



## Chapter 6

# File Documentation

### 6.1 main.cpp File Reference

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

#### Functions

- int [main](#) ()

#### 6.1.1 Function Documentation

##### 6.1.1.1 main()

```
int main ( )
```

Definition at line [88](#) of file [main.cpp](#).

### 6.2 main.cpp

[Go to the documentation of this file.](#)

```
00001 // Nour Ahmed  
00002 // Matrikal-Nr.: 5200991  
00003 // Assignment 1 - Own string Class  
00004  
00021 // #include "String.h"  
00022 #include "utilstring.h"  
00023  
00024 #include <iostream>  
00025  
00026  
00027  
00028 using namespace std;  
00029
```

```

00030
00031 //
00032 //
00033 //util::string concat(const char* lhsString, const char* rhsString) {
00034 //
00035 //
00036 //     // a temporary object to fill it with the concatenated strings
00037 //     // total size of the two strings combined
00038 //     size_t lhsSize = util::string::rawSize(lhsString);
00039 //     size_t rhsSize = util::string::rawSize(rhsString);
00040 //
00041 //     char* tempData = new char[lhsSize + rhsSize + 1];
00042 //
00043 //     // copy the passed array to the newly allocated tempData
00044 //     // copy the lhs string from the beginning of the string then the rhs string there after
00045 //     // -----
00046 //     // |0| ... |lhs size| ... |lhs size|rhs size + lhs size + 1|
00047 //     // |  lhs string  |  rhs string  | \0 |
00048 //     // -----
00049 //     // note that at the given position the lhs \0 termination will be overwritten as this copy
00050 //     // starts at its position
00051 //     int j = 0;
00052 //     while (lhsString[j] != '\0') {
00053 //         tempData[j] = lhsString[j];
00054 //         j++;
00055 //     }
00056 //
00057 //     int i = 0;
00058 //     while (rhsString[i] != '\0') {
00059 //         tempData[j] = rhsString[i];
00060 //         j++;
00061 //         i++;
00062 //     }
00063 //     tempData[j] = '\0'; // ensure destination string is null terminated
00064 //
00065 //     std::cout << " > [tempData] : " << tempData << std::endl;
00066 //
00067 //     util::string string3(tempData);
00068 //
00069 //     return string3;
00070 //
00071 //}
00072 //
00073 //
00074 //
00075
00076
00077
00078
00079
00080
00081
00082
00083
00084
00085
00086
00087
00088 int main() {
00089     char charArray[] = "text in a const char array";
00090     std::string stdString("another text in a std::string");
00091
00092     // instantiate objects
00093     util::string string1;
00094     util::string string2("initializing with const char array");
00095     util::string string3(charArray);
00096     util::string string4(stdString);
00097     util::string string5(string4);
00098
00099     // header -----
00100     /* for (int k = 1; k < 255; k++)
00101     {
00102         std::cout << "\n";
00103         std::cout << k << "\xB[" << k << "m Texting\033[0m\t\t";
00104     }*/
00105
00106
00107     util::printHeader("NourUtilString Application");
00108     cout << "- Nour Ahmed                                -" << endl;
00109     cout << "- Matrikal-Nr.: 5200991                                -" << endl;
00110     cout << "- Assignment 1 - Own string Class                        -" << endl;
00111     std::cout << "-----\n\n";
00112
00113     // Test Object Instantiation -----
00114     util::printSubHeader("Variable used for testing and their values");
00115     cout << "Variable used for testing and their values" << endl;

```



```

00116
00117     cout << " > [charArray]   charArray (size: " << util::string::rawSize(charArray) << ") : " <<
charArray << endl;
00118     cout << " > [stdString]   charArray (size: " << stdString.length() << ") : " << stdString << endl;
00119
00120     std::cout << "-----\n\n";
00121
00122     util::printSubHeader("Test object constructors and initialization");
00123     cout << " > [test with default constructors]   string1 (size: " << string1.size() << ") : " << string1
<< endl;
00124     cout << " > [test with const char*]           string2 (size: " << string2.size() << ") : " << string2
<< endl;
00125     cout << " > [test with std::string]           string3 (size: " << string3.size() << ") : " << string3
<< endl;
00126     cout << " > [test with char array]           string4 (size: " << string4.size() << ") : " << string4
<< endl;
00127     cout << " > [test with util::string]         string5 (size: " << string5.size() << ") : " << string4
<< endl;
00128     std::cout << "-----\n\n";
00129
00130
00131     // Test member methods -----
00132     util::printSubHeader("Test Member Methods");
00133     util::string temp = string2.substr(3, 5);
00134     cout << " > [test substr()]                   string2 (size: " << string2.size() << ") : " << string2 << endl;
00135     cout << " > [test substr()]                   substr(3,5) (size: " << temp.size() << ") : " << temp << endl;
00136
00137
00138
00139     std::cout << "-----\n\n";
00140
00141     // Test operators -----
00142     util::printSubHeader("Test operators");
00143
00144     string1 = string2;
00145     string2 = "more text for testing";
00146     string3 = std::string("text for std::string assignment");
00147
00148     cout << "[test = operator util::string = util::string]\n\t string1 = string2 -> string1 (size: " <<
string1.size() << ") : " << string1 << endl;
00149     cout << "[test = operator util::string = const char*]\n\t string2 = \"...\" -> string2 (size: " <<
string2.size() << ") : " << string2 << endl;
00150     cout << "[test = operator util::string = std::string]\n\t string3 = std::string(\"...\") -> string3
(size: " << string3.size() << ") : " << string3 << endl;
00151
00152     std::cout << "-----\n\n";
00153
00154     string1 = string2;
00155     cout << "[test == operator util::string == util::string]\n\t string1 == string2 -> " << ((string1
== string2) ? "true" : "false") << endl;
00156
00157     cout << "[test == operator util::string == std::string]\n\t string1 == stdString -> " << ((string1
== stdString) ? "true" : "false") << endl;
00158     cout << "[test == operator std::string == util::string]\n\t stdString == string1 -> " << ((stdString
== string1) ? "true" : "false") << endl;
00159
00160     cout << "[test == operator util::string == const char*]\n\t string1 == charArray -> " << ((string1
== charArray) ? "true" : "false") << endl;
00161     cout << "[test == operator const char* == util::string]\n\t charArray == string1 -> " << ((charArray
== string1) ? "true" : "false") << endl;
00162
00163     std::cout << "-----\n\n";
00164
00165     cout << "[test != operator util::string != util::string]\n\t string1 != string2 -> " << ((string1
!= string2) ? "true" : "false") << endl;
00166
00167     cout << "[test != operator util::string != std::string]\n\t string1 != stdString -> " << ((string1
!= stdString) ? "true" : "false") << endl;
00168     cout << "[test != operator std::string != util::string]\n\t stdString != string1 -> " << ((stdString
!= string1) ? "true" : "false") << endl;
00169
00170     cout << "[test != operator util::string != const char*]\n\t string1 != charArray -> " << ((string1
!= charArray) ? "true" : "false") << endl;
00171     cout << "[test != operator const char* != util::string]\n\t charArray != string1 -> " << ((charArray
!= string1) ? "true" : "false") << endl;
00172     std::cout << "-----\n\n";
00173
00174     cout << "[test [] operator util::string[]]" << endl;
00175     cout << "\tstring1: " << string1 << "-> string1[0]: " << string1[0] << endl;
00176     cout << "\tstring2: " << string2 << "-> string2[3]: " << string2[3] << endl;
00177     cout << "\tstring3: " << string3 << "-> string3[50]: " << string3[50] << endl;
00178     std::cout << "-----\n\n";
00179
00180     //cout << " > [util::string + const char*]   string4 + charArray: " << string4 + charArray << endl;
00181     string2 = string2 + string3;
00182     cout << "string2 (size: " << string2.size() << ") : " << string2 << endl;
00183     cout << "string3 (size: " << string3.size() << ") : " << string3 << endl;

```

```

00184     cout << " > [util::string + util::string]  string2 + string3: " << string2 + string3
00185         << "\nstring2 (size: " << string2.size() << ") : " << string2 << endl;
00186
00187
00188     std::cout << "-----\n\n";
00189
00190     cout << "[test << operator std::cout << util::string << int << std::string << char *]:\n\t"
00191         << string1 << ", "
00192         << "size: "
00193         << string1.size() << ", "
00194         << stdString << ", "
00195         << charArray
00196         << endl;
00197
00198
00199
00200     //cout << " > [util::string + const char*]  string4 + charArray: " << string4 + charArray << endl;
00201
00202     //util::string string6(string4 + charArray);
00203
00204     //cout << " > [test with util::string]          string6 (size: " << string6.size() << ") : " <<
string6 << endl;
00205
00206     std::cout << "-----\n\n";
00207
00208     //cout << " > [util::string + const char*]: " << util::concat(string4.c_str(), charArray) << endl;
00209
00210     std::cout << "-----\n\n";
00211
00212     char s1[100] = "programming ", s2[] = "is awesome";
00213     cout << "s1: " << s1 << endl;
00214     cout << "s2: " << s2 << endl;
00215
00216     /*util::concat(s1, s2, 2);
00217     cout << "s1: " << s1 << endl;
00218
00219     util::concat(s1, s2);
00220     cout << "s1: " << s1 << endl;
00221
00222     util::deepCopy(s1, s2, 3 , 10);
00223     cout << "s1: " << s1 << endl;*/
00224
00225     util::deepCopy(s1, s2);
00226     cout << "s1: " << s1 << endl;
00227
00228
00229     /*
00230     delete[] a;
00231     delete[] b;*/
00232
00233     //MyString::String string1;
00234     //MyString::String string2("Welt");
00235
00236
00237     //string1.setText("Hallo ");
00238
00239
00240
00241
00242     //cout << "String1 and String2: " << string1 << string2 << endl;
00243     //cout << "String1 + String2: " << pstring << endl;
00244     //cout << "Comparing 1 with 2: " << (string1 == string2) << endl;
00245     //cout << "Comparing 1 with 1: " << (string1 == string1) << endl;
00246     //cout << "Get: " << string1[0] << endl;
00247
00248     //MyString::It<char> it = pstring.begin();
00249     //while (it != pstring.end()) {
00250     //    cout << *it << endl;
00251     //    ++it;
00252     //}
00253
00254
00255     return 0;
00256 }

```

## 6.3 main.h File Reference

```
#include <iostream>
```

### 6.3.1 Detailed Description

=====

#### Author

: Nour Ahmed @email : [nahmed@stud.hs-bremen.de](mailto:nahmed@stud.hs-bremen.de), nour @repo : @createdOn : 23.11.2022  
 @description : Defines the entry point for the application

In this application the class [util::string](#) is used and tested. Each method and operator is tested with all possible usages (e.g., concatenating different strings etc.) Note: For this task no error handling is required. Example↵  
 : Accessing

### 6.3.2 an invalid index by using operator []

Definition in file [main.h](#).

## 6.4 main.h

[Go to the documentation of this file.](#)

```
00001 // CMakeProject1.h : Include file for standard system include files,
00002 // or project specific include files.
00003
00004 #pragma once
00005
00006 #include <iostream>
00007
00008 // TODO: Reference additional headers your program requires here.
00009
```

## 6.5 out/build/x64-Debug/CMakeFiles/3.20.21032501-MSVC\_2/Compiler↵ IdC/CMakeCCompilerId.c File Reference

### Macros

- #define [COMPILER\\_ID](#) ""
- #define [STRINGIFY\\_HELPER](#)(X) #X
- #define [STRINGIFY](#)(X) [STRINGIFY\\_HELPER](#)(X)
- #define [PLATFORM\\_ID](#)
- #define [ARCHITECTURE\\_ID](#)
- #define [DEC](#)(n)
- #define [HEX](#)(n)
- #define [C\\_DIALECT](#)

### Functions

- int [main](#) (int argc, char \*argv[])

## Variables

- char const \* [info\\_compiler](#) = "INFO" ":" "compiler[" COMPILER\_ID "]"
- char const \* [info\\_platform](#) = "INFO" ":" "platform[" PLATFORM\_ID "]"
- char const \* [info\\_arch](#) = "INFO" ":" "arch[" ARCHITECTURE\_ID "]"
- const char \* [info\\_language\\_dialect\\_default](#)

## 6.5.1 Macro Definition Documentation

### 6.5.1.1 ARCHITECTURE\_ID

```
#define ARCHITECTURE_ID
```

Definition at line [613](#) of file [CMakeCCompilerId.c](#).

### 6.5.1.2 C\_DIALECT

```
#define C_DIALECT
```

Definition at line [697](#) of file [CMakeCCompilerId.c](#).

### 6.5.1.3 COMPILER\_ID

```
#define COMPILER_ID ""
```

Definition at line [360](#) of file [CMakeCCompilerId.c](#).

### 6.5.1.4 DEC

```
#define DEC(  
    n )
```

**Value:**

```
( '0' + ((n) / 10000000) % 10 ), \
( '0' + ((n) / 1000000) % 10 ), \
( '0' + ((n) / 100000) % 10 ), \
( '0' + ((n) / 10000) % 10 ), \
( '0' + ((n) / 1000) % 10 ), \
( '0' + ((n) / 100) % 10 ), \
( '0' + ((n) / 10) % 10 ), \
( '0' + ((n) % 10) )
```

Definition at line [617](#) of file [CMakeCCompilerId.c](#).

### 6.5.1.5 HEX

```
#define HEX(  
    n )
```

#### Value:

```
( '0' + ((n)>>28 & 0xF) ), \  
( '0' + ((n)>>24 & 0xF) ), \  
( '0' + ((n)>>20 & 0xF) ), \  
( '0' + ((n)>>16 & 0xF) ), \  
( '0' + ((n)>>12 & 0xF) ), \  
( '0' + ((n)>>8  & 0xF) ), \  
( '0' + ((n)>>4  & 0xF) ), \  
( '0' + ((n)    & 0xF) )
```

Definition at line 628 of file [CMakeCCompilerId.c](#).

### 6.5.1.6 PLATFORM\_ID

```
#define PLATFORM_ID
```

Definition at line 485 of file [CMakeCCompilerId.c](#).

### 6.5.1.7 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY\_HELPER(X)
```

Definition at line 381 of file [CMakeCCompilerId.c](#).

### 6.5.1.8 STRINGIFY\_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

Definition at line 380 of file [CMakeCCompilerId.c](#).

## 6.5.2 Function Documentation

### 6.5.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Definition at line 717 of file [CMakeCCompilerId.c](#).

### 6.5.3 Variable Documentation

#### 6.5.3.1 info\_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 688 of file [CMakeCCompilerId.c](#).

#### 6.5.3.2 info\_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 367 of file [CMakeCCompilerId.c](#).

#### 6.5.3.3 info\_language\_dialect\_default

```
const char* info_language_dialect_default
```

**Initial value:**

```
=  
"INFO" ":" "dialect_default[" C_DIALECT "]"
```

Definition at line 706 of file [CMakeCCompilerId.c](#).

#### 6.5.3.4 info\_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 687 of file [CMakeCCompilerId.c](#).

## 6.6 CMakeCCompilerId.c

[Go to the documentation of this file.](#)

```

00001 #ifdef __cplusplus
00002 # error "A C++ compiler has been selected for C."
00003 #endif
00004
00005 #if defined(__18CXX)
00006 # define ID_VOID_MAIN
00007 #endif
00008 #if defined(__CLASSIC_C__)
00009 /* cv-qualifiers did not exist in K&R C */
00010 # define const
00011 # define volatile
00012 #endif
00013
00014
00015 /* Version number components: V=Version, R=Revision, P=Patch
00016    Version date components:  YYYY=Year, MM=Month, DD=Day  */
00017
00018 #if defined(__INTEL_COMPILER) || defined(__ICC)
00019 # define COMPILER_ID "Intel"
00020 # if defined(_MSC_VER)
00021 #   define SIMULATE_ID "MSVC"
00022 # endif
00023 # if defined(__GNUC__)
00024 #   define SIMULATE_ID "GNU"
00025 # endif
00026 /* __INTEL_COMPILER = VRP */
00027 # define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00028 # define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00029 # if defined(__INTEL_COMPILER_UPDATE)
00030 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00031 # else
00032 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER % 10)
00033 # endif
00034 # if defined(__INTEL_COMPILER_BUILD_DATE)
00035 /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00036 #   define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00037 # endif
00038 # if defined(_MSC_VER)
00039 /* _MSC_VER = VVRR */
00040 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00041 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00042 # endif
00043 # if defined(__GNUC__)
00044 #   define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00045 # elif defined(__GNUG__)
00046 #   define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00047 # endif
00048 # if defined(__GNUC_MINOR__)
00049 #   define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00050 # endif
00051 # if defined(__GNUC_PATCHLEVEL__)
00052 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00053 # endif
00054
00055 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00056 # define COMPILER_ID "IntelLLVM"
00057 #if defined(_MSC_VER)
00058 # define SIMULATE_ID "MSVC"
00059 #endif
00060 #if defined(__GNUC__)
00061 # define SIMULATE_ID "GNU"
00062 #endif
00063 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00064    * later. Look for 6 digit vs. 8 digit version number to decide encoding.
00065    * VVVV is no smaller than the current year when a versio is released.
00066    */
00067 #if __INTEL_LLVM_COMPILER < 1000000L
00068 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00069 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00070 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER % 10)
00071 #else
00072 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00073 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
00074 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER % 100)
00075 #endif
00076 #if defined(_MSC_VER)
00077 /* _MSC_VER = VVRR */
00078 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00079 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00080 #endif
00081 #if defined(__GNUC__)
00082 #   define SIMULATE_VERSION_MAJOR DEC(__GNUC__)

```

```

00083 #elif defined(__GNUG__)
00084 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00085 #endif
00086 #if defined(__GNUC_MINOR__)
00087 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00088 #endif
00089 #if defined(__GNUC_PATCHLEVEL__)
00090 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00091 #endif
00092
00093 #elif defined(__PATHCC__)
00094 # define COMPILER_ID "PathScale"
00095 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00096 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00097 # if defined(__PATHCC_PATCHLEVEL__)
00098 #   define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00099 # endif
00100
00101 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00102 # define COMPILER_ID "Embarcadero"
00103 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__>24 & 0x00FF)
00104 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__>16 & 0x00FF)
00105 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__ & 0xFFFF)
00106
00107 #elif defined(__BORLANDC__)
00108 # define COMPILER_ID "Borland"
00109 /* __BORLANDC__ = 0xVRR */
00110 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__>8)
00111 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00112
00113 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00114 # define COMPILER_ID "Watcom"
00115 /* __WATCOMC__ = VVRR */
00116 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00117 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00118 # if (__WATCOMC__ % 10) > 0
00119 #   define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00120 # endif
00121
00122 #elif defined(__WATCOMC__)
00123 # define COMPILER_ID "OpenWatcom"
00124 /* __WATCOMC__ = VVRP + 1100 */
00125 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00126 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00127 # if (__WATCOMC__ % 10) > 0
00128 #   define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00129 # endif
00130
00131 #elif defined(__SUNPRO_C)
00132 # define COMPILER_ID "SunPro"
00133 # if __SUNPRO_C >= 0x5100
00134 /* __SUNPRO_C = 0xVRRP */
00135 #   define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C>12)
00136 #   define COMPILER_VERSION_MINOR HEX(__SUNPRO_C>4 & 0xFF)
00137 #   define COMPILER_VERSION_PATCH HEX(__SUNPRO_C & 0xF)
00138 # else
00139 /* __SUNPRO_CC = 0xVRP */
00140 #   define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C>8)
00141 #   define COMPILER_VERSION_MINOR HEX(__SUNPRO_C>4 & 0xF)
00142 #   define COMPILER_VERSION_PATCH HEX(__SUNPRO_C & 0xF)
00143 # endif
00144
00145 #elif defined(__HP_cc)
00146 # define COMPILER_ID "HP"
00147 /* __HP_cc = VVRRPP */
00148 # define COMPILER_VERSION_MAJOR DEC(__HP_cc/10000)
00149 # define COMPILER_VERSION_MINOR DEC(__HP_cc/100 % 100)
00150 # define COMPILER_VERSION_PATCH DEC(__HP_cc % 100)
00151
00152 #elif defined(__DECC)
00153 # define COMPILER_ID "Compaq"
00154 /* __DECC_VER = VVRRTPPPP */
00155 # define COMPILER_VERSION_MAJOR DEC(__DECC_VER/10000000)
00156 # define COMPILER_VERSION_MINOR DEC(__DECC_VER/100000 % 100)
00157 # define COMPILER_VERSION_PATCH DEC(__DECC_VER % 10000)
00158
00159 #elif defined(__IBMC__) && defined(__COMPILER_VER__)
00160 # define COMPILER_ID "zOS"
00161 /* __IBMC__ = VRP */
00162 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00163 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00164 # define COMPILER_VERSION_PATCH DEC(__IBMC__ % 10)
00165
00166 #elif defined(__ibmxl__) && defined(__clang__)
00167 # define COMPILER_ID "XLclang"
00168 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00169 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)

```



```

00170 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00171 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00172
00173
00174 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ >= 800
00175 # define COMPILER_ID "XL"
00176 /* __IBMC__ = VRP */
00177 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00178 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00179 # define COMPILER_VERSION_PATCH DEC(__IBMC__ % 10)
00180
00181 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ < 800
00182 # define COMPILER_ID "VisualAge"
00183 /* __IBMC__ = VRP */
00184 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00185 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00186 # define COMPILER_VERSION_PATCH DEC(__IBMC__ % 10)
00187
00188 #elif defined(__NVCOMPILER)
00189 # define COMPILER_ID "NVHPC"
00190 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00191 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00192 # if defined(__NVCOMPILER_PATCHLEVEL__)
00193 # define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00194 # endif
00195
00196 #elif defined(__PGI)
00197 # define COMPILER_ID "PGI"
00198 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00199 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00200 # if defined(__PGIC_PATCHLEVEL__)
00201 # define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00202 # endif
00203
00204 #elif defined(_CRAYC)
00205 # define COMPILER_ID "Cray"
00206 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00207 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00208
00209 #elif defined(__TI_COMPILER_VERSION__)
00210 # define COMPILER_ID "TI"
00211 /* __TI_COMPILER_VERSION__ = VVRRRRPPP */
00212 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00213 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000 % 1000)
00214 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__ % 1000)
00215
00216 #elif defined(__FUJITSU) || defined(__FCC_VERSION) || defined(__fcc_version)
00217 # define COMPILER_ID "Fujitsu"
00218
00219 #elif defined(__ghs__)
00220 # define COMPILER_ID "GHS"
00221 /* __GHS_VERSION_NUMBER = VVVVRP */
00222 # ifdef __GHS_VERSION_NUMBER
00223 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00224 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00225 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER % 10)
00226 # endif
00227
00228 #elif defined(__TINYC__)
00229 # define COMPILER_ID "TinyCC"
00230
00231 #elif defined(__BCC__)
00232 # define COMPILER_ID "Bruce"
00233
00234 #elif defined(__SCO_VERSION__)
00235 # define COMPILER_ID "SCO"
00236
00237 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00238 # define COMPILER_ID "ARMCC"
00239 #if __ARMCC_VERSION >= 1000000
00240 /* __ARMCC_VERSION = VRRPPPP */
00241 # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00242 # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00243 # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION % 10000)
00244 #else
00245 /* __ARMCC_VERSION = VRPPPP */
00246 # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00247 # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00248 # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION % 10000)
00249 #endif
00250
00251
00252 #elif defined(__clang__) && defined(__apple_build_version__)
00253 # define COMPILER_ID "AppleClang"
00254 # if defined(_MSC_VER)
00255 # define SIMULATE_ID "MSVC"
00256 # endif

```

```

00257 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00258 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00259 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00260 # if defined(_MSC_VER)
00261     /* _MSC_VER = VVRR */
00262     # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00263     # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00264 # endif
00265 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00266
00267 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00268     # define COMPILER_ID "ARMClang"
00269     # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00270     # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00271     # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION % 10000)
00272 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00273
00274 #elif defined(__clang__)
00275     # define COMPILER_ID "Clang"
00276     # if defined(_MSC_VER)
00277         # define SIMULATE_ID "MSVC"
00278     # endif
00279     # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00280     # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00281     # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00282     # if defined(_MSC_VER)
00283         /* _MSC_VER = VVRR */
00284         # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00285         # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00286     # endif
00287
00288 #elif defined(__GNUC__)
00289     # define COMPILER_ID "GNU"
00290     # define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00291     # if defined(__GNUC_MINOR__)
00292         # define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00293     # endif
00294     # if defined(__GNUC_PATCHLEVEL__)
00295         # define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00296     # endif
00297
00298 #elif defined(_MSC_VER)
00299     # define COMPILER_ID "MSVC"
00300     /* _MSC_VER = VVRR */
00301     # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00302     # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00303     # if defined(_MSC_FULL_VER)
00304         # if _MSC_VER >= 1400
00305             /* _MSC_FULL_VER = VVRRPPPP */
00306             # define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00307         # else
00308             /* _MSC_FULL_VER = VVRRPPPP */
00309             # define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00310         # endif
00311     # endif
00312     # if defined(_MSC_BUILD)
00313         # define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00314     # endif
00315
00316 #elif defined(__VISUALDSPVERSION__) || defined(__ADSPBLACKFIN__) || defined(__ADSPTS__) ||
    defined(__ADSP21000__)
00317     # define COMPILER_ID "ADSP"
00318 #if defined(__VISUALDSPVERSION__)
00319     /* __VISUALDSPVERSION__ = 0xVVRRPP00 */
00320     # define COMPILER_VERSION_MAJOR HEX(__VISUALDSPVERSION__>>24)
00321     # define COMPILER_VERSION_MINOR HEX(__VISUALDSPVERSION__>>16 & 0xFF)
00322     # define COMPILER_VERSION_PATCH HEX(__VISUALDSPVERSION__>>8 & 0xFF)
00323 #endif
00324
00325 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00326     # define COMPILER_ID "IAR"
00327     # if defined(__VER__) && defined(__ICCARM__)
00328         # define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00329         # define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00330         # define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00331         # define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00332     # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
        defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRLSCV__) || defined(__ICCV850__) ||
        defined(__ICC8051__) || defined(__IC CSTM8__))
00333         # define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00334         # define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00335         # define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00336         # define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00337     # endif
00338
00339 #elif defined(__SDCC_VERSION_MAJOR__) || defined(SDCC)
00340     # define COMPILER_ID "SDCC"

```

```

00341 # if defined(__SDCC_VERSION_MAJOR)
00342 #   define COMPILER_VERSION_MAJOR DEC(__SDCC_VERSION_MAJOR)
00343 #   define COMPILER_VERSION_MINOR DEC(__SDCC_VERSION_MINOR)
00344 #   define COMPILER_VERSION_PATCH DEC(__SDCC_VERSION_PATCH)
00345 # else
00346 /* SDCC = VRP */
00347 #   define COMPILER_VERSION_MAJOR DEC(SDCC/100)
00348 #   define COMPILER_VERSION_MINOR DEC(SDCC/10 % 10)
00349 #   define COMPILER_VERSION_PATCH DEC(SDCC % 10)
00350 # endif
00351
00352
00353 /* These compilers are either not known or too old to define an
00354    identification macro. Try to identify the platform and guess that
00355    it is the native compiler. */
00356 #elif defined(__hpux) || defined(__hpua)
00357 #   define COMPILER_ID "HP"
00358
00359 #else /* unknown compiler */
00360 #   define COMPILER_ID ""
00361 #endif
00362
00363 /* Construct the string literal in pieces to prevent the source from
00364    getting matched. Store it in a pointer rather than an array
00365    because some compilers will just produce instructions to fill the
00366    array rather than assigning a pointer to a static array. */
00367 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00368 #ifdef SIMULATE_ID
00369 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00370 #endif
00371
00372 #ifdef __QNXNTO__
00373 char const* qnxnto = "INFO" ":" "qnxnto[]";
00374 #endif
00375
00376 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00377 char const* info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00378 #endif
00379
00380 #define STRINGIFY_HELPER(X) #X
00381 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00382
00383 /* Identify known platforms by name. */
00384 #if defined(__linux) || defined(__linux__) || defined(linux)
00385 #   define PLATFORM_ID "Linux"
00386
00387 #elif defined(__CYGWIN__)
00388 #   define PLATFORM_ID "Cygwin"
00389
00390 #elif defined(__MINGW32__)
00391 #   define PLATFORM_ID "MinGW"
00392
00393 #elif defined(__APPLE__)
00394 #   define PLATFORM_ID "Darwin"
00395
00396 #elif defined(__WIN32__) || defined(_WIN32) || defined(WIN32)
00397 #   define PLATFORM_ID "Windows"
00398
00399 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00400 #   define PLATFORM_ID "FreeBSD"
00401
00402 #elif defined(__NetBSD__) || defined(__NetBSD)
00403 #   define PLATFORM_ID "NetBSD"
00404
00405 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00406 #   define PLATFORM_ID "OpenBSD"
00407
00408 #elif defined(__sun) || defined(sun)
00409 #   define PLATFORM_ID "SunOS"
00410
00411 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00412 #   define PLATFORM_ID "AIX"
00413
00414 #elif defined(__hpux) || defined(__hpux__)
00415 #   define PLATFORM_ID "HP-UX"
00416
00417 #elif defined(__HAIKU__)
00418 #   define PLATFORM_ID "Haiku"
00419
00420 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00421 #   define PLATFORM_ID "BeOS"
00422
00423 #elif defined(__QNX__) || defined(__QNXNTO__)
00424 #   define PLATFORM_ID "QNX"
00425
00426 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00427 #   define PLATFORM_ID "Tru64"

```

```
00428
00429 #elif defined(__riscos) || defined(__riscos__)
00430 # define PLATFORM_ID "RISCos"
00431
00432 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00433 # define PLATFORM_ID "SINIX"
00434
00435 #elif defined(__UNIX_SV__)
00436 # define PLATFORM_ID "UNIX_SV"
00437
00438 #elif defined(__bsdos__)
00439 # define PLATFORM_ID "BSDOS"
00440
00441 #elif defined(_MPRAS) || defined(MPRAS)
00442 # define PLATFORM_ID "MP-RAS"
00443
00444 #elif defined(__osf) || defined(__osf__)
00445 # define PLATFORM_ID "OSF1"
00446
00447 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00448 # define PLATFORM_ID "SCO_SV"
00449
00450 #elif defined(__ultrix) || defined(__ultrix__) || defined(ULTRIX)
00451 # define PLATFORM_ID "ULTRIX"
00452
00453 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00454 # define PLATFORM_ID "Xenix"
00455
00456 #elif defined(__WATCOMC__)
00457 # if defined(__LINUX__)
00458 #   define PLATFORM_ID "Linux"
00459
00460 # elif defined(__DOS__)
00461 #   define PLATFORM_ID "DOS"
00462
00463 # elif defined(__OS2__)
00464 #   define PLATFORM_ID "OS2"
00465
00466 # elif defined(__WINDOWS__)
00467 #   define PLATFORM_ID "Windows3x"
00468
00469 # elif defined(__VXWORKS__)
00470 #   define PLATFORM_ID "VxWorks"
00471
00472 # else /* unknown platform */
00473 #   define PLATFORM_ID
00474 # endif
00475
00476 #elif defined(__INTEGRITY)
00477 # if defined(INT_178B)
00478 #   define PLATFORM_ID "Integrity178"
00479
00480 # else /* regular Integrity */
00481 #   define PLATFORM_ID "Integrity"
00482 # endif
00483
00484 #else /* unknown platform */
00485 # define PLATFORM_ID
00486
00487 #endif
00488
00489 /* For windows compilers MSVC and Intel we can determine
00490    the architecture of the compiler being used. This is because
00491    the compilers do not have flags that can change the architecture,
00492    but rather depend on which compiler is being used
00493 */
00494 #if defined(_WIN32) && defined(_MSC_VER)
00495 # if defined(_M_IA64)
00496 #   define ARCHITECTURE_ID "IA64"
00497
00498 # elif defined(_M_ARM64EC)
00499 #   define ARCHITECTURE_ID "ARM64EC"
00500
00501 # elif defined(_M_X64) || defined(_M_AMD64)
00502 #   define ARCHITECTURE_ID "x64"
00503
00504 # elif defined(_M_IX86)
00505 #   define ARCHITECTURE_ID "X86"
00506
00507 # elif defined(_M_ARM64)
00508 #   define ARCHITECTURE_ID "ARM64"
00509
00510 # elif defined(_M_ARM)
00511 #   if _M_ARM == 4
00512 #     define ARCHITECTURE_ID "ARMV4I"
00513 #   elif _M_ARM == 5
00514 #     define ARCHITECTURE_ID "ARMV5I"
```

```
00515 # else
00516 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00517 # endif
00518
00519 # elif defined(_M_MIPS)
00520 #   define ARCHITECTURE_ID "MIPS"
00521
00522 # elif defined(_M_SH)
00523 #   define ARCHITECTURE_ID "SHx"
00524
00525 # else /* unknown architecture */
00526 #   define ARCHITECTURE_ID ""
00527 # endif
00528
00529 #elif defined(__WATCOMC__)
00530 # if defined(_M_I86)
00531 #   define ARCHITECTURE_ID "I86"
00532
00533 # elif defined(_M_I86)
00534 #   define ARCHITECTURE_ID "X86"
00535
00536 # else /* unknown architecture */
00537 #   define ARCHITECTURE_ID ""
00538 # endif
00539
00540 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00541 # if defined(__ICCARM__)
00542 #   define ARCHITECTURE_ID "ARM"
00543
00544 # elif defined(__ICCRX__)
00545 #   define ARCHITECTURE_ID "RX"
00546
00547 # elif defined(__ICCRH850__)
00548 #   define ARCHITECTURE_ID "RH850"
00549
00550 # elif defined(__ICCRL78__)
00551 #   define ARCHITECTURE_ID "RL78"
00552
00553 # elif defined(__ICCRISCV__)
00554 #   define ARCHITECTURE_ID "RISCV"
00555
00556 # elif defined(__ICCAVR__)
00557 #   define ARCHITECTURE_ID "AVR"
00558
00559 # elif defined(__ICC430__)
00560 #   define ARCHITECTURE_ID "MSP430"
00561
00562 # elif defined(__ICCV850__)
00563 #   define ARCHITECTURE_ID "V850"
00564
00565 # elif defined(__ICC8051__)
00566 #   define ARCHITECTURE_ID "8051"
00567
00568 # elif defined(__ICSTM8__)
00569 #   define ARCHITECTURE_ID "STM8"
00570
00571 # else /* unknown architecture */
00572 #   define ARCHITECTURE_ID ""
00573 # endif
00574
00575 #elif defined(__ghs__)
00576 # if defined(__PPC64__)
00577 #   define ARCHITECTURE_ID "PPC64"
00578
00579 # elif defined(__ppc__)
00580 #   define ARCHITECTURE_ID "PPC"
00581
00582 # elif defined(__ARM__)
00583 #   define ARCHITECTURE_ID "ARM"
00584
00585 # elif defined(__x86_64__)
00586 #   define ARCHITECTURE_ID "x64"
00587
00588 # elif defined(__i386__)
00589 #   define ARCHITECTURE_ID "X86"
00590
00591 # else /* unknown architecture */
00592 #   define ARCHITECTURE_ID ""
00593 # endif
00594
00595 #elif defined(__TI_COMPILER_VERSION__)
00596 # if defined(__TI_ARM__)
00597 #   define ARCHITECTURE_ID "ARM"
00598
00599 # elif defined(__MSP430__)
00600 #   define ARCHITECTURE_ID "MSP430"
00601
```

```

00602 # elif defined(__TMS320C28XX__)
00603 #   define ARCHITECTURE_ID "TMS320C28x"
00604
00605 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00606 #   define ARCHITECTURE_ID "TMS320C6x"
00607
00608 # else /* unknown architecture */
00609 #   define ARCHITECTURE_ID ""
00610 # endif
00611
00612 #else
00613 #   define ARCHITECTURE_ID
00614 #endif
00615
00616 /* Convert integer to decimal digit literals. */
00617 #define DEC(n) \
00618   ('0' + ((n) / 10000000)%10), \
00619   ('0' + ((n) / 1000000)%10), \
00620   ('0' + ((n) / 100000)%10), \
00621   ('0' + ((n) / 10000)%10), \
00622   ('0' + ((n) / 1000)%10), \
00623   ('0' + ((n) / 100)%10), \
00624   ('0' + ((n) / 10)%10), \
00625   ('0' + ((n) % 10))
00626
00627 /* Convert integer to hex digit literals. */
00628 #define HEX(n) \
00629   ('0' + ((n)>>28 & 0xF)), \
00630   ('0' + ((n)>>24 & 0xF)), \
00631   ('0' + ((n)>>20 & 0xF)), \
00632   ('0' + ((n)>>16 & 0xF)), \
00633   ('0' + ((n)>>12 & 0xF)), \
00634   ('0' + ((n)>>8 & 0xF)), \
00635   ('0' + ((n)>>4 & 0xF)), \
00636   ('0' + ((n) & 0xF))
00637
00638 /* Construct a string literal encoding the version number components. */
00639 #ifdef COMPILER_VERSION_MAJOR
00640 char const info_version[] = {
00641   'I', 'N', 'F', 'O', ':',
00642   'c', 'o', 'm', 'p', 'i', 'l', 'e', 'r', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '[',
00643   COMPILER_VERSION_MAJOR,
00644 #   ifdef COMPILER_VERSION_MINOR
00645   '.', COMPILER_VERSION_MINOR,
00646 #   ifdef COMPILER_VERSION_PATCH
00647   '.', COMPILER_VERSION_PATCH,
00648 #   ifdef COMPILER_VERSION_TWEAK
00649   '.', COMPILER_VERSION_TWEAK,
00650 #   endif
00651 #   endif
00652 #   endif
00653   ']', '\0'};
00654 #endif
00655
00656 /* Construct a string literal encoding the internal version number. */
00657 #ifdef COMPILER_VERSION_INTERNAL
00658 char const info_version_internal[] = {
00659   'I', 'N', 'F', 'O', ':',
00660   'c', 'o', 'm', 'p', 'i', 'l', 'e', 'r', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '_',
00661   'i', 'n', 't', 'e', 'r', 'n', 'a', 'l', '[',
00662   COMPILER_VERSION_INTERNAL, ']', '\0'};
00663 #endif
00664
00665 /* Construct a string literal encoding the version number components. */
00666 #ifdef SIMULATE_VERSION_MAJOR
00667 char const info_simulate_version[] = {
00668   'I', 'N', 'F', 'O', ':',
00669   's', 'i', 'm', 'u', 'l', 'a', 't', 'e', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '[',
00670   SIMULATE_VERSION_MAJOR,
00671 #   ifdef SIMULATE_VERSION_MINOR
00672   '.', SIMULATE_VERSION_MINOR,
00673 #   ifdef SIMULATE_VERSION_PATCH
00674   '.', SIMULATE_VERSION_PATCH,
00675 #   ifdef SIMULATE_VERSION_TWEAK
00676   '.', SIMULATE_VERSION_TWEAK,
00677 #   endif
00678 #   endif
00679 #   endif
00680   ']', '\0'};
00681 #endif
00682
00683 /* Construct the string literal in pieces to prevent the source from
00684    getting matched. Store it in a pointer rather than an array
00685    because some compilers will just produce instructions to fill the
00686    array rather than assigning a pointer to a static array. */
00687 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "];"
00688 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "];"

```

```
00689
00690
00691
00692 #if !defined(__STDC__)
00693 # if (defined(_MSC_VER) && !defined(__clang__)) \
00694    || (defined(__ibmxl__) || defined(__IBMC__))
00695 #   define C_DIALECT "90"
00696 # else
00697 #   define C_DIALECT
00698 # endif
00699 #elif __STDC_VERSION__ >= 201000L
00700 # define C_DIALECT "11"
00701 #elif __STDC_VERSION__ >= 199901L
00702 # define C_DIALECT "99"
00703 #else
00704 # define C_DIALECT "90"
00705 #endif
00706 const char* info_language_dialect_default =
00707     "INFO" ":" "dialect_default[" C_DIALECT " ]";
00708
00709 /*-----*/
00710
00711 #ifdef ID_VOID_MAIN
00712 void main() {}
00713 #else
00714 # if defined(__CLASSIC_C__)
00715 int main(argc, argv) int argc; char *argv[];
00716 # else
00717 int main(int argc, char* argv[])
00718 # endif
00719 {
00720     int require = 0;
00721     require += info_compiler[argc];
00722     require += info_platform[argc];
00723     require += info_arch[argc];
00724 #ifdef COMPILER_VERSION_MAJOR
00725     require += info_version[argc];
00726 #endif
00727 #ifdef COMPILER_VERSION_INTERNAL
00728     require += info_version_internal[argc];
00729 #endif
00730 #ifdef SIMULATE_ID
00731     require += info_simulate[argc];
00732 #endif
00733 #ifdef SIMULATE_VERSION_MAJOR
00734     require += info_simulate_version[argc];
00735 #endif
00736 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00737     require += info_cray[argc];
00738 #endif
00739     require += info_language_dialect_default[argc];
00740     (void)argv;
00741     return require;
00742 }
00743 #endif
```

## 6.7 out/build/x64-Debug/CMakeFiles/3.20.21032501-MSVC\_2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

### Macros

- #define COMPILER\_ID ""
- #define STRINGIFY\_HELPER(X) #X
- #define STRINGIFY(X) STRINGIFY\_HELPER(X)
- #define PLATFORM\_ID
- #define ARCHITECTURE\_ID
- #define DEC(n)
- #define HEX(n)
- #define CXX\_STD \_\_cplusplus

### Functions

- int main (int argc, char \*argv[ ])

## Variables

- char const \* [info\\_compiler](#) = "INFO" ":" "compiler[" COMPILER\_ID "]"
- char const \* [info\\_platform](#) = "INFO" ":" "platform[" PLATFORM\_ID "]"
- char const \* [info\\_arch](#) = "INFO" ":" "arch[" ARCHITECTURE\_ID "]"
- const char \* [info\\_language\\_dialect\\_default](#)

## 6.7.1 Macro Definition Documentation

### 6.7.1.1 ARCHITECTURE\_ID

```
#define ARCHITECTURE_ID
```

Definition at line 598 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.1.2 COMPILER\_ID

```
#define COMPILER_ID ""
```

Definition at line 345 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.1.3 CXX\_STD

```
#define CXX_STD __cplusplus
```

Definition at line 690 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.1.4 DEC

```
#define DEC(  
    n )
```

#### Value:

```
( '0' + ((n) / 10000000) % 10 ), \
( '0' + ((n) / 1000000) % 10 ), \
( '0' + ((n) / 100000) % 10 ), \
( '0' + ((n) / 10000) % 10 ), \
( '0' + ((n) / 1000) % 10 ), \
( '0' + ((n) / 100) % 10 ), \
( '0' + ((n) / 10) % 10 ), \
( '0' + ((n) % 10) )
```

Definition at line 602 of file [CMakeCXXCompilerId.cpp](#).



### 6.7.1.5 HEX

```
#define HEX(  
    n )
```

**Value:**

```
( '0' + ((n)>>28 & 0xF) ), \  
( '0' + ((n)>>24 & 0xF) ), \  
( '0' + ((n)>>20 & 0xF) ), \  
( '0' + ((n)>>16 & 0xF) ), \  
( '0' + ((n)>>12 & 0xF) ), \  
( '0' + ((n)>>8  & 0xF) ), \  
( '0' + ((n)>>4  & 0xF) ), \  
( '0' + ((n)    & 0xF) )
```

Definition at line 613 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.1.6 PLATFORM\_ID

```
#define PLATFORM_ID
```

Definition at line 470 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.1.7 STRINGIFY

```
#define STRINGIFY(  
    X ) STRINGIFY_HELPER(X)
```

Definition at line 366 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.1.8 STRINGIFY\_HELPER

```
#define STRINGIFY_HELPER(  
    X ) #X
```

Definition at line 365 of file [CMakeCXXCompilerId.cpp](#).

## 6.7.2 Function Documentation

### 6.7.2.1 main()

```
int main (  
    int argc,  
    char * argv[] )
```

Definition at line 711 of file [CMakeCXXCompilerId.cpp](#).

## 6.7.3 Variable Documentation

### 6.7.3.1 info\_arch

```
char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]"
```

Definition at line 673 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.3.2 info\_compiler

```
char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]"
```

Definition at line 352 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.3.3 info\_language\_dialect\_default

```
const char* info_language_dialect_default
```

**Initial value:**

```
= "INFO" ":" "dialect_default["  
  "98"  
  "]"
```

Definition at line 693 of file [CMakeCXXCompilerId.cpp](#).

### 6.7.3.4 info\_platform

```
char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]"
```

Definition at line 672 of file [CMakeCXXCompilerId.cpp](#).

## 6.8 CMakeCXXCompilerId.cpp

[Go to the documentation of this file.](#)

```

00001 /* This source file must have a .cpp extension so that all C++ compilers
00002      recognize the extension without flags. Borland does not know .cxx for
00003      example. */
00004 #ifndef __cplusplus
00005 # error "A C compiler has been selected for C++."
00006 #endif
00007
00008
00009 /* Version number components: V=Version, R=Revision, P=Patch
00010      Version date components:  YYYY=Year, MM=Month, DD=Day */
00011
00012 #if defined(__COMO__)
00013 # define COMPILER_ID "Comeau"
00014 /* __COMO_VERSION__ = VRR */
00015 # define COMPILER_VERSION_MAJOR DEC(__COMO_VERSION__ / 100)
00016 # define COMPILER_VERSION_MINOR DEC(__COMO_VERSION__ % 100)
00017
00018 #elif defined(__INTEL_COMPILER) || defined(__ICC)
00019 # define COMPILER_ID "Intel"
00020 # if defined(_MSC_VER)
00021 #   define SIMULATE_ID "MSVC"
00022 # endif
00023 # if defined(__GNUC__)
00024 #   define SIMULATE_ID "GNU"
00025 # endif
00026 /* __INTEL_COMPILER = VRP */
00027 # define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00028 # define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00029 # if defined(__INTEL_COMPILER_UPDATE)
00030 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00031 # else
00032 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER % 10)
00033 # endif
00034 # if defined(__INTEL_COMPILER_BUILD_DATE)
00035 /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00036 #   define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00037 # endif
00038 # if defined(_MSC_VER)
00039 /* _MSC_VER = VVRR */
00040 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00041 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00042 # endif
00043 # if defined(__GNUC__)
00044 #   define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00045 # elif defined(__GNUG__)
00046 #   define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00047 # endif
00048 # if defined(__GNUC_MINOR__)
00049 #   define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00050 # endif
00051 # if defined(__GNUC_PATCHLEVEL__)
00052 #   define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00053 # endif
00054
00055 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00056 # define COMPILER_ID "IntelLLVM"
00057 #if defined(_MSC_VER)
00058 # define SIMULATE_ID "MSVC"
00059 #endif
00060 #if defined(__GNUC__)
00061 # define SIMULATE_ID "GNU"
00062 #endif
00063 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00064  * later. Look for 6 digit vs. 8 digit version number to decide encoding.
00065  * VVVV is no smaller than the current year when a versio is released.
00066  */
00067 #if __INTEL_LLVM_COMPILER < 1000000L
00068 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00069 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00070 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER % 10)
00071 #else
00072 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00073 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
00074 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER % 100)
00075 #endif
00076 #if defined(_MSC_VER)
00077 /* _MSC_VER = VVRR */
00078 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00079 #   define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00080 #endif
00081 #if defined(__GNUC__)
00082 #   define SIMULATE_VERSION_MAJOR DEC(__GNUC__)

```

```

00083 #elif defined(__GNUG__)
00084 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00085 #endif
00086 #if defined(__GNUC_MINOR__)
00087 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00088 #endif
00089 #if defined(__GNUC_PATCHLEVEL__)
00090 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00091 #endif
00092
00093 #elif defined(__PATHCC__)
00094 # define COMPILER_ID "PathScale"
00095 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00096 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00097 # if defined(__PATHCC_PATCHLEVEL__)
00098 #   define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00099 # endif
00100
00101 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00102 # define COMPILER_ID "Embarcadero"
00103 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__>24 & 0x00FF)
00104 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__>16 & 0x00FF)
00105 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__ & 0xFFFF)
00106
00107 #elif defined(__BORLANDC__)
00108 # define COMPILER_ID "Borland"
00109 /* __BORLANDC__ = 0xVRR */
00110 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__>8)
00111 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00112
00113 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00114 # define COMPILER_ID "Watcom"
00115 /* __WATCOMC__ = VVRR */
00116 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00117 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00118 # if (__WATCOMC__ % 10) > 0
00119 #   define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00120 # endif
00121
00122 #elif defined(__WATCOMC__)
00123 # define COMPILER_ID "OpenWatcom"
00124 /* __WATCOMC__ = VVRP + 1100 */
00125 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00126 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00127 # if (__WATCOMC__ % 10) > 0
00128 #   define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00129 # endif
00130
00131 #elif defined(__SUNPRO_CC)
00132 # define COMPILER_ID "SunPro"
00133 # if __SUNPRO_CC >= 0x5100
00134 /* __SUNPRO_CC = 0xVRRP */
00135 #   define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC>12)
00136 #   define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC>4 & 0xFF)
00137 #   define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC & 0xF)
00138 # else
00139 /* __SUNPRO_CC = 0xVRP */
00140 #   define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC>8)
00141 #   define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC>4 & 0xF)
00142 #   define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC & 0xF)
00143 # endif
00144
00145 #elif defined(__HP_aCC)
00146 # define COMPILER_ID "HP"
00147 /* __HP_aCC = VVRRPP */
00148 # define COMPILER_VERSION_MAJOR DEC(__HP_aCC/10000)
00149 # define COMPILER_VERSION_MINOR DEC(__HP_aCC/100 % 100)
00150 # define COMPILER_VERSION_PATCH DEC(__HP_aCC % 100)
00151
00152 #elif defined(__DECCXX)
00153 # define COMPILER_ID "Compaq"
00154 /* __DECCXX_VER = VVRRTPPPP */
00155 # define COMPILER_VERSION_MAJOR DEC(__DECCXX_VER/10000000)
00156 # define COMPILER_VERSION_MINOR DEC(__DECCXX_VER/100000 % 100)
00157 # define COMPILER_VERSION_PATCH DEC(__DECCXX_VER % 10000)
00158
00159 #elif defined(__IBMCPP__) && defined(__COMPILER_VER__)
00160 # define COMPILER_ID "zOS"
00161 /* __IBMCPP__ = VRP */
00162 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00163 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00164 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__ % 10)
00165
00166 #elif defined(__ibmxl__) && defined(__clang__)
00167 # define COMPILER_ID "XLclang"
00168 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00169 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)

```

```

00170 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00171 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00172
00173
00174 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ >= 800
00175 # define COMPILER_ID "XL"
00176 /* __IBMCPP__ = VRP */
00177 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00178 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00179 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__ % 10)
00180
00181 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ < 800
00182 # define COMPILER_ID "VisualAge"
00183 /* __IBMCPP__ = VRP */
00184 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00185 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00186 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__ % 10)
00187
00188 #elif defined(__NVCOMPILER)
00189 # define COMPILER_ID "NVHPC"
00190 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00191 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00192 # if defined(__NVCOMPILER_PATCHLEVEL__)
00193 #   define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00194 # endif
00195
00196 #elif defined(__PGI)
00197 # define COMPILER_ID "PGI"
00198 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00199 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00200 # if defined(__PGIC_PATCHLEVEL__)
00201 #   define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00202 # endif
00203
00204 #elif defined(_CRAYC)
00205 # define COMPILER_ID "Cray"
00206 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00207 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00208
00209 #elif defined(__TI_COMPILER_VERSION__)
00210 # define COMPILER_ID "TI"
00211 /* __TI_COMPILER_VERSION__ = VVRRRPPPP */
00212 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00213 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000 % 1000)
00214 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__ % 1000)
00215
00216 #elif defined(__FUJITSU) || defined(__FCC_VERSION) || defined(__fcc_version)
00217 # define COMPILER_ID "Fujitsu"
00218
00219 #elif defined(__ghs__)
00220 # define COMPILER_ID "GHS"
00221 /* __GHS_VERSION_NUMBER = VVVVRP */
00222 # ifdef __GHS_VERSION_NUMBER
00223 #   define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00224 #   define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00225 #   define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER % 10)
00226 # endif
00227
00228 #elif defined(__SCO_VERSION__)
00229 # define COMPILER_ID "SCO"
00230
00231 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00232 # define COMPILER_ID "ARMCC"
00233 #if __ARMCC_VERSION >= 1000000
00234   /* __ARMCC_VERSION = VRRPPPP */
00235   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00236   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00237   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION % 10000)
00238 #else
00239   /* __ARMCC_VERSION = VRPPPP */
00240   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00241   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00242   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION % 10000)
00243 #endif
00244
00245
00246 #elif defined(__clang__) && defined(__apple_build_version__)
00247 # define COMPILER_ID "AppleClang"
00248 # if defined(_MSC_VER)
00249 #   define SIMULATE_ID "MSVC"
00250 # endif
00251 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00252 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00253 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00254 # if defined(_MSC_VER)
00255   /* _MSC_VER = VVRR */
00256 #   define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)

```

```

00257 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00258 # endif
00259 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00260
00261 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00262 # define COMPILER_ID "ARMClang"
00263 # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00264 # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00265 # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION % 10000)
00266 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00267
00268 #elif defined(__clang__)
00269 # define COMPILER_ID "Clang"
00270 # if defined(_MSC_VER)
00271 # define SIMULATE_ID "MSVC"
00272 # endif
00273 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00274 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00275 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00276 # if defined(_MSC_VER)
00277 /* _MSC_VER = VVRR */
00278 # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00279 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00280 # endif
00281
00282 #elif defined(__GNUC__) || defined(__GNUG__)
00283 # define COMPILER_ID "GNU"
00284 # if defined(__GNUC__)
00285 # define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00286 # else
00287 # define COMPILER_VERSION_MAJOR DEC(__GNUG__)
00288 # endif
00289 # if defined(__GNUC_MINOR__)
00290 # define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00291 # endif
00292 # if defined(__GNUC_PATCHLEVEL__)
00293 # define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00294 # endif
00295
00296 #elif defined(_MSC_VER)
00297 # define COMPILER_ID "MSVC"
00298 /* _MSC_VER = VVRR */
00299 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00300 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00301 # if defined(_MSC_FULL_VER)
00302 # if _MSC_VER >= 1400
00303 /* _MSC_FULL_VER = VVRRPPPP */
00304 # define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00305 # else
00306 /* _MSC_FULL_VER = VVRRPPPP */
00307 # define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00308 # endif
00309 # endif
00310 # if defined(_MSC_BUILD)
00311 # define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00312 # endif
00313
00314 #elif defined(__VISUALDSPVERSION__) || defined(__ADSPBLACKFIN__) || defined(__ADSPTS__) ||
    defined(__ADSP21000__)
00315 # define COMPILER_ID "ADSP"
00316 #if defined(__VISUALDSPVERSION__)
00317 /* __VISUALDSPVERSION__ = 0xVVRRPP00 */
00318 # define COMPILER_VERSION_MAJOR HEX(__VISUALDSPVERSION__>>24)
00319 # define COMPILER_VERSION_MINOR HEX(__VISUALDSPVERSION__>>16 & 0xFF)
00320 # define COMPILER_VERSION_PATCH HEX(__VISUALDSPVERSION__>>8 & 0xFF)
00321 #endif
00322
00323 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00324 # define COMPILER_ID "IAR"
00325 # if defined(__VER__) && defined(__ICCARM__)
00326 # define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00327 # define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00328 # define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00329 # define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00330 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
    defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRL78__) || defined(__ICCV850__) ||
    defined(__ICC8051__) || defined(__ICCSSTM8__))
00331 # define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00332 # define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00333 # define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00334 # define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00335 # endif
00336
00337
00338 /* These compilers are either not known or too old to define an
00339 identification macro. Try to identify the platform and guess that
00340 it is the native compiler. */

```

```

00341 #elif defined(__hpux) || defined(__hpua)
00342 # define COMPILER_ID "HP"
00343
00344 #else /* unknown compiler */
00345 # define COMPILER_ID ""
00346 #endif
00347
00348 /* Construct the string literal in pieces to prevent the source from
00349    getting matched. Store it in a pointer rather than an array
00350    because some compilers will just produce instructions to fill the
00351    array rather than assigning a pointer to a static array. */
00352 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00353 #ifdef SIMULATE_ID
00354 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00355 #endif
00356
00357 #ifdef __QNXNTO__
00358 char const* qnxnto = "INFO" ":" "qnxnto[]";
00359 #endif
00360
00361 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00362 char const* info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00363 #endif
00364
00365 #define STRINGIFY_HELPER(X) #X
00366 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00367
00368 /* Identify known platforms by name. */
00369 #if defined(__linux) || defined(__linux__) || defined(linux)
00370 # define PLATFORM_ID "Linux"
00371
00372 #elif defined(__CYGWIN__)
00373 # define PLATFORM_ID "Cygwin"
00374
00375 #elif defined(__MINGW32__)
00376 # define PLATFORM_ID "MinGW"
00377
00378 #elif defined(__APPLE__)
00379 # define PLATFORM_ID "Darwin"
00380
00381 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00382 # define PLATFORM_ID "Windows"
00383
00384 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00385 # define PLATFORM_ID "FreeBSD"
00386
00387 #elif defined(__NetBSD__) || defined(__NetBSD)
00388 # define PLATFORM_ID "NetBSD"
00389
00390 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00391 # define PLATFORM_ID "OpenBSD"
00392
00393 #elif defined(__sun) || defined(sun)
00394 # define PLATFORM_ID "SunOS"
00395
00396 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00397 # define PLATFORM_ID "AIX"
00398
00399 #elif defined(__hpux) || defined(__hpux__)
00400 # define PLATFORM_ID "HP-UX"
00401
00402 #elif defined(__HAIKU__)
00403 # define PLATFORM_ID "Haiku"
00404
00405 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00406 # define PLATFORM_ID "BeOS"
00407
00408 #elif defined(__QNX__) || defined(__QNXNTO__)
00409 # define PLATFORM_ID "QNX"
00410
00411 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00412 # define PLATFORM_ID "Tru64"
00413
00414 #elif defined(__riscos) || defined(__riscos__)
00415 # define PLATFORM_ID "RISCos"
00416
00417 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00418 # define PLATFORM_ID "SINIX"
00419
00420 #elif defined(__UNIX_SV__)
00421 # define PLATFORM_ID "UNIX_SV"
00422
00423 #elif defined(__bsdos__)
00424 # define PLATFORM_ID "BSDOS"
00425
00426 #elif defined(_MPRAS) || defined(MPRAS)
00427 # define PLATFORM_ID "MP-RAS"

```

```

00428
00429 #elif defined(__osf) || defined(__osf__)
00430 # define PLATFORM_ID "OSF1"
00431
00432 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00433 # define PLATFORM_ID "SCO_SV"
00434
00435 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00436 # define PLATFORM_ID "ULTRIX"
00437
00438 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00439 # define PLATFORM_ID "Xenix"
00440
00441 #elif defined(__WATCOMC__)
00442 # if defined(__LINUX__)
00443 #   define PLATFORM_ID "Linux"
00444
00445 # elif defined(__DOS__)
00446 #   define PLATFORM_ID "DOS"
00447
00448 # elif defined(__OS2__)
00449 #   define PLATFORM_ID "OS2"
00450
00451 # elif defined(__WINDOWS__)
00452 #   define PLATFORM_ID "Windows3x"
00453
00454 # elif defined(__VXWORKS__)
00455 #   define PLATFORM_ID "VxWorks"
00456
00457 # else /* unknown platform */
00458 #   define PLATFORM_ID
00459 # endif
00460
00461 #elif defined(__INTEGRITY)
00462 # if defined(INT_178B)
00463 #   define PLATFORM_ID "Integrity178"
00464
00465 # else /* regular Integrity */
00466 #   define PLATFORM_ID "Integrity"
00467 # endif
00468
00469 #else /* unknown platform */
00470 # define PLATFORM_ID
00471
00472 #endif
00473
00474 /* For windows compilers MSVC and Intel we can determine
00475    the architecture of the compiler being used. This is because
00476    the compilers do not have flags that can change the architecture,
00477    but rather depend on which compiler is being used
00478 */
00479 #if defined(_WIN32) && defined(_MSC_VER)
00480 # if defined(_M_IA64)
00481 #   define ARCHITECTURE_ID "IA64"
00482
00483 # elif defined(_M_ARM64EC)
00484 #   define ARCHITECTURE_ID "ARM64EC"
00485
00486 # elif defined(_M_X64) || defined(_M_AMD64)
00487 #   define ARCHITECTURE_ID "x64"
00488
00489 # elif defined(_M_IX86)
00490 #   define ARCHITECTURE_ID "X86"
00491
00492 # elif defined(_M_ARM64)
00493 #   define ARCHITECTURE_ID "ARM64"
00494
00495 # elif defined(_M_ARM)
00496 #   if _M_ARM == 4
00497 #     define ARCHITECTURE_ID "ARMV4I"
00498 #   elif _M_ARM == 5
00499 #     define ARCHITECTURE_ID "ARMV5I"
00500 #   else
00501 #     define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00502 #   endif
00503
00504 # elif defined(_M_MIPS)
00505 #   define ARCHITECTURE_ID "MIPS"
00506
00507 # elif defined(_M_SH)
00508 #   define ARCHITECTURE_ID "SHx"
00509
00510 # else /* unknown architecture */
00511 #   define ARCHITECTURE_ID ""
00512 # endif
00513
00514 #elif defined(__WATCOMC__)

```



```

00515 # if defined(_M_I86)
00516 #   define ARCHITECTURE_ID "I86"
00517
00518 # elif defined(_M_IX86)
00519 #   define ARCHITECTURE_ID "X86"
00520
00521 # else /* unknown architecture */
00522 #   define ARCHITECTURE_ID ""
00523 # endif
00524
00525 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00526 # if defined(__ICCARM__)
00527 #   define ARCHITECTURE_ID "ARM"
00528
00529 # elif defined(__ICCRX__)
00530 #   define ARCHITECTURE_ID "RX"
00531
00532 # elif defined(__ICCRH850__)
00533 #   define ARCHITECTURE_ID "RH850"
00534
00535 # elif defined(__ICCRL78__)
00536 #   define ARCHITECTURE_ID "RL78"
00537
00538 # elif defined(__ICCRISCV__)
00539 #   define ARCHITECTURE_ID "RISCV"
00540
00541 # elif defined(__ICCAVR__)
00542 #   define ARCHITECTURE_ID "AVR"
00543
00544 # elif defined(__ICC430__)
00545 #   define ARCHITECTURE_ID "MSP430"
00546
00547 # elif defined(__ICCV850__)
00548 #   define ARCHITECTURE_ID "V850"
00549
00550 # elif defined(__ICC8051__)
00551 #   define ARCHITECTURE_ID "8051"
00552
00553 # elif defined(__ICCSTM8__)
00554 #   define ARCHITECTURE_ID "STM8"
00555
00556 # else /* unknown architecture */
00557 #   define ARCHITECTURE_ID ""
00558 # endif
00559
00560 #elif defined(__ghs__)
00561 # if defined(__PPC64__)
00562 #   define ARCHITECTURE_ID "PPC64"
00563
00564 # elif defined(__ppc__)
00565 #   define ARCHITECTURE_ID "PPC"
00566
00567 # elif defined(__ARM__)
00568 #   define ARCHITECTURE_ID "ARM"
00569
00570 # elif defined(__x86_64__)
00571 #   define ARCHITECTURE_ID "x64"
00572
00573 # elif defined(__i386__)
00574 #   define ARCHITECTURE_ID "X86"
00575
00576 # else /* unknown architecture */
00577 #   define ARCHITECTURE_ID ""
00578 # endif
00579
00580 #elif defined(__TI_COMPILER_VERSION__)
00581 # if defined(__TI_ARM__)
00582 #   define ARCHITECTURE_ID "ARM"
00583
00584 # elif defined(__MSP430__)
00585 #   define ARCHITECTURE_ID "MSP430"
00586
00587 # elif defined(__TMS320C28XX__)
00588 #   define ARCHITECTURE_ID "TMS320C28x"
00589
00590 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00591 #   define ARCHITECTURE_ID "TMS320C6x"
00592
00593 # else /* unknown architecture */
00594 #   define ARCHITECTURE_ID ""
00595 # endif
00596
00597 #else
00598 #   define ARCHITECTURE_ID
00599 #endif
00600
00601 /* Convert integer to decimal digit literals. */

```

```

00602 #define DEC(n) \
00603 ('0' + ((n) / 10000000)%10), \
00604 ('0' + ((n) / 1000000)%10), \
00605 ('0' + ((n) / 100000)%10), \
00606 ('0' + ((n) / 10000)%10), \
00607 ('0' + ((n) / 1000)%10), \
00608 ('0' + ((n) / 100)%10), \
00609 ('0' + ((n) / 10)%10), \
00610 ('0' + ((n) % 10))
00611
00612 /* Convert integer to hex digit literals. */
00613 #define HEX(n) \
00614 ('0' + ((n)>28 & 0xF)), \
00615 ('0' + ((n)>24 & 0xF)), \
00616 ('0' + ((n)>20 & 0xF)), \
00617 ('0' + ((n)>16 & 0xF)), \
00618 ('0' + ((n)>12 & 0xF)), \
00619 ('0' + ((n)>8 & 0xF)), \
00620 ('0' + ((n)>4 & 0xF)), \
00621 ('0' + ((n) & 0xF))
00622
00623 /* Construct a string literal encoding the version number components. */
00624 #ifdef COMPILER_VERSION_MAJOR
00625 char const info_version[] = {
00626 'I', 'N', 'F', 'O', ':',
00627 'c', 'o', 'm', 'p', 'i', 'l', 'e', 'r', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '[',
00628 COMPILER_VERSION_MAJOR,
00629 # ifdef COMPILER_VERSION_MINOR
00630 '.', COMPILER_VERSION_MINOR,
00631 # ifdef COMPILER_VERSION_PATCH
00632 '.', COMPILER_VERSION_PATCH,
00633 # ifdef COMPILER_VERSION_TWEAK
00634 '.', COMPILER_VERSION_TWEAK,
00635 # endif
00636 # endif
00637 # endif
00638 ']', '\0'};
00639 #endif
00640
00641 /* Construct a string literal encoding the internal version number. */
00642 #ifdef COMPILER_VERSION_INTERNAL
00643 char const info_version_internal[] = {
00644 'I', 'N', 'F', 'O', ':',
00645 'c', 'o', 'm', 'p', 'i', 'l', 'e', 'r', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '_',
00646 'i', 'n', 't', 'e', 'r', 'n', 'a', 'l', '[',
00647 COMPILER_VERSION_INTERNAL, ']', '\0'};
00648 #endif
00649
00650 /* Construct a string literal encoding the version number components. */
00651 #ifdef SIMULATE_VERSION_MAJOR
00652 char const info_simulate_version[] = {
00653 'I', 'N', 'F', 'O', ':',
00654 's', 'i', 'm', 'u', 'l', 'a', 't', 'e', '_', 'v', 'e', 'r', 's', 'i', 'o', 'n', '[',
00655 SIMULATE_VERSION_MAJOR,
00656 # ifdef SIMULATE_VERSION_MINOR
00657 '.', SIMULATE_VERSION_MINOR,
00658 # ifdef SIMULATE_VERSION_PATCH
00659 '.', SIMULATE_VERSION_PATCH,
00660 # ifdef SIMULATE_VERSION_TWEAK
00661 '.', SIMULATE_VERSION_TWEAK,
00662 # endif
00663 # endif
00664 # endif
00665 ']', '\0'};
00666 #endif
00667
00668 /* Construct the string literal in pieces to prevent the source from
00669 getting matched. Store it in a pointer rather than an array
00670 because some compilers will just produce instructions to fill the
00671 array rather than assigning a pointer to a static array. */
00672 char const* info_platform = "INFO ":" "platform[" PLATFORM_ID "];
00673 char const* info_arch = "INFO ":" "arch[" ARCHITECTURE_ID "];
00674
00675
00676
00677 #if defined(__INTEL_COMPILER) && defined(_MSVC_LANG) && _MSVC_LANG < 201403L
00678 # if defined(__INTEL_CXX11_MODE__)
00679 # if defined(__cpp_aggregate_nsdmi)
00680 # define CXX_STD 201402L
00681 # else
00682 # define CXX_STD 201103L
00683 # endif
00684 # else
00685 # define CXX_STD 199711L
00686 # endif
00687 #elif defined(_MSC_VER) && defined(_MSVC_LANG)
00688 # define CXX_STD _MSVC_LANG

```

```

00689 #else
00690 #   define CXX_STD __cplusplus
00691 #endif
00692
00693 const char* info_language_dialect_default = "INFO" ":" "dialect_default["
00694 #if CXX_STD > 202002L
00695     "23"
00696 #elif CXX_STD > 201703L
00697     "20"
00698 #elif CXX_STD >= 201703L
00699     "17"
00700 #elif CXX_STD >= 201402L
00701     "14"
00702 #elif CXX_STD >= 201103L
00703     "11"
00704 #else
00705     "98"
00706 #endif
00707 "];";
00708
00709 /*-----*/
00710
00711 int main(int argc, char* argv[])
00712 {
00713     int require = 0;
00714     require += info_compiler[argc];
00715     require += info_platform[argc];
00716 #ifdef COMPILER_VERSION_MAJOR
00717     require += info_version[argc];
00718 #endif
00719 #ifdef COMPILER_VERSION_INTERNAL
00720     require += info_version_internal[argc];
00721 #endif
00722 #ifdef SIMULATE_ID
00723     require += info_simulate[argc];
00724 #endif
00725 #ifdef SIMULATE_VERSION_MAJOR
00726     require += info_simulate_version[argc];
00727 #endif
00728 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00729     require += info_cray[argc];
00730 #endif
00731     require += info_language_dialect_default[argc];
00732     (void)argv;
00733     return require;
00734 }

```

## 6.9 out/build/x64-Debug/CMakeFiles/ShowIncludes/foo.h File Reference

### 6.10 foo.h

[Go to the documentation of this file.](#)

```
00001
```

## 6.11 out/build/x64-Debug/CMakeFiles/ShowIncludes/main.c File Reference

```
#include "foo.h"
```

### Functions

- int [main](#) ()

## 6.11.1 Function Documentation

### 6.11.1.1 main()

```
int main ( )
```

Definition at line 2 of file [main.c](#).

## 6.12 main.c

[Go to the documentation of this file.](#)

```
00001 #include "foo.h"
00002 int main() {}
```

## 6.13 String.cpp File Reference

```
#include "String.h"
```

## 6.14 String.cpp

[Go to the documentation of this file.](#)

```
00001 #include "String.h"
00002 #include <string.h>
00003
00004 namespace MyString {
00005
00009     String::String(size_t init_size) {
00010         this->data = new char[init_size];
00011         this->size = init_size;
00012
00013         if (this->data != NULL) {
00014             for (size_t i = 0; i < init_size; i++) {
00015                 data[i] = '\0';
00016             }
00017         }
00018     }
00019
00023     String::String(const char* data) {
00024
00025         if (!data) { //make sure the passed array is not NULL
00026             size_t size = strlen(data);
00027
00028             this->data = new char[size];
00029             this->size = size;
00030
00031         }
00032     }
00033
00037     String::String(const String& string) {
00038         this->size = string.getLength();
00039         this->data = new char[this->size];
00040
00041         if (this->data != NULL) {
00042             strncpy(this->data, string.data, this->size);
00043         }
00044     }
00045
00049     String::~String() {
```

```

00050         if (this->data != NULL)
00051             delete[] this->data;
00052     }
00053
00054     char* String::getText() {
00055         return data;
00056     }
00057
00058     void String::setText(const char* text) {
00059         delete[] this->data;
00060
00061         this->size = strlen(text);
00062         this->data = new char[this->size];
00063
00064         strncpy(this->data, text, this->size);
00065     }
00066
00067     size_t String::getLength() const {
00068         return strlen(this->data);
00069     }
00070
00071     void String::setLength(size_t size) {
00072         size_t old_length = this->getLength();
00073         char* old_data = this->data;
00074
00075         this->size = size;
00076         this->data = new char[size];
00077
00078         // Copy the characters (or add \0 if there are no characters anymore).
00079         for (size_t i = 0; i < size; i++) {
00080             if (i < old_length) {
00081                 this->data[i] = old_data[i];
00082             }
00083             else {
00084                 this->data[i] = '\0';
00085             }
00086         }
00087
00088         // Destroy the old data
00089         delete[] old_data;
00090
00091         // Ensure that the last element is \0
00092         this->data[size] = '\0';
00093     }
00094
00095     void String::add(const String& text) {
00096         size_t new_size = this->size + text.size;
00097         setLength(new_size);
00098
00099         size_t length = getLength();
00100         for (size_t i = length; i < new_size; i++) {
00101             this->data[i] = text.data[i - length];
00102         }
00103     }
00104
00105     char String::get(size_t pos) const {
00106         if (pos > getLength()) {
00107             return '\0';
00108         }
00109         return data[pos];
00110     }
00111
00112     bool String::compare(const String& string) const {
00113         return strcmp(this->data, string.data) == 0;
00114     }
00115
00116     char String::operator[] (size_t pos) {
00117         return this->get(pos);
00118     }
00119
00120     String operator+(const String& string1, const String& string2) {
00121         String string = String(string1);
00122         string.add(string2);
00123
00124         return string;
00125     }
00126
00127     bool operator==(const String& string1, const String& string2) {
00128         return string1.compare(string2);
00129     }
00130
00131     std::ostream& operator<<(std::ostream& ostream, const MyString::String& string) {
00132         return (ostream << string.data);
00133     }

```

```
00142 }
```

## 6.15 String.h File Reference

```
#include <iostream>
```

### Classes

- class [MyString::lt< T >](#)
- class [MyString::String](#)

### Namespaces

- namespace [MyString](#)

### Functions

- String [MyString::operator+](#) (const String &string1, const String &string2)
- bool [MyString::operator==](#) (const String &string1, const String &string2)
- std::ostream & [MyString::operator<<](#) (std::ostream &iostream, const [MyString::String](#) &string)

## 6.16 String.h

[Go to the documentation of this file.](#)

```
00001 #ifndef _H_MyString
00002 #define _H_MyString
00003
00004 #include <iostream>
00005
00006 namespace MyString {
00007
00008     template<typename T>
00009     class It {
00010     private:
00011         T* data;
00012     public:
00013         It(T* data) {
00014             this->data = data;
00015         }
00016
00017         T& operator*() {
00018             return *data;
00019         }
00020
00021         It<T>& operator++() {
00022             data++;
00023             return *this;
00024         }
00025
00026         bool operator==(const It<T>& a) {
00027             return data == a.data;
00028         }
00029
00030         bool operator!=(const It<T>& a) {
00031             return data != a.data;
00032         }
00033
00034     };
00035
00036     class String {
```

```

00040         friend std::ostream& operator<<(std::ostream& iostream, const MyString::String& string);
00041
00042     private:
00043         char* data;
00044         size_t size;
00045     public:
00046         String(const char* data);
00047
00048         String(size_t init_size = 20);
00049
00050         String(const String& string);
00051
00052         ~String();
00053
00054         char* getText();
00055
00056         void setText(const char* text);
00057
00061         size_t getLength() const;
00062
00068         void setLength(size_t size);
00069
00070         void add(const String& text);
00071
00072         char get(size_t pos) const;
00073
00074         bool compare(const String& string) const;
00075
00078         char operator[] (size_t pos);
00079
00082         It<char> begin() {
00083             It<char> it(data);
00084             return it;
00085         }
00086
00087         It<char> end()
00088         {
00089             It<char> it(data + size);
00090             return it;
00091         }
00092     };
00093
00096     String operator+(const String& string1, const String& string2);
00097
00098     bool operator==(const String& string1, const String& string2);
00099
00100     std::ostream& operator<<(std::ostream& iostream, const MyString::String& string);
00101 }
00102 #endif

```

## 6.17 utilstring.cpp File Reference

```
#include "utilstring.h"
```

### Namespaces

- namespace [util](#)

### Functions

- `std::ostream & util::operator<< (std::ostream &iostream, const util::string &myString)`
- `bool util::operator== (const std::string &lhsString, const util::string &rhsString)`
- `bool util::operator== (const char *lhsCharArray, const util::string &rhsString)`
- `bool util::operator!= (const std::string &lhsString, const util::string &rhsString)`
- `bool util::operator!= (const char *lhsCharArray, const util::string &rhsString)`
- `void util::concat (char *rawCharTarget, char *rawCharSource, size_t startPosition)`
- `void util::deepCopy (char *rawCharTarget, const char *rawCharSource, size_t destStartPosition, size_t src↵  
EndPosition)`
- `void util::printHeader (const char *text)`
- `void util::printSubHeader (const char *text)`

### 6.17.1 Detailed Description

#### Author

: Nour Ahmed @email : [nahmed@stud.hs-bremen.de](mailto:nahmed@stud.hs-bremen.de), [nourbrm02@gmail.com](mailto:nourbrm02@gmail.com) @repo ↔  
 : [https://github.com/nouremara/cpp\\_mystring](https://github.com/nouremara/cpp_mystring) @createdOn : 23.11.2022 @description  
 : implementation of own string class

this file presents an implementation of a class named string. This class behavior will be similar to the std::string and both

### 6.17.2 std::string and this string class are compatible.

Definition in file [utilstring.cpp](#).

## 6.18 utilstring.cpp

[Go to the documentation of this file.](#)

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 1 - Own String Class
00004
00018 #include "utilstring.h"
00019
00020 namespace util {
00021     /*=====
00022     *                               Constructors                               *
00023     *=====*/
00024
00026     string::string(size_t intialSize) {
00027         initialize_string(intialSize);
00028     }
00029     //-----
00030
00031
00033     string::string(const char* data) {
00034         initialize_string(rawSize(data)); // ensure string is initialized before using it
00035         deepCopy(data); // copy passed array to the string
00036     }
00037     //-----
00038
00040     string::string(const std::string& data) {
00041         initialize_string(rawSize(data.c_str())); // ensure string is initialized before using it
00042         deepCopy(data.c_str()); // copy passed array to the string
00043     }
00044     //-----
00045
00047     string::string(const string& data) {
00048         initialize_string(rawSize(data.c_str())); // ensure string is initialized before using it
00049         deepCopy(data.c_str()); // copy passed array to the string
00050     }
00051
00052
00053     /*=====
00054     *                               Destructor                               *
00055     *=====*/
00056     string::~string(void) {
00057         delete[] internal_buffer;
00058     }
00059
00060
00061     /*=====
00062     *                               Methods                               *
00063     *=====*/
00064
00065     void string::intialize_string(size_t length) {
00066         internal_buffer = new char[length + 1];
```



```

00067         buffer_size = length + 1;
00068
00069         // initialize an empty string
00070         internal_buffer[0] = '\0';
00071     }
00072     //-----
00073
00074     size_t string::size() const {
00075         return rawSize(internal_buffer);
00076     }
00077
00078     size_t string::length() const {
00079         return rawSize(internal_buffer);
00080     }
00081     //-----
00082
00083     size_t string::rawSize(const char* rawChar) {
00084         size_t length = 0;
00085         while (rawChar[length] != '\0') {
00086             length++;
00087         }
00088
00089         return length;
00090     }
00091     //-----
00092
00093     void string::deepCopy(const char* rawChar, size_t startPosition) {
00094         //check if internalData is of enough size to accommodate the passed array
00095         size_t rawCharSize = rawSize(rawChar);
00096         if (rawCharSize > size()) { // more space is needed
00097             //delete current internalData
00098             delete[] internal_buffer;
00099
00100             // re-initialize the string with the required size
00101             intialize_string(rawCharSize);
00102         }
00103
00104         //copy the passed array to the newly allocated internalData
00105         int j = startPosition;
00106         while (rawChar[j] != '\0') {
00107             internal_buffer[j] = rawChar[j];
00108             j++;
00109         }
00110
00111         internal_buffer[j] = '\0'; // ensure destination string is null terminated
00112         //string_size = rawCharSize; // set string size to the new one
00113     }
00114     //-----
00115
00116     char* string::c_str() const {
00117         return internal_buffer;
00118     }
00119     //-----
00120
00121     // Returns a pointer to an array that contains a null-terminated
00122     // sequence of characters(i.e., a C-string) representing the current
00123     // value of the string object.
00124     // Returns a substring object of type util::string which
00125     // starts at pos. Parameter length specifies the amount of
00126     // characters of the new util::string to be returned.
00127     string string::substr(size_t pos, size_t length) {
00128         string substring(length);
00129
00130         util::deepCopy(substring.c_str(), internal_buffer+ pos, 0, length);
00131
00132         return substring;
00133     }
00134     //-----
00135
00136     int string::compare(const char* lhsCharArray, const char* rhsCharArray)
00137     {
00138         // convert both pointers from 'char*' to 'unsigned char*'
00139         // needed for the difference calculations
00140         const unsigned char* p1 = (const unsigned char*)lhsCharArray;
00141         const unsigned char* p2 = (const unsigned char*)rhsCharArray;
00142
00143         // check if characters differ, or end of the first string (a terminating null) is reached
00144         while (*p1 && *p1 == *p2) {
00145             // proceed to the next pair of characters
00146             ++p1, ++p2;
00147         }
00148     }
00149
00150
00151
00152

```

```

00173         // return the ASCII difference
00174         return (*p1 > *p2) - (*p2 > *p1);
00175     }
00176
00177
00178     /*=====
00179     *                               Operators                               *
00180     *=====*/
00181
00182
00183
00184     //-----
00185
00186     string& string::operator=(const string& rhsString) {
00187         deepCopy(rhsString.c_str());
00188         return *this;
00189     }
00190
00191     string& string::operator=(const char* rhsCharArray) {
00192         deepCopy(rhsCharArray);
00193         return *this;
00194     }
00195
00196     string& string::operator=(const std::string& rhsString) {
00197         deepCopy(rhsString.c_str());
00198         return *this;
00199     }
00200     //-----
00201
00202
00203     bool string::operator==(const string& rhsString) {
00204         //if (string_size != rhsString.size()) return false;
00205         // note that compare returns 0 when the two strings are equal
00206         return !compare(internal_buffer, rhsString.c_str());
00207     }
00208
00209     bool string::operator==(const std::string& rhsString) {
00210         //if (string_size != rhsString.size()) return false;
00211
00212         // note that compare returns 0 when the two strings are equal
00213         return !compare(internal_buffer, rhsString.c_str());
00214     }
00215
00216     bool string::operator==(const char* charArray) {
00217         //if (string_size != rhsString.size()) return false;
00218
00219         // note that compare returns 0 when the two strings are equal
00220         return !compare(internal_buffer, charArray);
00221     }
00222     //-----
00223
00224     bool string::operator!=(const string& rhsString) {
00225         //if (string_size != rhsString.size()) return false;
00226         // note that compare returns 0 when the two strings are equal
00227         return compare(internal_buffer, rhsString.c_str());
00228     }
00229
00230     bool string::operator!=(const std::string& rhsString) {
00231         //if (string_size != rhsString.size()) return false;
00232
00233         // note that compare returns 0 when the two strings are equal
00234         return compare(internal_buffer, rhsString.c_str());
00235     }
00236
00237     bool string::operator!=(const char* charArray) {
00238         //if (string_size != rhsString.size()) return false;
00239
00240         // note that compare returns 0 when the two strings are equal
00241         return compare(internal_buffer, charArray);
00242     }
00243     //-----
00244
00245     const char string::operator[](size_t position) {
00246         if (position > size()) return '\0';
00247         return internal_buffer[position];
00248     }
00249     //-----
00250
00251
00252     string& string::operator+(const string& rhsString) {
00253         size_t total_size = size() + rhsString.size() + 1;
00254         char* temp = new char[total_size];
00255
00256         util::deepCopy(temp, internal_buffer, 0);
00257         util::deepCopy(temp, rhsString.c_str(), size());
00258
00259         delete[] internal_buffer;

```

```

00260
00261     internal_buffer = temp;
00262     buffer_size = total_size;
00263     std::cout << "\n\nbuffer_size " << buffer_size << std::endl;
00264     std::cout << "(size: " << size() << ") : " << internal_buffer << std::endl;
00265
00266     return *this;
00267 }
00268 //-----
00269
00270 string& string::operator+(const char* rhsString) {
00271
00272     // a temporary object to fill it with the concatenated strings
00273     // total size of the two strings combined
00274     size_t rhsSize = rawSize(rhsString);
00275     char* tempData = new char[size() + rhsSize + 1];
00276
00277     // copy the passed array to the newly allocated tempData
00278     // copy the lhs string from the beginning of the string then the rhs string there after
00279     // -----
00280     // |0| ... |lhs size| ... |lhs size|rhs size + lhs size + 1|
00281     // |  lhs string  |  rhs string  | \0 |
00282     // -----
00283     // note that at the given position the lhs \0 termination will be overwritten as this copy
00284     // starts at its position
00285
00286     int j = 0;
00287     while (internal_buffer[j] != '\0') {
00288         tempData[j] = internal_buffer[j];
00289         j++;
00290     }
00291
00292     int i = 0;
00293     while (rhsString[i] != '\0') {
00294         tempData[j] = rhsString[i];
00295         j++;
00296         i++;
00297     }
00298     tempData[j] = '\0'; // ensure destination string is null terminated
00299
00300     std::cout << " > [tempData] : " << tempData << std::endl;
00301
00302     util::string temp;
00303     //delete[] tempData;
00304
00305     return temp;
00306 }
00307
00308
00309
00310
00311
00312
00313
00314
00315     /*=====
00316     * non-member functions and operator methods for the cases util::string is on the rhs *
00317     *=====*/
00318
00319     std::ostream& operator<<(std::ostream& ostream, const util::string& myString) {
00320         return (ostream << myString.c_str());
00321     }
00322
00323     //-----
00324
00325
00326     bool operator==(const std::string& lhsString, const util::string& rhsString) {
00327         // note that compare returns 0 when the two strings are equal
00328         return !util::string::compare(lhsString.c_str(), rhsString.c_str());
00329     }
00330
00331     bool operator==(const char* lhsCharArray, const util::string& rhsString) {
00332         // note that compare returns 0 when the two strings are equal
00333         return !util::string::compare(lhsCharArray, rhsString.c_str());
00334     }
00335     //-----
00336
00337     bool operator!=(const std::string& lhsString, const util::string& rhsString) {
00338         // note that compare returns 0 when the two strings are equal
00339         return util::string::compare(lhsString.c_str(), rhsString.c_str());
00340     }
00341
00342     bool operator!=(const char* lhsCharArray, const util::string& rhsString) {
00343         // note that compare returns 0 when the two strings are equal
00344         return util::string::compare(lhsCharArray, rhsString.c_str());
00345     }
00346     //-----
00347
00348
00349     void concat(char* rawCharTarget, char* rawCharSource, size_t startPosition) {

```

```

00358         // if the startPosition is not given (we have its default -1)
00359         // do a normal concatenation of the two strings)
00360         // note that at the given position the lhs \0 termination will be
00361         // overwritten as this copy starts at its position
00362         if (startPosition == -1) {
00363             startPosition = util::string::rawSize(rawCharTarget);
00364         }
00365
00366         // deep copy rawCharSource into rawCharTarget beginning at startPosition
00367         for (size_t j = 0; rawCharSource[j] != '\0'; ++j, ++startPosition) {
00368             rawCharTarget[startPosition] = rawCharSource[j];
00369         }
00370
00371         // ensure destination string is null terminated
00372         rawCharTarget[startPosition] = '\0';
00373     }
00374
00375
00376     // destStartPosition default is to first location of the destRawChar
00377     // srcEndPosition default is to last character (before the \0) of the srcRawChar
00378     void deepCopy(char* rawCharTarget, const char* rawCharSource, size_t destStartPosition, size_t
srcEndPosition){
00379         // check and adjust for default values
00380         destStartPosition = (destStartPosition == -1) ? util::string::rawSize(rawCharTarget) :
destStartPosition;
00381         srcEndPosition = (srcEndPosition == -1) ? util::string::rawSize(rawCharSource):
srcEndPosition;
00382
00383         // deep copy rawCharSource into rawCharTarget beginning at startPosition
00384         // for (size_t j = 0; rawCharSource[j] != '\0'; ++j, ++destStartPosition) {
00385         for (size_t j = 0; j < srcEndPosition; ++j, ++destStartPosition) {
00386             rawCharTarget[destStartPosition] = rawCharSource[j];
00387         }
00388
00389         // ensure destination string is null terminated
00390         rawCharTarget[destStartPosition] = '\0';
00391     }
00392
00393     //-----
00394     void printHeader(const char* text){
00395         size_t spaces_needed = (80 - util::string::rawSize(text))/2 -2;
00396
00397         std::cout << "\033[96m"; // set text and background colors
00398         std::cout <<
"-----\n";
00399         for (int i = 0; i < spaces_needed; ++i) { std::cout << " "; }
00400         std::cout << text;
00401         for (int i = 0; i < spaces_needed; ++i) { std::cout << " "; }
00402         std::cout << "\n";
00403         std::cout <<
"-----\n";
00404         std::cout << "\033[0m"; // reset text and background colors
00405     }
00406
00407     void printSubHeader(const char* text) {
00408         std::cout << "\033[32m"; // set text and background colors
00409         std::cout << text;
00410         std::cout << "\033[0m\n"; // reset text and background colors
00411     }
00412
00413
00414
00415 } // namespace util

```

## 6.19 utilstring.h File Reference

implementation of own string class.

```

#include <cstdint>
#include <iostream>
#include <string>

```

### Classes

- class `util::string`

## Namespaces

- namespace [util](#)

## Macros

- `#define` [INITIAL\\_SIZE](#) 10

## Functions

- void [util::concat](#) (char \*rawCharTarget, char \*rawCharSource, size\_t startPosition)
- void [util::deepCopy](#) (char \*rawCharTarget, const char \*rawCharSource, size\_t destStartPosition, size\_t src↵ EndPosition)
- void [util::printHeader](#) (const char \*text)
- void [util::printSubHeader](#) (const char \*text)

### 6.19.1 Detailed Description

implementation of own string class.

=====

this file presents an implementation of a class named string. This class behavior will be similar and compatible to the std::string. This file contains the prototypes for the class, its methods and eventually any macros, constants, or global variables you will need to use it.

#### Author

: Nour Ahmed @email : [nahmed@stud.hs-bremen.de](mailto:nahmed@stud.hs-bremen.de), nour @repo : [https://github.com/nouremara/cpp\\_mystring](https://github.com/nouremara/cpp_mystring)↵

### 6.19.2 @createdOn : 23.11.2022

Definition in file [utilstring.h](#).

### 6.19.3 Macro Definition Documentation

#### 6.19.3.1 INITIAL\_SIZE

```
#define INITIAL_SIZE 10
```

Definition at line 33 of file [utilstring.h](#).

## 6.20 utilstring.h

[Go to the documentation of this file.](#)

```

00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 1 - Own string Class
00004
00020 // see: https://gist.github.com/philippeimboeck/099e540d800063e3e6ec
00021 // see: https://codereview.stackexchange.com/questions/98329/stdstring-implementation
00022 // see: https://en.wikipedia.org/wiki/Snake_case
00023
00024 #ifndef UTILSTRING_H
00025 #define UTILSTRING_H
00026
00027 #include <cstdint>
00028 #include <iostream>
00029 #include <string>
00030
00031 namespace util {
00032     // Initially, the class shall provide memory for 10 printable characters
00033     #define INITIAL_SIZE 10
00034
00035     class string {
00036     public:
00037         char* internal_buffer;
00038         size_t buffer_size;
00039         //size_t string_size;
00040
00041         /*===== Constructors =====*/
00042         string(size_t intialSize= INITIAL_SIZE);
00043         string(const char*);
00044         string(const std::string&);
00045         string(const string&);
00046         /*===== Destructor =====*/
00047         ~string(void);
00048
00049         /*===== Methods =====*/
00050
00051         void initialize_string(size_t length = 0);
00052         void deepCopy(const char* rawChar, size_t startPosition = 0);
00053         //void concat(char* rawCharTarget, const char* rawCharSource, size_t startPosition = 0);
00054
00055         // Returns a pointer to an array that contains a null-terminated
00056         // sequence of characters(i.e., a C-string) representing the current
00057         // value of the string object.
00058         // Returns a substring object of type util::string which
00059         // starts at pos. Parameter length specifies the amount of
00060         // characters of the new util::string to be returned.
00061         string substr(size_t pos, size_t length);
00062
00063         // Allows raw access to the internal C-string (through its char* pointer)
00064         char* c_str() const;
00065
00066         bool compare(const char* charArray) const;
00067         static int compare(const char* s1, const char* s2);
00068
00069         // Clears your string object
00070         // Erases the contents of the string, which becomes an empty string(with
00071         // a length of 0 characters).
00072         void clear();
00073
00074         size_t size() const;
00075         static size_t rawSize(const char* rawChar);
00076
00077         // Returns the amount of characters of your string excluding \0.
00078         // Might be smaller than the actual reserved memory.
00079         size_t length() const;
00080
00081         /*===== Operators =====*/
00082         string& operator+(const string& rhsString);
00083         string& operator+(const char* strInstance);
00084
00085         string& operator=(const string& rhsString);
00086         string& operator=(const char* rhsCharArray);
00087         string& operator=(const std::string& rhsString);
00088
00089         bool operator==(const string& rhsString);
00090         bool operator==(const std::string& rhsString);
00091         bool operator==(const char* charArray);

```

```

00100
00101     bool operator!=(const string& rhsString);
00102     bool operator!=(const std::string& rhsString);
00103     bool operator!=(const char* charArray);
00104
00105
00106     const char operator[](size_t position);
00107
00108
00109     /*=====
00110     *           Non-member function overloads           *
00111     *=====*/
00112     // Free operator methods for the cases util::string is on the rhs
00113     // Friendship enables access to private members
00114     friend std::ostream& operator<<(std::ostream& iostream, const util::string& myString);
00115
00116     friend bool operator==(const std::string& lhsString, const util::string& rhsString);
00117     friend bool operator==(const char* lhsCharArray, const util::string& rhsString);
00118
00119     friend bool operator!=(const std::string& lhsString, const util::string& rhsString);
00120     friend bool operator!=(const char* lhsCharArray, const util::string& rhsString);
00121 };
00122
00123     /*=====
00124     *           Some Utility functions           *
00125     *=====*/
00126     void concat(char* rawCharTarget, char* rawCharSource, size_t startPosition = -1);
00127
00128     void deepCopy(char* rawCharTarget, const char* rawCharSource, size_t destStartPosition = -1,
00129 size_t srcEndPosition = -1);
00129
00130
00131
00132 //-----
00133     void printHeader(const char* text);
00134     void printSubHeader(const char* text);
00135
00136
00137
00138 } // namespace util
00139
00140
00141 #endif /* UTILSTRING_H */

```

