Assignment 1 – Own string Class

NourUtilString

v1.0.0

Nour Ahmed
Matrikal-Nr.: 5200991

Implementation of Own String Class

This report presents an implementation of a class named string. This class behavior will be similar and compatible to the std::string. Here the prototypes for the class, its methods and eventually any macros, constants, or global variables needed are descried

Generated by Doxygen 1.9.5

Chapter 1

my_cpp_string

Implementation of Own String Class

In this presents an implementation of a class named `util::string`. This class behavior is similar to the `std::string` and both `std::string` and this `util::string` class are compatible.

Full and detailed examples of uses and tests of the class `util::string` are given in the `main.cpp` file. Each method and operator is very carefully tested (e.g., concatenating different strings, ..., etc).

An example test run is shown in the following screenshot:

Note that the **terminal output is colored** (using `ANSI escape codes`) for better visibility.

Doxygen generated documentation (in html and LaTeX formats) can be found at `doc/html/index.html` and `doc/latex/refman.pdf`, respectively. The configuration file `Doxyfile` is used with the Doxygen generation tool.

The following design and implementation criteria are followed:

- **No C/C++ standard functions or classes are used** to realize `util::string` class. This include, e.g., `strcmp`, `strlen` and of course using `std::string` as an internal representation of `util::string`.
  - This means own functions/methods are developed and implemented to calculate the length of a `char*`, to compare character sequences or to copy them full or partially.
- For now, **no error handling** (e.g., accessing an invalid index by using operator []) is implemented. This may be done later.
  - **Use this class at your own risk** :).
- The code follows `LLVM Coding Standards`.
- The *`sanke_case`* naming convention is used for variable and function names (with few exceptions).
- **Use this class at your own risk** :).

1.1 General Functionality

- The class `util::string` is implemented inside the two files `utilstring.cpp` and `utilstring.h`
- Class string is within the namespace `util`
- The memory management is done by using a pointer (`internal_buffer`) pointing to the data type char. `char*` are (normally) null terminated. This means, that the last character is always a `\0` (NULL character) which marks the end of a char sequence. This character is never printed as it just allows for detecting whether the end of a char sequence has been reached. The string is always null-terminate (internally!)
- Initially, the class provides memory for 10 printable characters. Note that this default value is provided by the constant `INITIAL_SIZE` (defined at the top of `utilstring.h`). It can be changed if another value is desired.
- A relatively simple concept is designed and implemented to extend the internal memory if `util::string` has to store more than 10 characters.

1.2 Constructors

The following constructors are implemented:

- `string()` : Default constructor with empty initialization
- `string(size_t intialSize)` : constructor with parameter for the initial memory size to initialization with.
- `string(const string&)` : Copy constructor: Creates a deep copy of a passed string.
- `string(const char*)` : Constructor with parameter `const char*`.
- `string(const std::string&)` : Constructor with parameter std::string.

1.3 Operators

The following operators are implemented:

- Operator + and += such that string, std::string and (const) char* can be added
- Assignment operator = such that string, std::string and (const) char* can be assigned
- Comparison operators == and != such that comparisons with `util::string`, `std::string` and `(const) char*` are possible. With respect to the last two cases, `std::string` and `const char*` may both be LHS as well as RHS arguments.
- operator [] to access individual characters of `util::string` object.
- Streaming operator << to print `util::string` to `std::cout`.

1.4 Methods

The following methods are implemented:

- `clear()`: Clears your string object.

  It erases the contents of the string, which becomes an empty string (with a length of 0 characters).

- `substr(pos, length)`: Returns a substring object of type util::string which starts at pos. Parameter length specifies the amount of characters of the new `util::string` to be returned.

  The substring is the portion of the object that starts at character position start_position and spans len characters (or until the end of the string, whichever comes first).

- `length()`: Returns the amount of characters of your string excluding `\0`. Might be smaller than the actual reserved memory.

- `size()`: (synonyme to `length()`) Returns the amount of characters of your string excluding `\0`. Might be smaller than the actual reserved memory.

- `capacity()`: Returns the size of the storage space currently allocated for the string, expressed in terms of bytes.

- `rawSize(const char* rawChar)` : Get the amount of characters of a raw char* string excluding the terminating `\0`.

- `c_str()`: Allows raw access to the internal C-string respectively the `char*` pointer.

  Returns a pointer to an array that contains a null-terminated sequence of characters (i.e., a C-string) representing the current value of the string object.

- `intialize_string(size_t length = 0)` Ensure a string is initialized before using it. It initialize an empty string with buffer size of the given length.

- `deepCopy(const char* rawChar, size_t startPosition = 0)`: Deep copy of primitive C-string into the string internal buffer.

  This function realizes a design and implementation of a concept to extend the internal memory if util::string has to store more than its current allocated buffer size.

  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

- `compare(const char* s1, const char* s2)`: 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.

1.5 Non-member Functions

Some non-member utility functions are implemented that help for better functionality and output. These functions are:

- `deepCopy(char* rawCharTarget, const char* rawCharSource, size_t destStartPosition, size_t srcEndPosition)` : Deep copy of primitive C-string into another primitive C-string. This function realizes a design and implementation of a concept to low-level copy and fill a primitive C-string with another primitive C-string starting from a given start position and with a desired number of character from the sourrce string.

  Note that the function copy the passed char array starting from the startPosition (i.e. it can write starting from any position in the destination string buffer) startPosition default is 0

- `printHeader(const char* text), printSubHeader(const char* text), printTestCase(const char* text)` : to print a nicely formatted and colored text header, sub header, title header, respectively to the terminal

1.6 References

- Standard Strings library : https://en.cppreference.com/w/cpp/string
- C++ ISO Standard https://isocpp.org/std/the-standard
- C++ documentation - DevDocs : https://devdocs.io/cpp/
- LLVM Coding Standards: https://llvm.org/docs/CodingStandards.html
- sanke_case convention : https://en.wikipedia.org/wiki/Snake_case
- Markdown Basic Syntax : https://www.markdownguide.org/basic-syntax
- Doxygen : https://www.doxygen.nl/index.html

Chapter 2

Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 util Namespace Reference

Classes

- class string

  Implementation of of own string class.

Functions

- 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)
- void deepCopy (char *rawCharTarget, const char *rawCharSource, size_t destStartPosition=-1, size_t srcEndPosition=-1)

  Deep copy of primitive C-string into another primitive C-string.
- void printHeader (const char *text)

  to print a nicely formatted and colored text header to the terminal
- void printSubHeader (const char *text)

  to print a nicely formatted and colored text sub header to the terminal
- void printTestCase (const char *text)

  to print a nicely formatted and colored text title header to the terminal

5.1.1 Function Documentation

5.1.1.1 deepCopy()

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

Deep copy of primitive C-string into another primitive C-string.

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)

destStartPosition default is to first location of the destRawChar srcEndPosition default is to last character (before the \0) of the srcRawChar

This function realizes a design and implementation of a concept to low-level copy and fill a primitive C-string with another primitive C-string starting from a given start position and with a desired number of character from the sourrce string. note that the function copy the passed char array starting from the startPosition (i.e. it can write starting from any position in the destination string buffer) startPosition default is 0

Parameters

rawCharTarget	pointer to the destination primitive C-string to copy to
rawCharSource	pointer to the source primitive C-string to copy from
destStartPosition	start position(in the target buffer) to start copying the source string to [default: 0].
srcEndPosition	end position (in the source string) to stop at [default: to the end of the source array].

Definition at line 400 of file utilstring.cpp.

5.1.1.2 operator"!=() [1/2]

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

Definition at line 377 of file utilstring.cpp.

5.1.1.3 operator"!=() [2/2]

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

Compares the contents of a string with another string, std::string, or a null - terminated array of char for non-equality. For the cases util::string is on the RHS.

Definition at line 372 of file utilstring.cpp.

5.1.1.4 operator<<()

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

Insert string into stream

Definition at line 356 of file utilstring.cpp.

5.1.1.5 operator==() [1/2]

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

Definition at line 366 of file utilstring.cpp.

5.1.1.6 operator==() [2/2]

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

Compares the contents of a string with another string, std::string, or a null - terminated array of char for equality. For the cases util::string is on the RHS.

Definition at line 361 of file utilstring.cpp.

5.1.1.7 printHeader()

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

to print a nicely formatted and colored text header to the terminal

utility functions for printing nice text output

ANSI Escape Sequences are used to color the console text, it works for windows and Linux. For Windows, you need to run the program in the new terminal as the old one does not support these codes. see: https://gist.github.com/fnky/458719343aabd01cfb17a3a4f7296797

Definition at line 433 of file utilstring.cpp.

5.1.1.8 printSubHeader()

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

to print a nicely formatted and colored text sub header to the terminal

Definition at line 450 of file utilstring.cpp.

5.1.1.9 printTestCase()

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

to print a nicely formatted and colored text title header to the terminal

Definition at line 456 of file utilstring.cpp.

Chapter 6

Class Documentation

6.1 util::string Class Reference

Implementation of of own string class.

`#include <utilstring.h>`

Public Member Functions

- string ()
- string (size_t intialSize)
- string (const char ∗charArray)
- string (const std::string &std_string)
- string (const string &)
- ∼string (void)
- void intialize_string (size_t length=0)

  Ensure a string is initialized before using it.
- void deepCopy (const char ∗rawChar, size_t startPosition=0)

  Deep copy of primitive C-string into the string internal buffer.
- string substr (size_t start_position, size_t length)

  Returns a newly constructed util::string object with a portion of the string.
- char ∗ c_str () const

  Allows raw access to the internal C-string (through its char∗ pointer)
- void clear ()
- size_t length () const
- size_t size () const
- size_t capacity () const
- string operator+ (const string &rhsString)
- string operator+ (const std::string &rhsString)
- string operator+ (const char ∗strInstance)
- string & operator+= (const string &rhsString)
- string & operator+= (const std::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)
- 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 char *lhsCharArray, const util::string &rhsString)
- bool operator!= (const char *lhsCharArray, const util::string &rhsString)

Related Functions

(Note that these are not member functions.)

- bool operator== (const std::string &lhsString, const util::string &rhsString)
- bool operator!= (const std::string &lhsString, const util::string &rhsString)
- void deepCopy (char *rawCharTarget, const char *rawCharSource, size_t destStartPosition=-1, size_t src↩ EndPosition=-1)

  Deep copy of primitive C-string into another primitive C-string.

6.1.1 Detailed Description

Implementation of of own string class.

@description

This class presents own string class implementation. This class behavior will be similar and compatible to the std::string.

Definition at line 42 of file utilstring.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 string() [1/5]

```
util::string::string ( )
```

default constructor with empty initialization

Default Constructor

Definition at line 28 of file utilstring.cpp.

6.1.2.2 string() [2/5]

```
util::string::string (
            size_t intialSize )
```

Constructor with a given initial size

Parameters

intialSize	number of bytes (characters) to intialize the string with.

Constructor with a given initial size

Definition at line 31 of file utilstring.cpp.

6.1.2.3 string() [3/5]

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

Constructor with parameter const char*

Parameters

charArray	primitive C-string to intialize the string with.

Constructor with char*

Definition at line 35 of file utilstring.cpp.

6.1.2.4 string() [4/5]

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

Constructor with parameter std::string

Parameters

std_string	c++ std::string to intialize the string with.

Constructor with parameter std::string

Definition at line 44 of file utilstring.cpp.

6.1.2.5 string() [5/5]

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

Copy constructor: Creates a deep copy of a passed string

Parameters

std_string	c++ util::string to intialize the string with.

Copy constructor: Creates a deep copy of a passed string

Definition at line 53 of file utilstring.cpp.

6.1.2.6 ~string()

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

Destructor to do the final cleanup and memory deallocation

Definition at line 63 of file utilstring.cpp.

6.1.3 Member Function Documentation

6.1.3.1 c_str()

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

Allows raw access to the internal C-string (through its char* pointer)

Returns a pointer to an array that contains a null-terminated sequence of characters (i.e., a C-string) representing the current value of the string object.

Returns

Returns a pointer to an array that contains a null-terminated sequence of characters.

Definition at line 143 of file utilstring.cpp.

6.1.3.2 capacity()

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

Returns the size of the storage space currently allocated for the string, expressed in terms of bytes.

Definition at line 86 of file utilstring.cpp.

6.1.3.3 clear()

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

Clears your string object Erases the contents of the string, which becomes an empty string (with a length of 0 characters).

Definition at line 96 of file utilstring.cpp.

6.1.3.4 compare()

```
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

s1	pointer to the primitive C string to be compared.
s2	pointer to the primitive C string to be compared with.

Returns

an integral value indicating the relationship between the strings

Return values

$<$	0 The first character that does not match has a lower value in s1 than in s2
0	The contents of both strings are equal
$>$	0 The first character that does not match has a greater value in s1 than that in in s2

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

str1	primitive C string to be compared.
str2	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 184 of file utilstring.cpp.

6.1.3.5 deepCopy()

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

Deep copy of primitive C-string into the string internal buffer.

This function realizes a design and implementation of a concept to extend the internal memory if util::string has to store more than its current allocated buffer size. 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

Parameters

rawChar	pointer to the primitive C-string to copy to the internal buffer.
startPosition	start position (in the target buffer) to start copying the source string to [default: 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 120 of file utilstring.cpp.

6.1.3.6 intialize_string()

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

Ensure a string is initialized before using it.

initialize an empty string with buffer size of the given length.

Parameters

length	number of bytes (characters) to allocate in memory for the internal buffer.

Definition at line 69 of file utilstring.cpp.

6.1.3.7 length()

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

Returns the amount of characters of your string excluding \0. Might be smaller than the actual reserved memory.

Definition at line 79 of file utilstring.cpp.

6.1.3.8 operator"!=() [1/3]

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

Definition at line 255 of file utilstring.cpp.

6.1.3.9 operator"!=() [2/3]

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

Definition at line 248 of file utilstring.cpp.

6.1.3.10 operator"!=() [3/3]

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

Compares the contents of a string with another string, std::string, or a null - terminated array of char for non-equality.

Definition at line 242 of file utilstring.cpp.

6.1.3.11 operator+() [1/3]

```
string util::string::operator+ (
            const char * strInstance )
```

Definition at line 342 of file utilstring.cpp.

6.1.3.12 operator+() [2/3]

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

Definition at line 332 of file utilstring.cpp.

6.1.3.13 operator+() [3/3]

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

Operator + such that string, std::string and (const) char* can be added

Definition at line 322 of file utilstring.cpp.

6.1.3.14 operator+=() [1/3]

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

concatenating util::string and const char*

Definition at line 306 of file utilstring.cpp.

6.1.3.15 operator+=() [2/3]

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

Definition at line 287 of file utilstring.cpp.

6.1.3.16 operator+=() [3/3]

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

Definition at line 269 of file utilstring.cpp.

6.1.3.17 operator=() [1/3]

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

Definition at line 210 of file utilstring.cpp.

6.1.3.18 operator=() [2/3]

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

Definition at line 215 of file utilstring.cpp.

6.1.3.19 operator=() [3/3]

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

Assigns a new value to the string, replacing its current contents.

Definition at line 205 of file utilstring.cpp.

6.1.3.20 operator==() [1/3]

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

Definition at line 234 of file utilstring.cpp.

6.1.3.21 operator==() [2/3]

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

Definition at line 227 of file utilstring.cpp.

6.1.3.22 operator==() [3/3]

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

Compares the contents of a string with another string, std::string, or a null - terminated array of char for equality.

Definition at line 221 of file utilstring.cpp.

6.1.3.23 operator[]()

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

Returns a reference to the character at position pos in the string.

Definition at line 263 of file utilstring.cpp.

6.1.3.24 rawSize()

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

Get the amount of characters of a raw char* string excluding the terminating \0.

Definition at line 103 of file utilstring.cpp.

6.1.3.25 size()

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

Get the length of the string synonyme to length()

Definition at line 78 of file utilstring.cpp.

6.1.3.26 substr()

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

Returns a newly constructed util::string object with a portion of the string.

The substring is the portion of the object that starts at character position start_position and spans len characters (or until the end of the string, whichever comes first).

Parameters

in	*start_position*	start position in the source string.
in	*length*	specifies the amount of characters of the new util::string to be returned.

Returns

Returns a substring object of type util::string which starts at pos

Definition at line 152 of file utilstring.cpp.

6.1.4 Friends And Related Function Documentation

6.1.4.1 deepCopy()

```
void deepCopy (
            char * rawCharTarget,
            const char * rawCharSource,
            size_t destStartPosition = -1,
            size_t srcEndPosition = -1 )  [related]
```

Deep copy of primitive C-string into another primitive C-string.

This function realizes a design and implementation of a concept to low-level copy and fill a primitive C-string with another primitive C-string starting from a given start position and with a desired number of character from the sourrce string. note that the function copy the passed char array starting from the startPosition (i.e. it can write starting from any position in the destination string buffer) startPosition default is 0

Parameters

rawCharTarget	pointer to the destination primitive C-string to copy to
rawCharSource	pointer to the source primitive C-string to copy from
destStartPosition	start position(in the target buffer) to start copying the source string to [default: 0].
srcEndPosition	end position (in the source string) to stop at [default: to the end of the source array].

Definition at line 400 of file utilstring.cpp.

6.1.4.2 operator"!= [1/2]

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

Definition at line 377 of file utilstring.cpp.

6.1.4.3 operator"!= [2/2]

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

Compares the contents of a string with another string, std::string, or a null - terminated array of char for non-equality. For the cases util::string is on the RHS.

Definition at line 372 of file utilstring.cpp.

6.1.4.4 operator<<

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

Insert string into stream

Definition at line 356 of file utilstring.cpp.

6.1.4.5 operator== [1/2]

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

Definition at line 366 of file utilstring.cpp.

6.1.4.6 operator== [2/2]

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

Compares the contents of a string with another string, std::string, or a null - terminated array of char for equality. For the cases util::string is on the RHS.

Definition at line 361 of file utilstring.cpp.

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

- utilstring.h
- utilstring.cpp

Chapter 7

File Documentation

7.1 main.cpp File Reference

: test of own implementation of string class

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

Functions

- int main ()

7.1.1 Detailed Description

: test of own implementation of string class

==========

Author

: Nour Ahmed @email : nahmed@stud.hs-bremen.de, nour @repo : https://github.com/nouremara/cpp_mystring @repo : @createdOn : 23.11.2022

Version

: 1.0.0 @description :

Defines the entry point for the NourUtilString application In this application the class util::string is used and tested. Each method and operator is tested with all possible uasges (e.g., concatenating different strings etc.) Note: For this task no error handling is required.Example: Accessing

7.1.2 an invalid index by using operator [ ]

Definition in file main.cpp.

7.1.3 Function Documentation

7.1.3.1 main()

```
int main ( )
```

Definition at line 28 of file main.cpp.

7.2 main.cpp

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 1 - Own string Class
00004
00024 #include "utilstring.h"
00025
00026 #include <iostream>
00027
00028 int main() {
00029     char charArray[] = "text in a const char array";
00030     std::string stdString("another text in a std::string");
00031
00032     // instantiate objects
00033     util::string string1;
00034     util::string string2("initializing with const char array");
00035     util::string string3(charArray);
00036     util::string string4(stdString);
00037     util::string string5(string4);
00038
00039     util::printHeader("NourUtilString Application");
00040     std::cout « "\033[1;30;106m- Nour Ahmed
            -" « std::endl;
00041     std::cout « "- Matrikal-Nr.: 5200991                                                    -" «
       std::endl;
00042     std::cout « "- Assignment 1 - Own string Class                                          -" «
       std::endl;
00043     std::cout «
       "-------------------------------------------------------------------------------\033[0m\n\n";
00044
00045     // Test Object Instantiation ------------------------------------------
00046     util::printSubHeader("Variable used for testing and their values");
00047     std::cout « "Variable used for testing and their values" « std::endl;
00048
00049     util::printTestCase("charArray");
00050     std::cout « "charArray (size: " « util::string::rawSize(charArray) « ")  : " « charArray «
       std::endl;
00051
00052     util::printTestCase("stdString");
00053     std::cout « "charArray (size: " « stdString.length() « ")  : " « stdString « std::endl;
00054
00055     std::cout « "-------------------------------------------------------------------------------\n\n";
00056
00057     util::printSubHeader("Test object constructors and initialization");
00058
00059     util::printTestCase("default constructor");
00060     std::cout « "\tstring1 (size: " « string1.size() « ")  : " « string1 « std::endl;
00061
00062     util::printTestCase("constructor with const char*");
00063     std::cout « "string2 (size: " « string2.size() « ") : " « string2 « std::endl;
00064
00065     util::printTestCase("constructor with std::string");
00066           std::cout « "string3 (size: " « string3.size() « ") : " « string3 « std::endl;
00067
00068     util::printTestCase("constructor with char array");
00069     std::cout « "string4 (size: " « string4.size() « ") : " « string4 « std::endl;
00070
00071     util::printTestCase("constructor with util::string");
00072     std::cout « "string5 (size: " « string5.size() « ") : " « string4 « std::endl;
00073     std::cout « "-------------------------------------------------------------------------------\n\n";
00074
00075
```

```
00076     // Test member methods -------------------------------------------------------
00077     util::printSubHeader("Test Member Methods");
00078 
00079 
00080     util::printTestCase("length()");
00081     std::cout « "string2 (size: " « string2.size() « ", capacity: " « string2.capacity() « ") : " «
      string2 « std::endl;
00082 
00083     util::printTestCase("size()");
00084     std::cout « "string2 (size: " « string2.size() « ", capacity: " « string2.capacity() « ") : " «
      string2 « std::endl;
00085 
00086     util::printTestCase("capacity()");
00087     std::cout « "string2 (size: " « string2.size() « ", capacity: " « string2.capacity() « ") : " «
      string2 « std::endl;
00088 
00089     util::string temp = string2.substr(3, 5);
00090     util::printTestCase("substr()");
00091     std::cout « "string2.substr(3,5) \t -> " « temp « std::endl;
00092 
00093     util::printTestCase("c_str()");
00094     std::cout « "string2.c_str() \t -> " « string2.c_str() « std::endl;
00095 
00096     string2.clear();
00097     util::printTestCase("clear()");
00098     std::cout « "string2.clear() -> string2 (size: " « string2.size() « ", capacity: " «
      string2.capacity() « ") : content: " « string2 « std::endl;
00099     std::cout « "-------------------------------------------------------------------------------\n\n";
00100 
00101     // Test operators ------------------------------------------------------
00102     util::printSubHeader("Test operators");
00103 
00104     util::printTestCase("operator «");
00105     std::cout « "std::cout « util::string « int « std::string « char *:\n"
00106         « "    string3 (size: " « string3.size() « ", capacity: " « string3.capacity() « ") : content:
      " « string3
00107         « stdString « ", "
00108         « charArray
00109         « std::endl;
00110     std::cout « "-------------------------------------------------------------------------------\n\n";
00111 
00112     util::printTestCase("operator +");
00113     std::cout « "\n\tutil::string + util::string \t: string2 + string3 -> " « string2 + string3 «
      std::endl;
00114 
00115     string5 = string5 + " see how + operator with char * works";
00116     std::cout « "\tutil::string + const char* \t: string5 = string5 + const char* -> (size: "  «
      string5.size() « ") : " « string5;
00117     std::cout «
      "\n-------------------------------------------------------------------------------\n\n";
00118 
00119     string4 += string3;
00120     util::printTestCase("operator +=");
00121     std::cout « "\n\twith util::string\t: string4 += string3 -> (size: " « string4.size() « ") : " «
      string4 « std::endl;
00122 
00123     string4 += " here += operator is used to add more text in char *";
00124     std::cout « "\twith const char* \t: string4 += const char* -> (size: " « string4.size() « ") : " «
      string4 « std::endl;
00125     std::cout « "-------------------------------------------------------------------------------\n\n";
00126 
00127 
00128     string1 = string4;
00129     string2 = "more text for testing";
00130     string3 = std::string("text for std::string assignment");
00131 
00132     util::printTestCase("operator =");
00133     std::cout « "\n\tutil::string = util::string\t string1 = string4 -> string1 (size: " «
      string1.size() « "): " « string1 « std::endl;
00134     std::cout « "\tutil::string = const char* \t string2 = \"...\" -> string2 (size: " «
      string2.size() « "): " « string2 « std::endl;
00135     std::cout « "\tutil::string = std::string \t string3 = std::string(\"...\") -> string3 (size: " «
      string3.size() « "): " « string3 « std::endl;
00136     std::cout « "-------------------------------------------------------------------------------\n\n";
00137 
00138     string1 = string2;
00139     util::printTestCase("operator ==");
00140     std::cout « "\n\tutil::string == util::string \t string1   == string2   -> " « ((string1 ==
      string2) ? "true" : "false") « std::endl;
00141     std::cout « "\tutil::string == std::string  \t string1   == stdString -> " « ((string1 ==
      stdString) ? "true" : "false") « std::endl;
00142     std::cout « "\tstd::string  == util::string \t stdString == string1   -> " « ((stdString ==
      string1) ? "true" : "false") « std::endl;
00143     std::cout « "\tutil::string == const char*  \t string1   == charArray -> " « ((string1 ==
      charArray) ? "true" : "false") « std::endl;
00144     std::cout « "\tconst char*  == util::string \t charArray == string1   -> " « ((charArray ==
      string1) ? "true" : "false") « std::endl;
```

```
00145     std::cout « "--------------------------------------------------------------------------------\n\n";
00146
00147     util::printTestCase("operator !=");
00148     std::cout « "\n\tutil::string != util::string \t string1   != string2   -> " « ((string1 !=
      string2) ? "true" : "false") « std::endl;
00149     std::cout « "\tutil::string != std::string  \t string1   != stdString -> " « ((string1 !=
      stdString) ? "true" : "false") « std::endl;
00150     std::cout « "\tstd::string  != util::string \t stdString != string1   -> " « ((stdString !=
      string1) ? "true" : "false") « std::endl;
00151     std::cout « "\tutil::string != const char*  \t string1   != charArray -> " « ((string1 !=
      charArray) ? "true" : "false") « std::endl;
00152     std::cout « "\tconst char*  != util::string \t charArray != string1   -> " « ((charArray !=
      string1) ? "true" : "false") « std::endl;
00153     std::cout « "--------------------------------------------------------------------------------\n\n";
00154
00155     util::printTestCase("operator []");
00156     std::cout « "\n\tstring1: " « string1 « "-> string1[0]: " « string1[0] « std::endl;
00157     std::cout « "\tstring2: " « string2 « "-> string2[3]: " « string2[3] « std::endl;
00158     //std::cout « "\tstring3: " « string3 « "-> string3[50]: " « string3[50] « std::endl;
00159
00160     string2[3] = 'A';
00161     std::cout « "\tstring2[3] = \'A\'" « "-> string2[3]: " « string2[3] « std::endl;
00162     std::cout « "\tstring2: " « string2 « "-> string2[3]: " « string2[3] « std::endl;
00163     std::cout « "--------------------------------------------------------------------------------\n\n";
00164
00165     // Test utility functions
00166     util::printSubHeader("Test utility functions");
00167
00168     char s1[100] = "programming ", s2[] = "is awesome";
00169     std::cout « "\ts1 (size: " « util::string::rawSize(s1) « ", capacity: 100) : content: " « s1 «
      std::endl;
00170     std::cout « "\ts2 (size: " « util::string::rawSize(s2) « ", capacity: "«
      util::string::rawSize(s2)+1 «") : content: " « s2 « std::endl;
00171
00172     util::printTestCase("util::deepCopy()");
00173     util::deepCopy(s1, s2);
00174     std::cout « "\n\tdeepCopy(s1, s2) -> s1 (size: " « util::string::rawSize(s1) « ", capacity: 100) :
      content: " « s1 « std::endl;
00175
00176     util::printTestCase("util::rawSize()");
00177     std::cout « "\n\tutil::string::rawSize(s1) -> " « util::string::rawSize(s1) « std::endl;
00178
00179     util::printTestCase("util::string::compare()");
00180     std::cout « "\n\tutil::string::compare(s1,s2) -> " « util::string::compare(s1, s2) « std::endl;
00181     std::cout « "\tutil::string::compare(s1,s1) -> " « util::string::compare(s1, s1) « std::endl;
00182     std::cout « "--------------------------------------------------------------------------------\n\n";
00183
00184     util::printSubHeader("Test utility functions");
00185     util::printTestCase("util::printHeader()");      std::cout « std::endl;
00186     util::printTestCase("util::printSubHeader()");  std::cout « std::endl;
00187     util::printTestCase("util::printTestCase()");   std::cout « std::endl;
00188     std::cout « "\tThese functions are used to print the above colored headers :)" « std::endl;
00189     std::cout « "--------------------------------------------------------------------------------\n\n";
00190
00191     return 0;
00192 }
```

7.3 out/build/x64-Debug/CMakeFiles/3.21.21080301-MSVC_2/CompilerIdC/CMakeCCompilerId.c File Reference

Macros

- #define __has_include(x) 0
- #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

7.3.1 Macro Definition Documentation

7.3.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 17 of file CMakeCCompilerId.c.

7.3.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 668 of file CMakeCCompilerId.c.

7.3.1.3 C_DIALECT

```
#define C_DIALECT
```

Definition at line 757 of file CMakeCCompilerId.c.

7.3.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 412 of file CMakeCCompilerId.c.

7.3.1.5 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 672 of file CMakeCCompilerId.c.

7.3.1.6 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 683 of file CMakeCCompilerId.c.

7.3.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 540 of file CMakeCCompilerId.c.

7.3.1.8 STRINGIFY

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

Definition at line 433 of file CMakeCCompilerId.c.

7.3.1.9 STRINGIFY_HELPER

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

Definition at line 432 of file CMakeCCompilerId.c.

7.3.2 Function Documentation

7.3.2.1 main()

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

Definition at line 781 of file CMakeCCompilerId.c.

7.3.3 Variable Documentation

7.3.3.1 info_arch

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

Definition at line 749 of file CMakeCCompilerId.c.

7.3.3.2 info_compiler

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

Definition at line 419 of file CMakeCCompilerId.c.

7.3.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

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

Definition at line 770 of file CMakeCCompilerId.c.

7.3.3.4 info_platform

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

Definition at line 748 of file CMakeCCompilerId.c.

7.4 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 #if !defined(__has_include)
00015 /* If the compiler does not have __has_include, pretend the answer is
00016    always no.  */
00017 #  define __has_include(x) 0
00018 #endif
00019
00020
00021 /* Version number components: V=Version, R=Revision, P=Patch
00022    Version date components:   YYYY=Year, MM=Month,   DD=Day  */
00023
00024 #if defined(__INTEL_COMPILER) || defined(__ICC)
00025 # define COMPILER_ID "Intel"
00026 # if defined(_MSC_VER)
00027 #  define SIMULATE_ID "MSVC"
00028 # endif
00029 # if defined(__GNUC__)
00030 #  define SIMULATE_ID "GNU"
00031 # endif
00032   /* __INTEL_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,
00033      except that a few beta releases use the old format with V=2021.  */
00034 # if __INTEL_COMPILER < 2021 || __INTEL_COMPILER == 202110 || __INTEL_COMPILER == 202111
00035 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00036 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00037 #  if defined(__INTEL_COMPILER_UPDATE)
00038 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00039 #  else
00040 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER   % 10)
00041 #  endif
00042 # else
00043 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER)
00044 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER_UPDATE)
00045    /* The third version component from --version is an update index,
00046       but no macro is provided for it.  */
00047 #  define COMPILER_VERSION_PATCH DEC(0)
00048 # endif
00049 # if defined(__INTEL_COMPILER_BUILD_DATE)
00050    /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00051 #  define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00052 # endif
00053 # if defined(_MSC_VER)
00054    /* _MSC_VER = VVRR */
00055 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00056 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00057 # endif
00058 # if defined(__GNUC__)
00059 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00060 # elif defined(__GNUG__)
00061 #  define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00062 # endif
00063 # if defined(__GNUC_MINOR__)
00064 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00065 # endif
00066 # if defined(__GNUC_PATCHLEVEL__)
00067 #  define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00068 # endif
00069
```

```
00070 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00071 # define COMPILER_ID "IntelLLVM"
00072 #if defined(_MSC_VER)
00073 # define SIMULATE_ID "MSVC"
00074 #endif
00075 #if defined(__GNUC__)
00076 # define SIMULATE_ID "GNU"
00077 #endif
00078 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00079  * later.  Look for 6 digit vs. 8 digit version number to decide encoding.
00080  * VVVV is no smaller than the current year when a version is released.
00081  */
00082 #if __INTEL_LLVM_COMPILER < 1000000L
00083 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00084 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00085 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER    % 10)
00086 #else
00087 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00088 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
00089 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER     % 100)
00090 #endif
00091 #if defined(_MSC_VER)
00092   /* _MSC_VER = VVRR */
00093 # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00094 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00095 #endif
00096 #if defined(__GNUC__)
00097 # define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00098 #elif defined(__GNUG__)
00099 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00100 #endif
00101 #if defined(__GNUC_MINOR__)
00102 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00103 #endif
00104 #if defined(__GNUC_PATCHLEVEL__)
00105 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00106 #endif
00107
00108 #elif defined(__PATHCC__)
00109 # define COMPILER_ID "PathScale"
00110 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00111 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00112 # if defined(__PATHCC_PATCHLEVEL__)
00113 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00114 # endif
00115
00116 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00117 # define COMPILER_ID "Embarcadero"
00118 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__»24 & 0x00FF)
00119 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__»16 & 0x00FF)
00120 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__     & 0xFFFF)
00121
00122 #elif defined(__BORLANDC__)
00123 # define COMPILER_ID "Borland"
00124   /* __BORLANDC__ = 0xVRR */
00125 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__»8)
00126 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00127
00128 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00129 # define COMPILER_ID "Watcom"
00130    /* __WATCOMC__ = VVRR */
00131 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00132 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00133 # if (__WATCOMC__ % 10) > 0
00134 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00135 # endif
00136
00137 #elif defined(__WATCOMC__)
00138 # define COMPILER_ID "OpenWatcom"
00139    /* __WATCOMC__ = VVRP + 1100 */
00140 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00141 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00142 # if (__WATCOMC__ % 10) > 0
00143 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00144 # endif
00145
00146 #elif defined(__SUNPRO_C)
00147 # define COMPILER_ID "SunPro"
00148 # if __SUNPRO_C >= 0x5100
00149    /* __SUNPRO_C = 0xVRRP */
00150 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C»12)
00151 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_C»4 & 0xFF)
00152 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00153 # else
00154    /* __SUNPRO_CC = 0xVRP */
00155 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_C»8)
00156 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_C»4 & 0xF)
```

```
00157 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_C    & 0xF)
00158 # endif
00159
00160 #elif defined(__HP_cc)
00161 # define COMPILER_ID "HP"
00162   /* __HP_cc = VVRRPP */
00163 # define COMPILER_VERSION_MAJOR DEC(__HP_cc/10000)
00164 # define COMPILER_VERSION_MINOR DEC(__HP_cc/100 % 100)
00165 # define COMPILER_VERSION_PATCH DEC(__HP_cc     % 100)
00166
00167 #elif defined(__DECC)
00168 # define COMPILER_ID "Compaq"
00169   /* __DECC_VER = VVRRTPPPP */
00170 # define COMPILER_VERSION_MAJOR DEC(__DECC_VER/10000000)
00171 # define COMPILER_VERSION_MINOR DEC(__DECC_VER/100000  % 100)
00172 # define COMPILER_VERSION_PATCH DEC(__DECC_VER         % 10000)
00173
00174 #elif defined(__IBMC__) && defined(__COMPILER_VER__)
00175 # define COMPILER_ID "zOS"
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(__ibmxl__) && defined(__clang__)
00182 # define COMPILER_ID "XLClang"
00183 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00184 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00185 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00186 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00187
00188
00189 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ >= 800
00190 # define COMPILER_ID "XL"
00191   /* __IBMC__ = VRP */
00192 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00193 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00194 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00195
00196 #elif defined(__IBMC__) && !defined(__COMPILER_VER__) && __IBMC__ < 800
00197 # define COMPILER_ID "VisualAge"
00198   /* __IBMC__ = VRP */
00199 # define COMPILER_VERSION_MAJOR DEC(__IBMC__/100)
00200 # define COMPILER_VERSION_MINOR DEC(__IBMC__/10 % 10)
00201 # define COMPILER_VERSION_PATCH DEC(__IBMC__    % 10)
00202
00203 #elif defined(__NVCOMPILER)
00204 # define COMPILER_ID "NVHPC"
00205 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00206 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00207 # if defined(__NVCOMPILER_PATCHLEVEL__)
00208 #  define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00209 # endif
00210
00211 #elif defined(__PGI)
00212 # define COMPILER_ID "PGI"
00213 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00214 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00215 # if defined(__PGIC_PATCHLEVEL__)
00216 #  define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00217 # endif
00218
00219 #elif defined(_CRAYC)
00220 # define COMPILER_ID "Cray"
00221 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00222 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00223
00224 #elif defined(__TI_COMPILER_VERSION__)
00225 # define COMPILER_ID "TI"
00226   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00227 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00228 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00229 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00230
00231 #elif defined(__CLANG_FUJITSU)
00232 # define COMPILER_ID "FujitsuClang"
00233 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00234 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00235 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00236 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00237
00238
00239 #elif defined(__FUJITSU)
00240 # define COMPILER_ID "Fujitsu"
00241 # if defined(__FCC_version__)
00242 #   define COMPILER_VERSION __FCC_version__
00243 # elif defined(__FCC_major__)
```

```
00244 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00245 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00246 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00247 # endif
00248 # if defined(__fcc_version)
00249 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00250 # elif defined(__FCC_VERSION)
00251 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00252 # endif
00253
00254
00255 #elif defined(__ghs__)
00256 # define COMPILER_ID "GHS"
00257 /* __GHS_VERSION_NUMBER = VVVVRP */
00258 # ifdef __GHS_VERSION_NUMBER
00259 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00260 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00261 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00262 # endif
00263
00264 #elif defined(__TINYC__)
00265 # define COMPILER_ID "TinyCC"
00266
00267 #elif defined(__BCC__)
00268 # define COMPILER_ID "Bruce"
00269
00270 #elif defined(__SCO_VERSION__)
00271 # define COMPILER_ID "SCO"
00272
00273 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00274 # define COMPILER_ID "ARMCC"
00275 #if __ARMCC_VERSION >= 1000000
00276   /* __ARMCC_VERSION = VRRPPPP */
00277   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00278   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00279   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00280 #else
00281   /* __ARMCC_VERSION = VRPPPP */
00282   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00283   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00284   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00285 #endif
00286
00287
00288 #elif defined(__clang__) && defined(__apple_build_version__)
00289 # define COMPILER_ID "AppleClang"
00290 # if defined(_MSC_VER)
00291 #  define SIMULATE_ID "MSVC"
00292 # endif
00293 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00294 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00295 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00296 # if defined(_MSC_VER)
00297   /* _MSC_VER = VVRR */
00298 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00299 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00300 # endif
00301 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00302
00303 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00304 # define COMPILER_ID "ARMClang"
00305   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00306   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00307   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION     % 10000)
00308 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00309
00310 #elif defined(__clang__) && __has_include(<hip/hip_version.h>)
00311 # define COMPILER_ID "ROCMClang"
00312 # if defined(_MSC_VER)
00313 #  define SIMULATE_ID "MSVC"
00314 # elif defined(__clang__)
00315 #  define SIMULATE_ID "Clang"
00316 # elif defined(__GNUC__)
00317 #  define SIMULATE_ID "GNU"
00318 # endif
00319 # if defined(__clang__) && __has_include(<hip/hip_version.h>)
00320 #  include <hip/hip_version.h>
00321 #  define COMPILER_VERSION_MAJOR DEC(HIP_VERSION_MAJOR)
00322 #  define COMPILER_VERSION_MINOR DEC(HIP_VERSION_MINOR)
00323 #  define COMPILER_VERSION_PATCH DEC(HIP_VERSION_PATCH)
00324 # endif
00325
00326 #elif defined(__clang__)
00327 # define COMPILER_ID "Clang"
00328 # if defined(_MSC_VER)
00329 #  define SIMULATE_ID "MSVC"
00330 # endif
```

```
00331 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00332 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00333 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00334 # if defined(_MSC_VER)
00335    /* _MSC_VER = VVRR */
00336 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00337 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00338 # endif
00339
00340 #elif defined(__GNUC__)
00341 # define COMPILER_ID "GNU"
00342 # define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00343 # if defined(__GNUC_MINOR__)
00344 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00345 # endif
00346 # if defined(__GNUC_PATCHLEVEL__)
00347 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00348 # endif
00349
00350 #elif defined(_MSC_VER)
00351 # define COMPILER_ID "MSVC"
00352   /* _MSC_VER = VVRR */
00353 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00354 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00355 # if defined(_MSC_FULL_VER)
00356 #  if _MSC_VER >= 1400
00357     /* _MSC_FULL_VER = VVRRPPPPP */
00358 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00359 #  else
00360     /* _MSC_FULL_VER = VVRRPPPP */
00361 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00362 #  endif
00363 # endif
00364 # if defined(_MSC_BUILD)
00365 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00366 # endif
00367
00368 #elif defined(__VISUALDSPVERSION__) || defined(__ADSPBLACKFIN__) || defined(__ADSPTS__) ||
       defined(__ADSP21000__)
00369 # define COMPILER_ID "ADSP"
00370 #if defined(__VISUALDSPVERSION__)
00371   /* __VISUALDSPVERSION__ = 0xVVRRPP00 */
00372 # define COMPILER_VERSION_MAJOR HEX(__VISUALDSPVERSION__»24)
00373 # define COMPILER_VERSION_MINOR HEX(__VISUALDSPVERSION__»16 & 0xFF)
00374 # define COMPILER_VERSION_PATCH HEX(__VISUALDSPVERSION__»8  & 0xFF)
00375 #endif
00376
00377 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00378 # define COMPILER_ID "IAR"
00379 # if defined(__VER__) && defined(__ICCARM__)
00380 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00381 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00382 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00383 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00384 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
       defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
       defined(__ICC8051__) || defined(__ICCSTM8__))
00385 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00386 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00387 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00388 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00389 # endif
00390
00391 #elif defined(__SDCC_VERSION_MAJOR) || defined(SDCC)
00392 # define COMPILER_ID "SDCC"
00393 # if defined(__SDCC_VERSION_MAJOR)
00394 #  define COMPILER_VERSION_MAJOR DEC(__SDCC_VERSION_MAJOR)
00395 #  define COMPILER_VERSION_MINOR DEC(__SDCC_VERSION_MINOR)
00396 #  define COMPILER_VERSION_PATCH DEC(__SDCC_VERSION_PATCH)
00397 # else
00398   /* SDCC = VRP */
00399 #  define COMPILER_VERSION_MAJOR DEC(SDCC/100)
00400 #  define COMPILER_VERSION_MINOR DEC(SDCC/10 % 10)
00401 #  define COMPILER_VERSION_PATCH DEC(SDCC    % 10)
00402 # endif
00403
00404
00405 /* These compilers are either not known or too old to define an
00406   identification macro.  Try to identify the platform and guess that
00407   it is the native compiler.  */
00408 #elif defined(__hpux) || defined(__hpua)
00409 # define COMPILER_ID "HP"
00410
00411 #else /* unknown compiler */
00412 # define COMPILER_ID ""
00413 #endif
00414
```

```
00415 /* Construct the string literal in pieces to prevent the source from
00416    getting matched.  Store it in a pointer rather than an array
00417    because some compilers will just produce instructions to fill the
00418    array rather than assigning a pointer to a static array.  */
00419 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00420 #ifdef SIMULATE_ID
00421 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00422 #endif
00423
00424 #ifdef __QNXNTO__
00425 char const* qnxnto = "INFO" ":" "qnxnto[]";
00426 #endif
00427
00428 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00429 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
00430 #endif
00431
00432 #define STRINGIFY_HELPER(X) #X
00433 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00434
00435 /* Identify known platforms by name.  */
00436 #if defined(__linux) || defined(__linux__) || defined(linux)
00437 # define PLATFORM_ID "Linux"
00438
00439 #elif defined(__MSYS__)
00440 # define PLATFORM_ID "MSYS"
00441
00442 #elif defined(__CYGWIN__)
00443 # define PLATFORM_ID "Cygwin"
00444
00445 #elif defined(__MINGW32__)
00446 # define PLATFORM_ID "MinGW"
00447
00448 #elif defined(__APPLE__)
00449 # define PLATFORM_ID "Darwin"
00450
00451 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00452 # define PLATFORM_ID "Windows"
00453
00454 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00455 # define PLATFORM_ID "FreeBSD"
00456
00457 #elif defined(__NetBSD__) || defined(__NetBSD)
00458 # define PLATFORM_ID "NetBSD"
00459
00460 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00461 # define PLATFORM_ID "OpenBSD"
00462
00463 #elif defined(__sun) || defined(sun)
00464 # define PLATFORM_ID "SunOS"
00465
00466 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00467 # define PLATFORM_ID "AIX"
00468
00469 #elif defined(__hpux) || defined(__hpux__)
00470 # define PLATFORM_ID "HP-UX"
00471
00472 #elif defined(__HAIKU__)
00473 # define PLATFORM_ID "Haiku"
00474
00475 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00476 # define PLATFORM_ID "BeOS"
00477
00478 #elif defined(__QNX__) || defined(__QNXNTO__)
00479 # define PLATFORM_ID "QNX"
00480
00481 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00482 # define PLATFORM_ID "Tru64"
00483
00484 #elif defined(__riscos) || defined(__riscos__)
00485 # define PLATFORM_ID "RISCos"
00486
00487 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00488 # define PLATFORM_ID "SINIX"
00489
00490 #elif defined(__UNIX_SV__)
00491 # define PLATFORM_ID "UNIX_SV"
00492
00493 #elif defined(__bsdos__)
00494 # define PLATFORM_ID "BSDOS"
00495
00496 #elif defined(_MPRAS) || defined(MPRAS)
00497 # define PLATFORM_ID "MP-RAS"
00498
00499 #elif defined(__osf) || defined(__osf__)
00500 # define PLATFORM_ID "OSF1"
00501
```

```
00502 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00503 # define PLATFORM_ID "SCO_SV"
00504
00505 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00506 # define PLATFORM_ID "ULTRIX"
00507
00508 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00509 # define PLATFORM_ID "Xenix"
00510
00511 #elif defined(__WATCOMC__)
00512 # if defined(__LINUX__)
00513 #  define PLATFORM_ID "Linux"
00514
00515 # elif defined(__DOS__)
00516 #  define PLATFORM_ID "DOS"
00517
00518 # elif defined(__OS2__)
00519 #  define PLATFORM_ID "OS2"
00520
00521 # elif defined(__WINDOWS__)
00522 #  define PLATFORM_ID "Windows3x"
00523
00524 # elif defined(__VXWORKS__)
00525 #  define PLATFORM_ID "VxWorks"
00526
00527 # else /* unknown platform */
00528 #  define PLATFORM_ID
00529 # endif
00530
00531 #elif defined(__INTEGRITY)
00532 # if defined(INT_178B)
00533 #  define PLATFORM_ID "Integrity178"
00534
00535 # else /* regular Integrity */
00536 #  define PLATFORM_ID "Integrity"
00537 # endif
00538
00539 #else /* unknown platform */
00540 # define PLATFORM_ID
00541
00542 #endif
00543
00544 /* For windows compilers MSVC and Intel we can determine
00545    the architecture of the compiler being used.  This is because
00546    the compilers do not have flags that can change the architecture,
00547    but rather depend on which compiler is being used
00548 */
00549 #if defined(_WIN32) && defined(_MSC_VER)
00550 # if defined(_M_IA64)
00551 #  define ARCHITECTURE_ID "IA64"
00552
00553 # elif defined(_M_ARM64EC)
00554 #  define ARCHITECTURE_ID "ARM64EC"
00555
00556 # elif defined(_M_X64) || defined(_M_AMD64)
00557 #  define ARCHITECTURE_ID "x64"
00558
00559 # elif defined(_M_IX86)
00560 #  define ARCHITECTURE_ID "X86"
00561
00562 # elif defined(_M_ARM64)
00563 #  define ARCHITECTURE_ID "ARM64"
00564
00565 # elif defined(_M_ARM)
00566 #  if _M_ARM == 4
00567 #   define ARCHITECTURE_ID "ARMV4I"
00568 #  elif _M_ARM == 5
00569 #   define ARCHITECTURE_ID "ARMV5I"
00570 #  else
00571 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00572 #  endif
00573
00574 # elif defined(_M_MIPS)
00575 #  define ARCHITECTURE_ID "MIPS"
00576
00577 # elif defined(_M_SH)
00578 #  define ARCHITECTURE_ID "SHx"
00579
00580 # else /* unknown architecture */
00581 #  define ARCHITECTURE_ID ""
00582 # endif
00583
00584 #elif defined(__WATCOMC__)
00585 # if defined(_M_I86)
00586 #  define ARCHITECTURE_ID "I86"
00587
00588 # elif defined(_M_IX86)
```

```
00589 #  define ARCHITECTURE_ID "X86"
00590
00591 # else /* unknown architecture */
00592 #  define ARCHITECTURE_ID ""
00593 # endif
00594
00595 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00596 # if defined(__ICCARM__)
00597 #  define ARCHITECTURE_ID "ARM"
00598
00599 # elif defined(__ICCRX__)
00600 #  define ARCHITECTURE_ID "RX"
00601
00602 # elif defined(__ICCRH850__)
00603 #  define ARCHITECTURE_ID "RH850"
00604
00605 # elif defined(__ICCRL78__)
00606 #  define ARCHITECTURE_ID "RL78"
00607
00608 # elif defined(__ICCRISCV__)
00609 #  define ARCHITECTURE_ID "RISCV"
00610
00611 # elif defined(__ICCAVR__)
00612 #  define ARCHITECTURE_ID "AVR"
00613
00614 # elif defined(__ICC430__)
00615 #  define ARCHITECTURE_ID "MSP430"
00616
00617 # elif defined(__ICCV850__)
00618 #  define ARCHITECTURE_ID "V850"
00619
00620 # elif defined(__ICC8051__)
00621 #  define ARCHITECTURE_ID "8051"
00622
00623 # elif defined(__ICCSTM8__)
00624 #  define ARCHITECTURE_ID "STM8"
00625
00626 # else /* unknown architecture */
00627 #  define ARCHITECTURE_ID ""
00628 # endif
00629
00630 #elif defined(__ghs__)
00631 # if defined(__PPC64__)
00632 #  define ARCHITECTURE_ID "PPC64"
00633
00634 # elif defined(__ppc__)
00635 #  define ARCHITECTURE_ID "PPC"
00636
00637 # elif defined(__ARM__)
00638 #  define ARCHITECTURE_ID "ARM"
00639
00640 # elif defined(__x86_64__)
00641 #  define ARCHITECTURE_ID "x64"
00642
00643 # elif defined(__i386__)
00644 #  define ARCHITECTURE_ID "X86"
00645
00646 # else /* unknown architecture */
00647 #  define ARCHITECTURE_ID ""
00648 # endif
00649
00650 #elif defined(__TI_COMPILER_VERSION__)
00651 # if defined(__TI_ARM__)
00652 #  define ARCHITECTURE_ID "ARM"
00653
00654 # elif defined(__MSP430__)
00655 #  define ARCHITECTURE_ID "MSP430"
00656
00657 # elif defined(__TMS320C28XX__)
00658 #  define ARCHITECTURE_ID "TMS320C28x"
00659
00660 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00661 #  define ARCHITECTURE_ID "TMS320C6x"
00662
00663 # else /* unknown architecture */
00664 #  define ARCHITECTURE_ID ""
00665 # endif
00666
00667 #else
00668 #  define ARCHITECTURE_ID
00669 #endif
00670
00671 /* Convert integer to decimal digit literals.  */
00672 #define DEC(n)                   \
00673   ('0' + (((n) / 10000000)%10)), \
00674   ('0' + (((n) / 1000000)%10)),  \
00675   ('0' + (((n) / 100000)%10)),   \
```

```
00676   ('0' + (((n) / 10000)%10)),     \
00677   ('0' + (((n) / 1000)%10)),      \
00678   ('0' + (((n) / 100)%10)),       \
00679   ('0' + (((n) / 10)%10)),        \
00680   ('0' +  ((n) % 10))
00681
00682 /* Convert integer to hex digit literals.  */
00683 #define HEX(n)             \
00684   ('0' + ((n)>>28 & 0xF)), \
00685   ('0' + ((n)>>24 & 0xF)), \
00686   ('0' + ((n)>>20 & 0xF)), \
00687   ('0' + ((n)>>16 & 0xF)), \
00688   ('0' + ((n)>>12 & 0xF)), \
00689   ('0' + ((n)>>8  & 0xF)), \
00690   ('0' + ((n)>>4  & 0xF)), \
00691   ('0' + ((n)     & 0xF))
00692
00693 /* Construct a string literal encoding the version number. */
00694 #ifdef COMPILER_VERSION
00695 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "]";
00696
00697 /* Construct a string literal encoding the version number components. */
00698 #elif defined(COMPILER_VERSION_MAJOR)
00699 char const info_version[] = {
00700   'I', 'N', 'F', 'O', ':',
00701   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00702   COMPILER_VERSION_MAJOR,
00703 # ifdef COMPILER_VERSION_MINOR
00704   '.', COMPILER_VERSION_MINOR,
00705 #  ifdef COMPILER_VERSION_PATCH
00706   '.', COMPILER_VERSION_PATCH,
00707 #   ifdef COMPILER_VERSION_TWEAK
00708     '.', COMPILER_VERSION_TWEAK,
00709 #   endif
00710 #  endif
00711 # endif
00712   ']','\0'};
00713 #endif
00714
00715 /* Construct a string literal encoding the internal version number. */
00716 #ifdef COMPILER_VERSION_INTERNAL
00717 char const info_version_internal[] = {
00718   'I', 'N', 'F', 'O', ':',
00719   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00720   'i','n','t','e','r','n','a','l','[',
00721   COMPILER_VERSION_INTERNAL,']','\0'};
00722 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00723 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
      COMPILER_VERSION_INTERNAL_STR "]";
00724 #endif
00725
00726 /* Construct a string literal encoding the version number components. */
00727 #ifdef SIMULATE_VERSION_MAJOR
00728 char const info_simulate_version[] = {
00729   'I', 'N', 'F', 'O', ':',
00730   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00731   SIMULATE_VERSION_MAJOR,
00732 # ifdef SIMULATE_VERSION_MINOR
00733   '.', SIMULATE_VERSION_MINOR,
00734 #  ifdef SIMULATE_VERSION_PATCH
00735   '.', SIMULATE_VERSION_PATCH,
00736 #   ifdef SIMULATE_VERSION_TWEAK
00737     '.', SIMULATE_VERSION_TWEAK,
00738 #   endif
00739 #  endif
00740 # endif
00741   ']','\0'};
00742 #endif
00743
00744 /* Construct the string literal in pieces to prevent the source from
00745    getting matched.  Store it in a pointer rather than an array
00746    because some compilers will just produce instructions to fill the
00747    array rather than assigning a pointer to a static array.  */
00748 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00749 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00750
00751
00752
00753 #if !defined(__STDC__) && !defined(__clang__)
00754 # if defined(_MSC_VER) || defined(__ibmxl__) || defined(__IBMC__)
00755 #  define C_DIALECT "90"
00756 # else
00757 #  define C_DIALECT
00758 # endif
00759 #elif __STDC_VERSION__ > 201710L
00760 # define C_DIALECT "23"
00761 #elif __STDC_VERSION__ >= 201710L
```

```
00762 # define C_DIALECT "17"
00763 #elif __STDC_VERSION__ >= 201000L
00764 # define C_DIALECT "11"
00765 #elif __STDC_VERSION__ >= 199901L
00766 # define C_DIALECT "99"
00767 #else
00768 # define C_DIALECT "90"
00769 #endif
00770 const char* info_language_dialect_default =
00771   "INFO" ":" "dialect_default[" C_DIALECT "]";
00772
00773 /*--------------------------------------------------------------------------*/
00774
00775 #ifdef ID_VOID_MAIN
00776 void main() {}
00777 #else
00778 # if defined(__CLASSIC_C__)
00779 int main(argc, argv) int argc; char *argv[];
00780 # else
00781 int main(int argc, char* argv[])
00782 # endif
00783 {
00784   int require = 0;
00785   require += info_compiler[argc];
00786   require += info_platform[argc];
00787   require += info_arch[argc];
00788 #ifdef COMPILER_VERSION_MAJOR
00789   require += info_version[argc];
00790 #endif
00791 #ifdef COMPILER_VERSION_INTERNAL
00792   require += info_version_internal[argc];
00793 #endif
00794 #ifdef SIMULATE_ID
00795   require += info_simulate[argc];
00796 #endif
00797 #ifdef SIMULATE_VERSION_MAJOR
00798   require += info_simulate_version[argc];
00799 #endif
00800 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00801   require += info_cray[argc];
00802 #endif
00803   require += info_language_dialect_default[argc];
00804   (void)argv;
00805   return require;
00806 }
00807 #endif
```

7.5 out/build/x64-Debug/CMakeFiles/3.21.21080301-MSVC_2/CompilerIdCXX/CMakeCXXCompilerId.cpp File Reference

Macros

- #define __has_include(x) 0
- #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

7.5.1 Macro Definition Documentation

7.5.1.1 __has_include

```
#define __has_include(
              x ) 0
```

Definition at line 11 of file CMakeCXXCompilerId.cpp.

7.5.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 653 of file CMakeCXXCompilerId.cpp.

7.5.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 397 of file CMakeCXXCompilerId.cpp.

7.5.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 751 of file CMakeCXXCompilerId.cpp.

7.5.1.5 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 657 of file CMakeCXXCompilerId.cpp.

7.5.1.6 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 668 of file CMakeCXXCompilerId.cpp.

7.5.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 525 of file CMakeCXXCompilerId.cpp.

7.5.1.8 STRINGIFY

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

Definition at line 418 of file CMakeCXXCompilerId.cpp.

7.5.1.9 STRINGIFY_HELPER

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

Definition at line 417 of file CMakeCXXCompilerId.cpp.

7.5.2 Function Documentation

7.5.2.1 main()

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

Definition at line 772 of file CMakeCXXCompilerId.cpp.

7.5.3 Variable Documentation

7.5.3.1 info_arch

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

Definition at line 734 of file CMakeCXXCompilerId.cpp.

7.5.3.2 info_compiler

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

Definition at line 404 of file CMakeCXXCompilerId.cpp.

7.5.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

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

Definition at line 754 of file CMakeCXXCompilerId.cpp.

7.5.3.4 info_platform

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

Definition at line 733 of file CMakeCXXCompilerId.cpp.

7.6 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 #if !defined(__has_include)
00009 /* If the compiler does not have __has_include, pretend the answer is
00010    always no.  */
00011 #  define __has_include(x) 0
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(__COMO__)
00019 # define COMPILER_ID "Comeau"
00020   /* __COMO_VERSION__ = VRR */
00021 # define COMPILER_VERSION_MAJOR DEC(__COMO_VERSION__ / 100)
00022 # define COMPILER_VERSION_MINOR DEC(__COMO_VERSION__ % 100)
00023
00024 #elif defined(__INTEL_COMPILER) || defined(__ICC)
00025 # define COMPILER_ID "Intel"
00026 # if defined(_MSC_VER)
00027 #  define SIMULATE_ID "MSVC"
00028 # endif
00029 # if defined(__GNUC__)
00030 #  define SIMULATE_ID "GNU"
00031 # endif
00032   /* __INTEL_COMPILER = VRP prior to 2021, and then VVVV for 2021 and later,
00033      except that a few beta releases use the old format with V=2021.  */
00034 # if __INTEL_COMPILER < 2021 || __INTEL_COMPILER == 202110 || __INTEL_COMPILER == 202111
00035 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER/100)
00036 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER/10 % 10)
00037 #  if defined(__INTEL_COMPILER_UPDATE)
00038 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER_UPDATE)
00039 #  else
00040 #   define COMPILER_VERSION_PATCH DEC(__INTEL_COMPILER   % 10)
00041 #  endif
00042 # else
00043 #  define COMPILER_VERSION_MAJOR DEC(__INTEL_COMPILER)
00044 #  define COMPILER_VERSION_MINOR DEC(__INTEL_COMPILER_UPDATE)
00045   /* The third version component from --version is an update index,
00046      but no macro is provided for it.  */
00047 #  define COMPILER_VERSION_PATCH DEC(0)
00048 # endif
00049 # if defined(__INTEL_COMPILER_BUILD_DATE)
00050   /* __INTEL_COMPILER_BUILD_DATE = YYYYMMDD */
00051 #  define COMPILER_VERSION_TWEAK DEC(__INTEL_COMPILER_BUILD_DATE)
00052 # endif
00053 # if defined(_MSC_VER)
00054   /* _MSC_VER = VVRR */
00055 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00056 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00057 # endif
00058 # if defined(__GNUC__)
00059 #  define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00060 # elif defined(__GNUG__)
00061 #  define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00062 # endif
00063 # if defined(__GNUC_MINOR__)
00064 #  define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00065 # endif
00066 # if defined(__GNUC_PATCHLEVEL__)
00067 #  define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00068 # endif
00069
```

```
00070 #elif (defined(__clang__) && defined(__INTEL_CLANG_COMPILER)) || defined(__INTEL_LLVM_COMPILER)
00071 # define COMPILER_ID "IntelLLVM"
00072 #if defined(_MSC_VER)
00073 # define SIMULATE_ID "MSVC"
00074 #endif
00075 #if defined(__GNUC__)
00076 # define SIMULATE_ID "GNU"
00077 #endif
00078 /* __INTEL_LLVM_COMPILER = VVVVRP prior to 2021.2.0, VVVVRRPP for 2021.2.0 and
00079  * later.  Look for 6 digit vs. 8 digit version number to decide encoding.
00080  * VVVV is no smaller than the current year when a version is released.
00081  */
00082 #if __INTEL_LLVM_COMPILER < 1000000L
00083 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/100)
00084 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/10 % 10)
00085 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER    % 10)
00086 #else
00087 # define COMPILER_VERSION_MAJOR DEC(__INTEL_LLVM_COMPILER/10000)
00088 # define COMPILER_VERSION_MINOR DEC(__INTEL_LLVM_COMPILER/100 % 100)
00089 # define COMPILER_VERSION_PATCH DEC(__INTEL_LLVM_COMPILER     % 100)
00090 #endif
00091 #if defined(_MSC_VER)
00092   /* _MSC_VER = VVRR */
00093 # define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00094 # define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00095 #endif
00096 #if defined(__GNUC__)
00097 # define SIMULATE_VERSION_MAJOR DEC(__GNUC__)
00098 #elif defined(__GNUG__)
00099 # define SIMULATE_VERSION_MAJOR DEC(__GNUG__)
00100 #endif
00101 #if defined(__GNUC_MINOR__)
00102 # define SIMULATE_VERSION_MINOR DEC(__GNUC_MINOR__)
00103 #endif
00104 #if defined(__GNUC_PATCHLEVEL__)
00105 # define SIMULATE_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00106 #endif
00107
00108 #elif defined(__PATHCC__)
00109 # define COMPILER_ID "PathScale"
00110 # define COMPILER_VERSION_MAJOR DEC(__PATHCC__)
00111 # define COMPILER_VERSION_MINOR DEC(__PATHCC_MINOR__)
00112 # if defined(__PATHCC_PATCHLEVEL__)
00113 #  define COMPILER_VERSION_PATCH DEC(__PATHCC_PATCHLEVEL__)
00114 # endif
00115
00116 #elif defined(__BORLANDC__) && defined(__CODEGEARC_VERSION__)
00117 # define COMPILER_ID "Embarcadero"
00118 # define COMPILER_VERSION_MAJOR HEX(__CODEGEARC_VERSION__»24 & 0x00FF)
00119 # define COMPILER_VERSION_MINOR HEX(__CODEGEARC_VERSION__»16 & 0x00FF)
00120 # define COMPILER_VERSION_PATCH DEC(__CODEGEARC_VERSION__     & 0xFFFF)
00121
00122 #elif defined(__BORLANDC__)
00123 # define COMPILER_ID "Borland"
00124   /* __BORLANDC__ = 0xVRR */
00125 # define COMPILER_VERSION_MAJOR HEX(__BORLANDC__»8)
00126 # define COMPILER_VERSION_MINOR HEX(__BORLANDC__ & 0xFF)
00127
00128 #elif defined(__WATCOMC__) && __WATCOMC__ < 1200
00129 # define COMPILER_ID "Watcom"
00130    /* __WATCOMC__ = VVRR */
00131 # define COMPILER_VERSION_MAJOR DEC(__WATCOMC__ / 100)
00132 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00133 # if (__WATCOMC__ % 10) > 0
00134 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00135 # endif
00136
00137 #elif defined(__WATCOMC__)
00138 # define COMPILER_ID "OpenWatcom"
00139    /* __WATCOMC__ = VVRP + 1100 */
00140 # define COMPILER_VERSION_MAJOR DEC((__WATCOMC__ - 1100) / 100)
00141 # define COMPILER_VERSION_MINOR DEC((__WATCOMC__ / 10) % 10)
00142 # if (__WATCOMC__ % 10) > 0
00143 #  define COMPILER_VERSION_PATCH DEC(__WATCOMC__ % 10)
00144 # endif
00145
00146 #elif defined(__SUNPRO_CC)
00147 # define COMPILER_ID "SunPro"
00148 # if __SUNPRO_CC >= 0x5100
00149    /* __SUNPRO_CC = 0xVRRP */
00150 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»12)
00151 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xFF)
00152 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00153 # else
00154    /* __SUNPRO_CC = 0xVRP */
00155 #  define COMPILER_VERSION_MAJOR HEX(__SUNPRO_CC»8)
00156 #  define COMPILER_VERSION_MINOR HEX(__SUNPRO_CC»4 & 0xF)
```

```
00157 #  define COMPILER_VERSION_PATCH HEX(__SUNPRO_CC    & 0xF)
00158 # endif
00159
00160 #elif defined(__HP_aCC)
00161 # define COMPILER_ID "HP"
00162   /* __HP_aCC = VVRRPP */
00163 # define COMPILER_VERSION_MAJOR DEC(__HP_aCC/10000)
00164 # define COMPILER_VERSION_MINOR DEC(__HP_aCC/100 % 100)
00165 # define COMPILER_VERSION_PATCH DEC(__HP_aCC     % 100)
00166
00167 #elif defined(__DECCXX)
00168 # define COMPILER_ID "Compaq"
00169   /* __DECCXX_VER = VVRRTPPPP */
00170 # define COMPILER_VERSION_MAJOR DEC(__DECCXX_VER/10000000)
00171 # define COMPILER_VERSION_MINOR DEC(__DECCXX_VER/100000  % 100)
00172 # define COMPILER_VERSION_PATCH DEC(__DECCXX_VER         % 10000)
00173
00174 #elif defined(__IBMCPP__) && defined(__COMPILER_VER__)
00175 # define COMPILER_ID "zOS"
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(__ibmxl__) && defined(__clang__)
00182 # define COMPILER_ID "XLClang"
00183 # define COMPILER_VERSION_MAJOR DEC(__ibmxl_version__)
00184 # define COMPILER_VERSION_MINOR DEC(__ibmxl_release__)
00185 # define COMPILER_VERSION_PATCH DEC(__ibmxl_modification__)
00186 # define COMPILER_VERSION_TWEAK DEC(__ibmxl_ptf_fix_level__)
00187
00188
00189 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ >= 800
00190 # define COMPILER_ID "XL"
00191   /* __IBMCPP__ = VRP */
00192 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00193 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00194 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00195
00196 #elif defined(__IBMCPP__) && !defined(__COMPILER_VER__) && __IBMCPP__ < 800
00197 # define COMPILER_ID "VisualAge"
00198   /* __IBMCPP__ = VRP */
00199 # define COMPILER_VERSION_MAJOR DEC(__IBMCPP__/100)
00200 # define COMPILER_VERSION_MINOR DEC(__IBMCPP__/10 % 10)
00201 # define COMPILER_VERSION_PATCH DEC(__IBMCPP__    % 10)
00202
00203 #elif defined(__NVCOMPILER)
00204 # define COMPILER_ID "NVHPC"
00205 # define COMPILER_VERSION_MAJOR DEC(__NVCOMPILER_MAJOR__)
00206 # define COMPILER_VERSION_MINOR DEC(__NVCOMPILER_MINOR__)
00207 # if defined(__NVCOMPILER_PATCHLEVEL__)
00208 #  define COMPILER_VERSION_PATCH DEC(__NVCOMPILER_PATCHLEVEL__)
00209 # endif
00210
00211 #elif defined(__PGI)
00212 # define COMPILER_ID "PGI"
00213 # define COMPILER_VERSION_MAJOR DEC(__PGIC__)
00214 # define COMPILER_VERSION_MINOR DEC(__PGIC_MINOR__)
00215 # if defined(__PGIC_PATCHLEVEL__)
00216 #  define COMPILER_VERSION_PATCH DEC(__PGIC_PATCHLEVEL__)
00217 # endif
00218
00219 #elif defined(_CRAYC)
00220 # define COMPILER_ID "Cray"
00221 # define COMPILER_VERSION_MAJOR DEC(_RELEASE_MAJOR)
00222 # define COMPILER_VERSION_MINOR DEC(_RELEASE_MINOR)
00223
00224 #elif defined(__TI_COMPILER_VERSION__)
00225 # define COMPILER_ID "TI"
00226   /* __TI_COMPILER_VERSION__ = VVVRRRPPP */
00227 # define COMPILER_VERSION_MAJOR DEC(__TI_COMPILER_VERSION__/1000000)
00228 # define COMPILER_VERSION_MINOR DEC(__TI_COMPILER_VERSION__/1000   % 1000)
00229 # define COMPILER_VERSION_PATCH DEC(__TI_COMPILER_VERSION__        % 1000)
00230
00231 #elif defined(__CLANG_FUJITSU)
00232 # define COMPILER_ID "FujitsuClang"
00233 # define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00234 # define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00235 # define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00236 # define COMPILER_VERSION_INTERNAL_STR __clang_version__
00237
00238
00239 #elif defined(__FUJITSU)
00240 # define COMPILER_ID "Fujitsu"
00241 # if defined(__FCC_version__)
00242 #   define COMPILER_VERSION __FCC_version__
00243 # elif defined(__FCC_major__)
```

```
00244 #   define COMPILER_VERSION_MAJOR DEC(__FCC_major__)
00245 #   define COMPILER_VERSION_MINOR DEC(__FCC_minor__)
00246 #   define COMPILER_VERSION_PATCH DEC(__FCC_patchlevel__)
00247 # endif
00248 # if defined(__fcc_version)
00249 #   define COMPILER_VERSION_INTERNAL DEC(__fcc_version)
00250 # elif defined(__FCC_VERSION)
00251 #   define COMPILER_VERSION_INTERNAL DEC(__FCC_VERSION)
00252 # endif
00253
00254
00255 #elif defined(__ghs__)
00256 # define COMPILER_ID "GHS"
00257 /* __GHS_VERSION_NUMBER = VVVVRP */
00258 # ifdef __GHS_VERSION_NUMBER
00259 # define COMPILER_VERSION_MAJOR DEC(__GHS_VERSION_NUMBER / 100)
00260 # define COMPILER_VERSION_MINOR DEC(__GHS_VERSION_NUMBER / 10 % 10)
00261 # define COMPILER_VERSION_PATCH DEC(__GHS_VERSION_NUMBER      % 10)
00262 # endif
00263
00264 #elif defined(__SCO_VERSION__)
00265 # define COMPILER_ID "SCO"
00266
00267 #elif defined(__ARMCC_VERSION) && !defined(__clang__)
00268 # define COMPILER_ID "ARMCC"
00269 #if __ARMCC_VERSION >= 1000000
00270   /* __ARMCC_VERSION = VRRPPPP */
00271   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/1000000)
00272   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 100)
00273   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION     % 10000)
00274 #else
00275   /* __ARMCC_VERSION = VRPPPP */
00276   # define COMPILER_VERSION_MAJOR DEC(__ARMCC_VERSION/100000)
00277   # define COMPILER_VERSION_MINOR DEC(__ARMCC_VERSION/10000 % 10)
00278   # define COMPILER_VERSION_PATCH DEC(__ARMCC_VERSION    % 10000)
00279 #endif
00280
00281
00282 #elif defined(__clang__) && defined(__apple_build_version__)
00283 # define COMPILER_ID "AppleClang"
00284 # if defined(_MSC_VER)
00285 #  define SIMULATE_ID "MSVC"
00286 # endif
00287 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00288 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00289 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00290 # if defined(_MSC_VER)
00291   /* _MSC_VER = VVRR */
00292 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
00293 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00294 # endif
00295 # define COMPILER_VERSION_TWEAK DEC(__apple_build_version__)
00296
00297 #elif defined(__clang__) && defined(__ARMCOMPILER_VERSION)
00298 # define COMPILER_ID "ARMClang"
00299   # define COMPILER_VERSION_MAJOR DEC(__ARMCOMPILER_VERSION/1000000)
00300   # define COMPILER_VERSION_MINOR DEC(__ARMCOMPILER_VERSION/10000 % 100)
00301   # define COMPILER_VERSION_PATCH DEC(__ARMCOMPILER_VERSION     % 10000)
00302 # define COMPILER_VERSION_INTERNAL DEC(__ARMCOMPILER_VERSION)
00303
00304 #elif defined(__clang__) && __has_include(<hip/hip_version.h>)
00305 # define COMPILER_ID "ROCMClang"
00306 # if defined(_MSC_VER)
00307 #  define SIMULATE_ID "MSVC"
00308 # elif defined(__clang__)
00309 #  define SIMULATE_ID "Clang"
00310 # elif defined(__GNUC__)
00311 #  define SIMULATE_ID "GNU"
00312 # endif
00313 # if defined(__clang__) && __has_include(<hip/hip_version.h>)
00314 #  include <hip/hip_version.h>
00315 #  define COMPILER_VERSION_MAJOR DEC(HIP_VERSION_MAJOR)
00316 #  define COMPILER_VERSION_MINOR DEC(HIP_VERSION_MINOR)
00317 #  define COMPILER_VERSION_PATCH DEC(HIP_VERSION_PATCH)
00318 # endif
00319
00320 #elif defined(__clang__)
00321 # define COMPILER_ID "Clang"
00322 # if defined(_MSC_VER)
00323 #  define SIMULATE_ID "MSVC"
00324 # endif
00325 # define COMPILER_VERSION_MAJOR DEC(__clang_major__)
00326 # define COMPILER_VERSION_MINOR DEC(__clang_minor__)
00327 # define COMPILER_VERSION_PATCH DEC(__clang_patchlevel__)
00328 # if defined(_MSC_VER)
00329   /* _MSC_VER = VVRR */
00330 #  define SIMULATE_VERSION_MAJOR DEC(_MSC_VER / 100)
```

```
00331 #  define SIMULATE_VERSION_MINOR DEC(_MSC_VER % 100)
00332 # endif
00333
00334 #elif defined(__GNUC__) || defined(__GNUG__)
00335 # define COMPILER_ID "GNU"
00336 # if defined(__GNUC__)
00337 #  define COMPILER_VERSION_MAJOR DEC(__GNUC__)
00338 # else
00339 #  define COMPILER_VERSION_MAJOR DEC(__GNUG__)
00340 # endif
00341 # if defined(__GNUC_MINOR__)
00342 #  define COMPILER_VERSION_MINOR DEC(__GNUC_MINOR__)
00343 # endif
00344 # if defined(__GNUC_PATCHLEVEL__)
00345 #  define COMPILER_VERSION_PATCH DEC(__GNUC_PATCHLEVEL__)
00346 # endif
00347
00348 #elif defined(_MSC_VER)
00349 # define COMPILER_ID "MSVC"
00350   /* _MSC_VER = VVRR */
00351 # define COMPILER_VERSION_MAJOR DEC(_MSC_VER / 100)
00352 # define COMPILER_VERSION_MINOR DEC(_MSC_VER % 100)
00353 # if defined(_MSC_FULL_VER)
00354 #  if _MSC_VER >= 1400
00355     /* _MSC_FULL_VER = VVRRPPPPP */
00356 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 100000)
00357 #  else
00358     /* _MSC_FULL_VER = VVRRPPPP */
00359 #   define COMPILER_VERSION_PATCH DEC(_MSC_FULL_VER % 10000)
00360 #  endif
00361 # endif
00362 # if defined(_MSC_BUILD)
00363 #  define COMPILER_VERSION_TWEAK DEC(_MSC_BUILD)
00364 # endif
00365
00366 #elif defined(__VISUALDSPVERSION__) || defined(__ADSPBLACKFIN__) || defined(__ADSPTS__) ||
       defined(__ADSP21000__)
00367 # define COMPILER_ID "ADSP"
00368 #if defined(__VISUALDSPVERSION__)
00369   /* __VISUALDSPVERSION__ = 0xVVRRPP00 */
00370 # define COMPILER_VERSION_MAJOR HEX(__VISUALDSPVERSION__»24)
00371 # define COMPILER_VERSION_MINOR HEX(__VISUALDSPVERSION__»16 & 0xFF)
00372 # define COMPILER_VERSION_PATCH HEX(__VISUALDSPVERSION__»8  & 0xFF)
00373 #endif
00374
00375 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00376 # define COMPILER_ID "IAR"
00377 # if defined(__VER__) && defined(__ICCARM__)
00378 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 1000000)
00379 #  define COMPILER_VERSION_MINOR DEC(((__VER__) / 1000) % 1000)
00380 #  define COMPILER_VERSION_PATCH DEC((__VER__) % 1000)
00381 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00382 # elif defined(__VER__) && (defined(__ICCAVR__) || defined(__ICCRX__) || defined(__ICCRH850__) ||
       defined(__ICCRL78__) || defined(__ICC430__) || defined(__ICCRISCV__) || defined(__ICCV850__) ||
       defined(__ICC8051__) || defined(__ICCSTM8__))
00383 #  define COMPILER_VERSION_MAJOR DEC((__VER__) / 100)
00384 #  define COMPILER_VERSION_MINOR DEC((__VER__) - (((__VER__) / 100)*100))
00385 #  define COMPILER_VERSION_PATCH DEC(__SUBVERSION__)
00386 #  define COMPILER_VERSION_INTERNAL DEC(__IAR_SYSTEMS_ICC__)
00387 # endif
00388
00389
00390 /* These compilers are either not known or too old to define an
00391   identification macro.  Try to identify the platform and guess that
00392   it is the native compiler.  */
00393 #elif defined(__hpux) || defined(__hpua)
00394 # define COMPILER_ID "HP"
00395
00396 #else /* unknown compiler */
00397 # define COMPILER_ID ""
00398 #endif
00399
00400 /* Construct the string literal in pieces to prevent the source from
00401    getting matched.  Store it in a pointer rather than an array
00402    because some compilers will just produce instructions to fill the
00403    array rather than assigning a pointer to a static array.  */
00404 char const* info_compiler = "INFO" ":" "compiler[" COMPILER_ID "]";
00405 #ifdef SIMULATE_ID
00406 char const* info_simulate = "INFO" ":" "simulate[" SIMULATE_ID "]";
00407 #endif
00408
00409 #ifdef __QNXNTO__
00410 char const* qnxnto = "INFO" ":" "qnxnto[]";
00411 #endif
00412
00413 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00414 char const *info_cray = "INFO" ":" "compiler_wrapper[CrayPrgEnv]";
```

```
00415 #endif
00416
00417 #define STRINGIFY_HELPER(X) #X
00418 #define STRINGIFY(X) STRINGIFY_HELPER(X)
00419
00420 /* Identify known platforms by name.  */
00421 #if defined(__linux) || defined(__linux__) || defined(linux)
00422 # define PLATFORM_ID "Linux"
00423
00424 #elif defined(__MSYS__)
00425 # define PLATFORM_ID "MSYS"
00426
00427 #elif defined(__CYGWIN__)
00428 # define PLATFORM_ID "Cygwin"
00429
00430 #elif defined(__MINGW32__)
00431 # define PLATFORM_ID "MinGW"
00432
00433 #elif defined(__APPLE__)
00434 # define PLATFORM_ID "Darwin"
00435
00436 #elif defined(_WIN32) || defined(__WIN32__) || defined(WIN32)
00437 # define PLATFORM_ID "Windows"
00438
00439 #elif defined(__FreeBSD__) || defined(__FreeBSD)
00440 # define PLATFORM_ID "FreeBSD"
00441
00442 #elif defined(__NetBSD__) || defined(__NetBSD)
00443 # define PLATFORM_ID "NetBSD"
00444
00445 #elif defined(__OpenBSD__) || defined(__OPENBSD)
00446 # define PLATFORM_ID "OpenBSD"
00447
00448 #elif defined(__sun) || defined(sun)
00449 # define PLATFORM_ID "SunOS"
00450
00451 #elif defined(_AIX) || defined(__AIX) || defined(__AIX__) || defined(__aix) || defined(__aix__)
00452 # define PLATFORM_ID "AIX"
00453
00454 #elif defined(__hpux) || defined(__hpux__)
00455 # define PLATFORM_ID "HP-UX"
00456
00457 #elif defined(__HAIKU__)
00458 # define PLATFORM_ID "Haiku"
00459
00460 #elif defined(__BeOS) || defined(__BEOS__) || defined(_BEOS)
00461 # define PLATFORM_ID "BeOS"
00462
00463 #elif defined(__QNX__) || defined(__QNXNTO__)
00464 # define PLATFORM_ID "QNX"
00465
00466 #elif defined(__tru64) || defined(_tru64) || defined(__TRU64__)
00467 # define PLATFORM_ID "Tru64"
00468
00469 #elif defined(__riscos) || defined(__riscos__)
00470 # define PLATFORM_ID "RISCos"
00471
00472 #elif defined(__sinix) || defined(__sinix__) || defined(__SINIX__)
00473 # define PLATFORM_ID "SINIX"
00474
00475 #elif defined(__UNIX_SV__)
00476 # define PLATFORM_ID "UNIX_SV"
00477
00478 #elif defined(__bsdos__)
00479 # define PLATFORM_ID "BSDOS"
00480
00481 #elif defined(_MPRAS) || defined(MPRAS)
00482 # define PLATFORM_ID "MP-RAS"
00483
00484 #elif defined(__osf) || defined(__osf__)
00485 # define PLATFORM_ID "OSF1"
00486
00487 #elif defined(_SCO_SV) || defined(SCO_SV) || defined(sco_sv)
00488 # define PLATFORM_ID "SCO_SV"
00489
00490 #elif defined(__ultrix) || defined(__ultrix__) || defined(_ULTRIX)
00491 # define PLATFORM_ID "ULTRIX"
00492
00493 #elif defined(__XENIX__) || defined(_XENIX) || defined(XENIX)
00494 # define PLATFORM_ID "Xenix"
00495
00496 #elif defined(__WATCOMC__)
00497 # if defined(__LINUX__)
00498 #  define PLATFORM_ID "Linux"
00499
00500 # elif defined(__DOS__)
00501 #  define PLATFORM_ID "DOS"
```

```
00502
00503 # elif defined(__OS2__)
00504 #  define PLATFORM_ID "OS2"
00505
00506 # elif defined(__WINDOWS__)
00507 #  define PLATFORM_ID "Windows3x"
00508
00509 # elif defined(__VXWORKS__)
00510 #  define PLATFORM_ID "VxWorks"
00511
00512 # else /* unknown platform */
00513 #  define PLATFORM_ID
00514 # endif
00515
00516 #elif defined(__INTEGRITY)
00517 # if defined(INT_178B)
00518 #  define PLATFORM_ID "Integrity178"
00519
00520 # else /* regular Integrity */
00521 #  define PLATFORM_ID "Integrity"
00522 # endif
00523
00524 #else /* unknown platform */
00525 # define PLATFORM_ID
00526
00527 #endif
00528
00529 /* For windows compilers MSVC and Intel we can determine
00530    the architecture of the compiler being used.  This is because
00531    the compilers do not have flags that can change the architecture,
00532    but rather depend on which compiler is being used
00533 */
00534 #if defined(_WIN32) && defined(_MSC_VER)
00535 # if defined(_M_IA64)
00536 #  define ARCHITECTURE_ID "IA64"
00537
00538 # elif defined(_M_ARM64EC)
00539 #  define ARCHITECTURE_ID "ARM64EC"
00540
00541 # elif defined(_M_X64) || defined(_M_AMD64)
00542 #  define ARCHITECTURE_ID "x64"
00543
00544 # elif defined(_M_IX86)
00545 #  define ARCHITECTURE_ID "X86"
00546
00547 # elif defined(_M_ARM64)
00548 #  define ARCHITECTURE_ID "ARM64"
00549
00550 # elif defined(_M_ARM)
00551 #  if _M_ARM == 4
00552 #   define ARCHITECTURE_ID "ARMV4I"
00553 #  elif _M_ARM == 5
00554 #   define ARCHITECTURE_ID "ARMV5I"
00555 #  else
00556 #   define ARCHITECTURE_ID "ARMV" STRINGIFY(_M_ARM)
00557 #  endif
00558
00559 # elif defined(_M_MIPS)
00560 #  define ARCHITECTURE_ID "MIPS"
00561
00562 # elif defined(_M_SH)
00563 #  define ARCHITECTURE_ID "SHx"
00564
00565 # else /* unknown architecture */
00566 #  define ARCHITECTURE_ID ""
00567 # endif
00568
00569 #elif defined(__WATCOMC__)
00570 # if defined(_M_I86)
00571 #  define ARCHITECTURE_ID "I86"
00572
00573 # elif defined(_M_IX86)
00574 #  define ARCHITECTURE_ID "X86"
00575
00576 # else /* unknown architecture */
00577 #  define ARCHITECTURE_ID ""
00578 # endif
00579
00580 #elif defined(__IAR_SYSTEMS_ICC__) || defined(__IAR_SYSTEMS_ICC)
00581 # if defined(__ICCARM__)
00582 #  define ARCHITECTURE_ID "ARM"
00583
00584 # elif defined(__ICCRX__)
00585 #  define ARCHITECTURE_ID "RX"
00586
00587 # elif defined(__ICCRH850__)
00588 #  define ARCHITECTURE_ID "RH850"
```

```
00589
00590 # elif defined(__ICCRL78__)
00591 #  define ARCHITECTURE_ID "RL78"
00592
00593 # elif defined(__ICCRISCV__)
00594 #  define ARCHITECTURE_ID "RISCV"
00595
00596 # elif defined(__ICCAVR__)
00597 #  define ARCHITECTURE_ID "AVR"
00598
00599 # elif defined(__ICC430__)
00600 #  define ARCHITECTURE_ID "MSP430"
00601
00602 # elif defined(__ICCV850__)
00603 #  define ARCHITECTURE_ID "V850"
00604
00605 # elif defined(__ICC8051__)
00606 #  define ARCHITECTURE_ID "8051"
00607
00608 # elif defined(__ICCSTM8__)
00609 #  define ARCHITECTURE_ID "STM8"
00610
00611 # else /* unknown architecture */
00612 #  define ARCHITECTURE_ID ""
00613 # endif
00614
00615 #elif defined(__ghs__)
00616 # if defined(__PPC64__)
00617 #  define ARCHITECTURE_ID "PPC64"
00618
00619 # elif defined(__ppc__)
00620 #  define ARCHITECTURE_ID "PPC"
00621
00622 # elif defined(__ARM__)
00623 #  define ARCHITECTURE_ID "ARM"
00624
00625 # elif defined(__x86_64__)
00626 #  define ARCHITECTURE_ID "x64"
00627
00628 # elif defined(__i386__)
00629 #  define ARCHITECTURE_ID "X86"
00630
00631 # else /* unknown architecture */
00632 #  define ARCHITECTURE_ID ""
00633 # endif
00634
00635 #elif defined(__TI_COMPILER_VERSION__)
00636 # if defined(__TI_ARM__)
00637 #  define ARCHITECTURE_ID "ARM"
00638
00639 # elif defined(__MSP430__)
00640 #  define ARCHITECTURE_ID "MSP430"
00641
00642 # elif defined(__TMS320C28XX__)
00643 #  define ARCHITECTURE_ID "TMS320C28x"
00644
00645 # elif defined(__TMS320C6X__) || defined(_TMS320C6X)
00646 #  define ARCHITECTURE_ID "TMS320C6x"
00647
00648 # else /* unknown architecture */
00649 #  define ARCHITECTURE_ID ""
00650 # endif
00651
00652 #else
00653 #  define ARCHITECTURE_ID
00654 #endif
00655
00656 /* Convert integer to decimal digit literals.  */
00657 #define DEC(n)                   \
00658   ('0' + (((n) / 10000000)%10)), \
00659   ('0' + (((n) / 1000000)%10)),  \
00660   ('0' + (((n) / 100000)%10)),   \
00661   ('0' + (((n) / 10000)%10)),    \
00662   ('0' + (((n) / 1000)%10)),     \
00663   ('0' + (((n) / 100)%10)),      \
00664   ('0' + (((n) / 10)%10)),       \
00665   ('0' +  ((n) % 10))
00666
00667 /* Convert integer to hex digit literals.  */
00668 #define HEX(n)                   \
00669   ('0' + ((n)>>28 & 0xF)), \
00670   ('0' + ((n)>>24 & 0xF)), \
00671   ('0' + ((n)>>20 & 0xF)), \
00672   ('0' + ((n)>>16 & 0xF)), \
00673   ('0' + ((n)>>12 & 0xF)), \
00674   ('0' + ((n)>>8  & 0xF)), \
00675   ('0' + ((n)>>4  & 0xF)), \
```

```
00676   ('0' + ((n)      & 0xF))
00677
00678 /* Construct a string literal encoding the version number. */
00679 #ifdef COMPILER_VERSION
00680 char const* info_version = "INFO" ":" "compiler_version[" COMPILER_VERSION "]";
00681
00682 /* Construct a string literal encoding the version number components. */
00683 #elif defined(COMPILER_VERSION_MAJOR)
00684 char const info_version[] = {
00685   'I', 'N', 'F', 'O', ':',
00686   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','[',
00687   COMPILER_VERSION_MAJOR,
00688 # ifdef COMPILER_VERSION_MINOR
00689   '.', COMPILER_VERSION_MINOR,
00690 #  ifdef COMPILER_VERSION_PATCH
00691    '.', COMPILER_VERSION_PATCH,
00692 #   ifdef COMPILER_VERSION_TWEAK
00693     '.', COMPILER_VERSION_TWEAK,
00694 #   endif
00695 #  endif
00696 # endif
00697   ']','\0'};
00698 #endif
00699
00700 /* Construct a string literal encoding the internal version number. */
00701 #ifdef COMPILER_VERSION_INTERNAL
00702 char const info_version_internal[] = {
00703   'I', 'N', 'F', 'O', ':',
00704   'c','o','m','p','i','l','e','r','_','v','e','r','s','i','o','n','_',
00705   'i','n','t','e','r','n','a','l','[',
00706   COMPILER_VERSION_INTERNAL,']','\0'};
00707 #elif defined(COMPILER_VERSION_INTERNAL_STR)
00708 char const* info_version_internal = "INFO" ":" "compiler_version_internal["
      COMPILER_VERSION_INTERNAL_STR "]";
00709 #endif
00710
00711 /* Construct a string literal encoding the version number components. */
00712 #ifdef SIMULATE_VERSION_MAJOR
00713 char const info_simulate_version[] = {
00714   'I', 'N', 'F', 'O', ':',
00715   's','i','m','u','l','a','t','e','_','v','e','r','s','i','o','n','[',
00716   SIMULATE_VERSION_MAJOR,
00717 # ifdef SIMULATE_VERSION_MINOR
00718   '.', SIMULATE_VERSION_MINOR,
00719 #  ifdef SIMULATE_VERSION_PATCH
00720    '.', SIMULATE_VERSION_PATCH,
00721 #   ifdef SIMULATE_VERSION_TWEAK
00722     '.', SIMULATE_VERSION_TWEAK,
00723 #   endif
00724 #  endif
00725 # endif
00726   ']','\0'};
00727 #endif
00728
00729 /* Construct the string literal in pieces to prevent the source from
00730    getting matched.  Store it in a pointer rather than an array
00731    because some compilers will just produce instructions to fill the
00732    array rather than assigning a pointer to a static array.  */
00733 char const* info_platform = "INFO" ":" "platform[" PLATFORM_ID "]";
00734 char const* info_arch = "INFO" ":" "arch[" ARCHITECTURE_ID "]";
00735
00736
00737
00738 #if defined(__INTEL_COMPILER) && defined(_MSVC_LANG) && _MSVC_LANG < 201403L
00739 #  if defined(__INTEL_CXX11_MODE__)
00740 #    if defined(__cpp_aggregate_nsdmi)
00741 #      define CXX_STD 201402L
00742 #    else
00743 #      define CXX_STD 201103L
00744 #    endif
00745 #  else
00746 #    define CXX_STD 199711L
00747 #  endif
00748 #elif defined(_MSC_VER) && defined(_MSVC_LANG)
00749 #  define CXX_STD _MSVC_LANG
00750 #else
00751 #  define CXX_STD __cplusplus
00752 #endif
00753
00754 const char* info_language_dialect_default = "INFO" ":" "dialect_default["
00755 #if CXX_STD > 202002L
00756   "23"
00757 #elif CXX_STD > 201703L
00758   "20"
00759 #elif CXX_STD >= 201703L
00760   "17"
00761 #elif CXX_STD >= 201402L
```

```
00762   "14"
00763 #elif CXX_STD >= 201103L
00764   "11"
00765 #else
00766   "98"
00767 #endif
00768 "]";
00769
00770 /*--------------------------------------------------------------------------*/
00771
00772 int main(int argc, char* argv[])
00773 {
00774   int require = 0;
00775   require += info_compiler[argc];
00776   require += info_platform[argc];
00777 #ifdef COMPILER_VERSION_MAJOR
00778   require += info_version[argc];
00779 #endif
00780 #ifdef COMPILER_VERSION_INTERNAL
00781   require += info_version_internal[argc];
00782 #endif
00783 #ifdef SIMULATE_ID
00784   require += info_simulate[argc];
00785 #endif
00786 #ifdef SIMULATE_VERSION_MAJOR
00787   require += info_simulate_version[argc];
00788 #endif
00789 #if defined(__CRAYXT_COMPUTE_LINUX_TARGET)
00790   require += info_cray[argc];
00791 #endif
00792   require += info_language_dialect_default[argc];
00793   (void)argv;
00794   return require;
00795 }
```

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

7.8 foo.h

Go to the documentation of this file.

```
00001
```

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

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

Functions

- int main ()

7.9.1 Function Documentation

7.9.1.1 main()

```
int main ( )
```

Definition at line 2 of file main.c.

7.10 main.c

Go to the documentation of this file.

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

7.11 README.md File Reference

7.12 utilstring.cpp File Reference

implementation of own string class.

```
#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::deepCopy (char *rawCharTarget, const char *rawCharSource, size_t destStartPosition=-1, size_t srcEndPosition=-1)

  Deep copy of primitive C-string into another primitive C-string.
- void util::printHeader (const char *text)

  to print a nicely formatted and colored text header to the terminal
- void util::printSubHeader (const char *text)

  to print a nicely formatted and colored text sub header to the terminal
- void util::printTestCase (const char *text)

  to print a nicely formatted and colored text title header to the terminal

7.12.1 Detailed Description

implementation of own string class.

==========

Author

Nour Ahmed @email nahmed@stud.hs-bremen.de, nourbrm02@gmail.com @repo https://github.com/nouremara/cpp_mystring @createdOn 23.11.2022

Version

1.0.0 @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

7.12.2 std::string and this string class are compatible.

Definition in file utilstring.cpp.

7.13 utilstring.cpp

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 1 - Own String Class
00004
00020 #include "utilstring.h"
00021
00022 namespace util {
00023     /*=========================================================================*
00024      *                          Constructors                                   *
00025      *=========================================================================*/
00026
00028     string::string() { intialize_string(INITIAL_SIZE); }
00029
00031     string::string(size_t intialSize) { intialize_string(intialSize); }
00032     //-------------------------------------------------------------
00033
00035     string::string(const char* charArray) {
00036         // ensure string is initialized before using it
00037         intialize_string(rawSize(charArray));
00038         // copy passed array to the string
00039         deepCopy(charArray);
00040     }
00041     //-------------------------------------------------------------
00042
00044     string::string(const std::string& std_string) {
00045         // ensure string is initialized before using it
00046         intialize_string(rawSize(std_string.c_str()));
00047         // copy passed array to the string
00048         deepCopy(std_string.c_str());
00049     }
00050     //-------------------------------------------------------------
00051
00053     string::string(const string& data) {
00054         // ensure string is initialized before using it
00055         intialize_string(rawSize(data.c_str()));
00056         // copy passed array to the string
00057         deepCopy(data.c_str());
00058     }
00059
00060     /*=========================================================================*
00061      *                          Destructor                                     *
```

```
00062      *=========================================================================*/
00063     string::~string(void) { delete[] internal_buffer; }
00064
00065     /*=========================================================================*
00066      *                            Methods                                      *
00067      *=========================================================================*/
00068
00069     void string::intialize_string(size_t length) {
00070         internal_buffer = new char[length + 1];
00071         buffer_size = length + 1;
00072
00073         // initialize an empty string
00074         internal_buffer[0] = '\0';
00075     }
00076     //---------------------------------------------------------------
00077
00078     size_t string::size() const { return rawSize(internal_buffer); }
00079     size_t string::length() const { return rawSize(internal_buffer); }
00080     //---------------------------------------------------------------
00081
00086     size_t string::capacity() const {
00087         return buffer_size;
00088     }
00089     //---------------------------------------------------------------
00090
00096     void string::clear() {
00097         // we only need to set the termination character to the first postion
00098         // to indicate that the string is empty
00099         // initialize an empty string
00100         internal_buffer[0] = '\0';
00101     }
00102
00103     size_t string::rawSize(const char* rawChar) {
00104         size_t length = 0;
00105         while (rawChar[length] != '\0') {
00106             length++;
00107         }
00108
00109         return length;
00110     }
00111     //---------------------------------------------------------------
00112
00120     void string::deepCopy(const char* rawChar, size_t startPosition) {
00121         // check if internalData is of enough size to accommodate the passed array
00122         size_t rawCharSize = rawSize(rawChar);
00123         if (rawCharSize > size()) { // more space is needed
00124           // delete current internalData
00125             delete[] internal_buffer;
00126
00127             // re-initialize the string with the required size
00128             intialize_string(rawCharSize);
00129         }
00130
00131         // copy the passed array to the newly allocated internalData
00132         int j = startPosition;
00133         while (rawChar[j] != '\0') {
00134             internal_buffer[j] = rawChar[j];
00135             j++;
00136         }
00137
00138         internal_buffer[j] = '\0'; // ensure destination string is null terminated
00139         // string_size = rawCharSize;       // set string size to the new one
00140     }
00141     //---------------------------------------------------------------
00142
00143     char* string::c_str() const { return internal_buffer; }
00144     //---------------------------------------------------------------
00145
00146     // Returns a pointer to an array that contains a null-terminated
00147     // sequence of characters(i.e., a C-string) representing the current
00148     // value of the string object.
00149     // Returns a substring object of type util::string which
00150     // starts at pos. Parameter length specifies the amount of
00151     // characters of the new util::string to be returned.
00152     string string::substr(size_t start_position, size_t length) {
00153         // make sure the required length does not exeed the number of characters
00154         // already in the string
00155         size_t string_length = size();
00156         if ((start_position + length) > string_length) {
00157             length = string_length - start_position;
00158         }
00159
00160         // ititasiate a util::string object to return
00161         string substring(length);
00162
00163         //fill the object with the desired substring
00164         util::deepCopy(substring.c_str(), internal_buffer + start_position, 0, length);
```

```
00165
00166         return substring;
00167     }
00168
00169     //-------------------------------------------------------------
00170
00184     int string::compare(const char* lhsCharArray, const char* rhsCharArray) {
00185         // convert both pointers from `char*` to `unsigned char*`
00186         // needed for the difference calculations
00187         const unsigned char* p1 = (const unsigned char*)lhsCharArray;
00188         const unsigned char* p2 = (const unsigned char*)rhsCharArray;
00189
00190         // check if characters differ, or end of the first string (a terminating null)
00191         // is reached
00192         while (*p1 && *p1 == *p2) {
00193             // proceed to the next pair of characters
00194             ++p1, ++p2;
00195         }
00196
00197         // return the ASCII difference
00198         return (*p1 > *p2) - (*p2 > *p1);
00199     }
00200
00201     /*=========================================================================*
00202      *                          Operators                                      *
00203      *=========================================================================*/
00204
00205     string& string::operator=(const string& rhsString) {
00206         deepCopy(rhsString.c_str());
00207         return *this;
00208     }
00209
00210     string& string::operator=(const char* rhsCharArray) {
00211         deepCopy(rhsCharArray);
00212         return *this;
00213     }
00214
00215     string& string::operator=(const std::string& rhsString) {
00216         deepCopy(rhsString.c_str());
00217         return *this;
00218     }
00219     //-------------------------------------------------------------
00220
00221     bool string::operator==(const string& rhsString) {
00222         // if (string_size != rhsString.size()) return false;
00223         // note that compare returns 0 when the two strings are equal
00224         return !compare(internal_buffer, rhsString.c_str());
00225     }
00226
00227     bool string::operator==(const std::string& rhsString) {
00228         // if (string_size != rhsString.size()) return false;
00229
00230         // note that compare returns 0 when the two strings are equal
00231         return !compare(internal_buffer, rhsString.c_str());
00232     }
00233
00234     bool string::operator==(const char* charArray) {
00235         // if (string_size != rhsString.size()) return false;
00236
00237         // note that compare returns 0 when the two strings are equal
00238         return !compare(internal_buffer, charArray);
00239     }
00240     //-------------------------------------------------------------
00241
00242     bool string::operator!=(const string& rhsString) {
00243         // if (string_size != rhsString.size()) return false;
00244         // note that compare returns 0 when the two strings are equal
00245         return compare(internal_buffer, rhsString.c_str());
00246     }
00247
00248     bool string::operator!=(const std::string& rhsString) {
00249         // if (string_size != rhsString.size()) return false;
00250
00251         // note that compare returns 0 when the two strings are equal
00252         return compare(internal_buffer, rhsString.c_str());
00253     }
00254
00255     bool string::operator!=(const char* charArray) {
00256         // if (string_size != rhsString.size()) return false;
00257
00258         // note that compare returns 0 when the two strings are equal
00259         return compare(internal_buffer, charArray);
00260     }
00261     //-------------------------------------------------------------
00262
00263     char& string::operator[](size_t position) {
00264         //if (position > size()) return '\0';
```

```
00265         return internal_buffer[position];
00266     }
00267     //-------------------------------------------------------------
00268
00269     string& string::operator+=(const string& rhsString) {
00270         size_t total_size = size() + rhsString.size() + 1;
00271         char* temp = new char[total_size];
00272
00273         util::deepCopy(temp, internal_buffer, 0);
00274         util::deepCopy(temp, rhsString.c_str(), size());
00275
00276         delete[] internal_buffer;
00277
00278         internal_buffer = temp;
00279         buffer_size = total_size;
00280         //std::cout << "\n\nbuffer_size " << buffer_size << std::endl;
00281         //std::cout << "(size: " << size() << ") : " << internal_buffer << std::endl;
00282
00283         return *this;
00284     }
00285     //-------------------------------------------------------------
00286
00287     string& string::operator+=(const std::string& rhsString) {
00288         size_t total_size = size() + rhsString.size() + 1;
00289         char* temp = new char[total_size];
00290
00291         util::deepCopy(temp, internal_buffer, 0);
00292         util::deepCopy(temp, rhsString.c_str(), size());
00293
00294         delete[] internal_buffer;
00295
00296         internal_buffer = temp;
00297         buffer_size = total_size;
00298
00299         return *this;
00300     }
00301     //-------------------------------------------------------------
00302
00306     string& string::operator+=(const char* rhsString) {
00307         size_t total_size = size() + rawSize(rhsString) + 1;
00308         char* temp = new char[total_size];
00309
00310         util::deepCopy(temp, internal_buffer, 0);
00311         util::deepCopy(temp, rhsString, size());
00312
00313         delete[] internal_buffer;
00314
00315         internal_buffer = temp;
00316         buffer_size = total_size;
00317
00318         return *this;
00319     }
00320     //-------------------------------------------------------------
00321
00322     string string::operator+(const string& rhsString) {
00323         string temp(size() + rhsString.size());
00324
00325         util::deepCopy(temp.c_str(), internal_buffer, 0);
00326         util::deepCopy(temp.c_str(), rhsString.c_str(), size());
00327
00328         return temp;
00329     }
00330     //-------------------------------------------------------------
00331
00332     string string::operator+(const std::string& rhsString) {
00333         string temp(size() + rhsString.size());
00334
00335         util::deepCopy(temp.c_str(), internal_buffer, 0);
00336         util::deepCopy(temp.c_str(), rhsString.c_str(), size());
00337
00338         return temp;
00339     }
00340     //-------------------------------------------------------------
00341
00342     string string::operator+(const char* rhsString) {
00343         string temp(size() + rawSize(rhsString));
00344
00345         util::deepCopy(temp.c_str(), internal_buffer, 0);
00346         util::deepCopy(temp.c_str(), rhsString, size());
00347
00348         return temp;
00349     }
00350
00351     /*============================================================================*
00352      * non-member (friend) functions and operator methods for the cases           *
00353      * util::string is on the RHS                                                 *
00354      *============================================================================*/
```

```
00355
00356     std::ostream& operator«(std::ostream& iostream, const util::string& myString) {
00357         return (iostream « myString.c_str());
00358     }
00359     //-------------------------------------------------------------
00360
00361     bool operator==(const std::string& lhsString, const util::string& rhsString) {
00362         // note that compare returns 0 when the two strings are equal
00363         return !util::string::compare(lhsString.c_str(), rhsString.c_str());
00364     }
00365
00366     bool operator==(const char* lhsCharArray, const util::string& rhsString) {
00367         // note that compare returns 0 when the two strings are equal
00368         return !util::string::compare(lhsCharArray, rhsString.c_str());
00369     }
00370     //-------------------------------------------------------------
00371
00372     bool operator!=(const std::string& lhsString, const util::string& rhsString) {
00373         // note that compare returns 0 when the two strings are equal
00374         return util::string::compare(lhsString.c_str(), rhsString.c_str());
00375     }
00376
00377     bool operator!=(const char* lhsCharArray, const util::string& rhsString) {
00378         // note that compare returns 0 when the two strings are equal
00379         return util::string::compare(lhsCharArray, rhsString.c_str());
00380     }
00381
00382
00383     /*=============================================================================*
00384     *             Some Utility functions                                          *
00385     *=============================================================================*/
00386
00400     void deepCopy(char* rawCharTarget,
00401                   const char* rawCharSource,
00402                   size_t destStartPosition,
00403                   size_t srcEndPosition) {
00404         // check and adjust for default values
00405         destStartPosition = (destStartPosition == -1)
00406             ? util::string::rawSize(rawCharTarget)
00407             : destStartPosition;
00408         srcEndPosition = (srcEndPosition == -1) ? util::string::rawSize(rawCharSource)
00409             : srcEndPosition;
00410
00411         // deep copy rawCharSource into rawCharTarget beginning at startPosition
00412         for (size_t j = 0; j < srcEndPosition; ++j, ++destStartPosition) {
00413             rawCharTarget[destStartPosition] = rawCharSource[j];
00414         }
00415
00416         // ensure destination string is null terminated
00417         rawCharTarget[destStartPosition] = '\0';
00418     }
00419
00420     /*=============================================================================*
00421      *      Some Utility functions for printing nice text output                   *
00422      *=============================================================================*/
00423
00433     void printHeader(const char* text) {
00434         size_t spaces_needed = (80 - util::string::rawSize(text)) / 2 - 2;
00435
00436         std::cout « "\033[1;30;106m"; // set text and background colors
00437         std::cout «
      "-------------------------------------------------------------------------------\n-";
00438         for (int i = 0; i < spaces_needed; ++i) {
00439             std::cout « " ";
00440         }
00441         std::cout « text;
00442         for (int i = 0; i < spaces_needed; ++i) {
00443             std::cout « " ";
00444         }
00445         std::cout « " -\n";
00446         std::cout «
      "-------------------------------------------------------------------------------\n";
00447         std::cout « "\033[0m"; // reset text and background colors
00448     }
00449
00450     void printSubHeader(const char* text) {
00451         std::cout « "\033[32m"; // set text and background colors
00452         std::cout « text;
00453         std::cout « "\033[0m\n"; // reset text and background colors
00454     }
00455
00456     void printTestCase(const char* text) {
00457         std::cout « "\033[93m  > ["; // set text and background colors
00458         std::cout « text;
00459         std::cout « "]\033[0m \t"; // reset text and background colors
00460     }
00461
```

```
00462
00463 } // namespace util
```

7.14 utilstring.h File Reference

implementation of own string class.

```
#include <cstddef>
#include <iostream>
#include <string>
```

Classes

- class util::string

  Implementation of of own string class.

Namespaces

- namespace util

Macros

- #define INITIAL_SIZE 10

Functions

- void util::deepCopy (char *rawCharTarget, const char *rawCharSource, size_t destStartPosition=-1, size_t srcEndPosition=-1)

  Deep copy of primitive C-string into another primitive C-string.
- void util::printHeader (const char *text)

  to print a nicely formatted and colored text header to the terminal
- void util::printSubHeader (const char *text)

  to print a nicely formatted and colored text sub header to the terminal
- void util::printTestCase (const char *text)

  to print a nicely formatted and colored text title header to the terminal

7.14.1 Detailed Description

implementation of own string class.

==========

Author

Nour Ahmed @email nahmed@stud.hs-bremen.de, nour @repo https://github.com/nouremara/cpp_mystring @createdOn 23.11.2022

Version

1.0.0 @description

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

7.14.2 any macros, constants, or global variables you will need to use it.

Definition in file utilstring.h.

7.14.3 Macro Definition Documentation

7.14.3.1 INITIAL_SIZE

```
#define INITIAL_SIZE 10
```

Initially, the class shall provide memory for 10 printable characters Note: value is implementation detail and subject to change

Definition at line 32 of file utilstring.h.

7.15 utilstring.h

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 1 - Own string Class
00004
00021 #ifndef UTILSTRING_H
00022 #define UTILSTRING_H
00023
00024 #include <cstddef>
00025 #include <iostream>
00026 #include <string>
00027
00028 namespace util {
00032 #define INITIAL_SIZE 10
00033
00042     class string {
00043         char* internal_buffer;
00044         size_t buffer_size;
00045         // size_t string_size;
00046
00047     public:
00048         /*=========================================================================*
00049          *                          Constructors                                   *
00050          *=========================================================================*/
00052         string();
00053
00057         string(size_t intialSize);
00058
00062         string(const char* charArray);
00063
00067         string(const std::string& std_string);
00068
00072         string(const string&);
00073
00074         /*=========================================================================*
00075          *                          Destructor                                     *
00076          *=========================================================================*/
00078         ~string(void);
00079
00080         /*=========================================================================*
00081          *                          Methods                                        *
00082          *=========================================================================*/
00091         void intialize_string(size_t length = 0);
00092         //-------------------------------------------------------------
00093
00110         void deepCopy(const char* rawChar, size_t startPosition = 0);
00111         //-------------------------------------------------------------
00112
```

```
00125         string substr(size_t start_position, size_t length);
00126         //-------------------------------------------------------------
00127
00139         char* c_str() const;
00140         //-------------------------------------------------------------
00141
00155         static int compare(const char* s1, const char* s2);
00156         //-------------------------------------------------------------
00157
00163         void clear();
00164         //-------------------------------------------------------------
00165
00166
00170         static size_t rawSize(const char* rawChar);
00171         //-------------------------------------------------------------
00172
00177         size_t length() const;
00178         size_t size() const;
00179             //---------------------------------------------------------
00180
00185         size_t capacity() const;
00186         //-------------------------------------------------------------
00187
00188         /*=========================================================================*
00189          *                          Operators                                      *
00190          *=========================================================================*/
00191
00195         string operator+(const string& rhsString);
00196         string operator+(const std::string& rhsString);
00197         string operator+(const char* strInstance);
00198         //-------------------------------------------------------------
00199
00200         string& operator+=(const string& rhsString);
00201         string& operator+=(const std::string& rhsString);
00202         string& operator+=(const char* strInstance);
00203         //-------------------------------------------------------------
00204
00206         string& operator=(const string& rhsString);
00207         string& operator=(const char* rhsCharArray);
00208         string& operator=(const std::string& rhsString);
00209         //-------------------------------------------------------------
00210
00215         bool operator==(const string& rhsString);
00216         bool operator==(const std::string& rhsString);
00217         bool operator==(const char* charArray);
00218         //-------------------------------------------------------------
00219
00224         bool operator!=(const string& rhsString);
00225         bool operator!=(const std::string& rhsString);
00226         bool operator!=(const char* charArray);
00227         //-------------------------------------------------------------
00228
00230         char& operator[](size_t position);
00231
00232         /*=========================================================================*
00233          *             Non-member function overloads                               *
00234          *=========================================================================*/
00235          // Free operator methods for the cases util::string is on the RHS
00236          // Friendship enables access to private members
00237
00239         friend std::ostream& operator<<(std::ostream& iostream, const util::string& myString);
00240
00241
00249         friend bool operator==(const std::string& lhsString, const util::string& rhsString);
00250         friend bool operator==(const char* lhsCharArray, const util::string& rhsString);
00251
00259         friend bool operator!=(const std::string& lhsString, const util::string& rhsString);
00260         friend bool operator!=(const char* lhsCharArray, const util::string& rhsString);
00261     };
00262
00263
00264     /*=========================================================================*
00265      *             Some Utility functions                                      *
00266      *=========================================================================*/
00267
00288     void deepCopy(char* rawCharTarget,
00289                   const char* rawCharSource,
00290                   size_t destStartPosition = -1,
00291                   size_t srcEndPosition = -1);
00292     //-----------------------------------------------
00293
00295     void printHeader(const char* text);
00296
00298     void printSubHeader(const char* text);
00299
00301     void printTestCase(const char* text);
00302
```

```
00303 } // namespace util
00304
00305 #endif /* UTILSTRING_H */
```