NourUtilList

1.0.0

Generated by Doxygen 1.9.5

Chapter 1

my_cpp_doubly_linked_list

Implementation of Own Doubly Linked List Class

In this repository, I have an implementation of a class named `util::list`. This class behavior is a simplified implementation of the `std::list`. Class `list` represents a container which organizes stored objects with a so-called doubly linked list. A doubly linked list is basically a list of nodes which are connected among each other.

The doubly linked list data structure is implemented as shown in the following figure:

For each object that is to be stored, a new node is internally created by class `list`. Besides the object to store, each node has two pointers; `prev` and `next`. `prev` points to the previous node, `next` points to the next node.

The first node (`head`) and last node (`tail`) are special nodes since in these cases `prev` or `next` do not point to a predecessor or successor. When adding the first object to the list, an initial node is being created which represents the first and last node at the same time.

Another special node is the `beyond_tail` node. It is a a placeholder node which is the successor of the last node stored in the list. The main purpose of this node is to be used by the method `end()` to return an iterator with it. That is to conform with the STL container conventions that never returns an iterator pointing to a valid (last) object. As this element acts as a placeholder/sentinel, any attempt to access its object results in undefined behavior.

Full and detailed examples of uses and tests of the class `util::list` are given in the `main.cpp` file. Each method and operator is very carefully tested (e.g., calling pop_front() on an empty list, ..., 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.

1.1 Design and Implementation Criteria:

The following design and implementation criteria are followed:

- **No C/C++ standard functions or classes are used** to realize `util::list` class. This include, e.g., `size()`, `push_back()` and of course using `std::list` or similar as an internal representation of `util::list`.
  - This means own functions/methods are developed and implemented to do all required operations.
- Class `util::list` is made generic such that any type can be stored within its nodes. • The class has a **default constructor** that initializes it to an empty list.
- For now, **error handling** is implemented in a simplified fashion. The `util::list` class implements an exception-based error handling for various error cases. Example: Calling `pop_front()` on an empty list shall result in an appropriate exception. Note that some errors may not be handled. 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).
- :fire: **Use this class at your own risk** :fire: :).

1.2 General Functionality

- The class `util::list` is implemented inside the two files `utillist.cpp` and `utillist.h`
- Class list is within the namespace `util`.

1.3 Iterator Concept

To allow iterating `util::list`, a simplified version of the iterator concept is implemented. the class `iterator` as a so-called nested class is added to the to the public part of `util::list`. THe following functionality is implemented for this `iterator`:

- `operator==`: Two iterators are equal if they point to the same node
- `operator!=`: Two iterators are equal if they point to different nodes
- `operator++`: Point/Go to the next node element. Both **prefix and postfix** variants are supported.
- `operator*`: Return a reference to the object stored in the current node
- `operator->`: Return a pointer to the object stored in the current node

1.4 Constructors

The following constructors are implemented:

- `list()` : Default constructor with empty initialization. This **default constructor** that initializes it to an empty list.

1.5 Operators

The following operators are implemented:

- `operator <<` Streaming operator `<<` to print `util::list` elements and some information (size) to `std::cout`.

1.6 Methods

The following methods are implemented:

- `T& front()`: Gives access to the first element of the list
- `T& back()`: Gives access to the last element of the list
- `empty()`: Returns true, if the list does not contain elements
- `size()`: Returns the amount of stored objects
- `clear()`: Clears the list
- `void push_back(const T& element)`: Adds element to the end of the list
- `void pop_back()`: Removes the last element from the list
- `void push_front(const T& element)`: Adds element to the front of the list
- `void pop_front`: Removes the first element from the list
- `display()`: print the list elements and size to the standard `outstream`
- `begin()`: Returns an iterator which points to the first element of the list
- `end()`: Returns a special iterator which points to a placeholder node (`beyond_tail`) which is the successor of the last node stored in the list.
- `insert(iterator, const Object& element)`: Adds element one position before the object iterator is currently pointing to. Iterator can be the `end()` iterator. Returns an iterator pointing to the newly added element.
- `erase(iterator)`: Erases the object iterator is currently pointing to. Returns an iterator to the successor of the erased object. iterator must be valid and dereferenceable. For that reason, the `end()` iterator cannot be used as a parameter.

Notes for the above methods and to follow the spirit of how class std::list behavior. :

- `T` represents the actual type, not a node.
- when calling `front()` and `back()`, they do not return nodes.
- `push_front()` or `push_back()` do not taking nodes as arguments. They take the actual data element.

1.7 Extra Functions and Utilities

Some extra (non-member) functions and utilities are implemented (in the header file `my_extras.h`) that help for better functionality and output. These functions are:

- `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

1.8 References

- Standard list library : https://en.cppreference.com/w/cpp/container/list
- 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 list

  Implementation of of own doubly linked list class.

Functions

- void printHeader (const char *text)
- void printSubHeader (const char *text)
- void printTestCase (const char *text)
- template<class U >
  std::ostream & operator<< (std::ostream &os, const util::list< U > &theList)

5.1.1 Detailed Description

namespace util to contain the Class list

5.1.2 Function Documentation

5.1.2.1 operator<<()

```
template<class U >
std::ostream & util::operator<< (
            std::ostream & os,
            const util::list< U > & theList )
```

Definition at line 147 of file utillist.cpp.

5.1.2.2 printHeader()

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

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 37 of file my_extras.h.

5.1.2.3 printSubHeader()

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

Definition at line 54 of file my_extras.h.

5.1.2.4 printTestCase()

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

Definition at line 60 of file my_extras.h.

Chapter 6

Class Documentation

6.1 util::list< T >::iterator Class Reference

`#include <utillist.h>`

Public Member Functions

- iterator (Node *pNode)
- bool operator== (const iterator &it) const
- bool operator!= (const iterator &it) const
- iterator & operator++ ()
- iterator operator++ (int)
- T & operator* () const
- T * operator-> () const

Friends

- class list

6.1.1 Detailed Description

template<class T>
class util::list< T >::iterator

To allow iterating your list, a simplified version of the iterator concept is implemented as a nested class

Definition at line 153 of file utillist.h.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 iterator()

```
template<class T >
util::list< T >::iterator::iterator (
            Node * pNode )  [inline]
```

constructor create the iterator and associate it to a node.

Definition at line 160 of file utillist.h.

6.1.3 Member Function Documentation

6.1.3.1 operator"!=()

```
template<class T >
bool util::list< T >::iterator::operator!= (
            const iterator & it ) const  [inline]
```

Two iterators are not equal if they point to different nodes.

Definition at line 166 of file utillist.h.

6.1.3.2 operator*()

```
template<class T >
T & util::list< T >::iterator::operator* ( ) const  [inline]
```

Return a reference to the object stored in the current node.

Definition at line 183 of file utillist.h.

6.1.3.3 operator++() [1/2]

```
template<class T >
iterator & util::list< T >::iterator::operator++ ( )  [inline]
```

advance the iterator to the next node.

Definition at line 169 of file utillist.h.

6.1.3.4 operator++() [2/2]

```
template<class T >
iterator util::list< T >::iterator::operator++ (
            int  )  [inline]
```

advance the iterator to the next node (Postfix variant).

Definition at line 176 of file utillist.h.

6.1.3.5 operator->()

```
template<class T >
T * util::list< T >::iterator::operator-> ( ) const  [inline]
```

Return a pointer to the object stored in the current node.

Definition at line 186 of file utillist.h.

6.1.3.6 operator==()

```
template<class T >
bool util::list< T >::iterator::operator== (
            const iterator & it ) const  [inline]
```

Two iterators are equal if they point to the same node.

Definition at line 163 of file utillist.h.

6.1.4 Friends And Related Function Documentation

6.1.4.1 list

```
template<class T >
friend class list  [friend]
```

Definition at line 154 of file utillist.h.

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

- utillist.h

6.2 util::list< T > Class Template Reference

Implementation of of own doubly linked list class.

`#include <utillist.h>`

Classes

- class iterator

Public Member Functions

- list ()
- ~list ()
- list (const list< T > &other_doubly_linked_list)=delete
- list & operator= (list const &)=delete
- T & front ()
- T & back ()
- void push_front (const T &element)
- void pop_front () throw (char*)
- void pop_back ()
- void push_back (const T &element)
- void clear ()
- size_t size () const
- bool empty () const
- iterator begin ()
- iterator end ()

  Returns a special iterator which points to a placeholder node which is the successor of the last node stored in the list. Calling method end() of any STL container never returns an iterator pointing to the last object but an iterator which does not point to a valid object. As this element acts as a placeholder any attempt to access its object results in undefined behavior.
- iterator insert (iterator position, const T &element)
- iterator erase (iterator position) throw (char*)

  Remove element at given position.

Friends

- template<class U >
  std::ostream & operator<< (std::ostream &os, const list< U > &theList)

6.2.1 Detailed Description

template<class T>
class util::list< T >

Implementation of of own doubly linked list class.

Template to make the list generic such that any type can be stored. T represents the actual type

This class presents own list class implementation which organizes stored objects with a so-called doubly linked list. A doubly linked list is basically a list of nodes which are connected among each other in both directions.

This util::list is a container that supports adding, insertion, removal, and moving of elements from anywhere in the container. It provides also bidirectional iteration capability. This class behavior will be similar (but simplified) to the std::list.

Definition at line 51 of file utillist.h.

6.2.2 Constructor & Destructor Documentation

6.2.2.1 list() [1/2]

```
template<class T >
util::list< T >::list ( )  [inline]
```

default constructor creates an empty list

Definition at line 86 of file utillist.h.

6.2.2.2 ~list()

```
template<class T >
util::list< T >::~list
```

destructor

Definition at line 30 of file utillist.cpp.

6.2.2.3 list() [2/2]

```
template<class T >
util::list< T >::list (
            const list< T > & other_doubly_linked_list )  [delete]
```

6.2.3 Member Function Documentation

6.2.3.1 back()

```
template<class T >
T & util::list< T >::back ( )  [inline]
```

Gives access to the last element of the list

Definition at line 98 of file utillist.h.

6.2.3.2 begin()

```
template<class T >
iterator util::list< T >::begin ( )  [inline]
```

Returns an iterator which points to the first element of your list.

Definition at line 190 of file utillist.h.

6.2.3.3 clear()

```
template<class T >
void util::list< T >::clear
```

Clears the list

Definition at line 101 of file utillist.cpp.

6.2.3.4 empty()

```
template<class T >
bool util::list< T >::empty ( ) const  [inline]
```

Returns true, if the list does not contain elements

Definition at line 141 of file utillist.h.

6.2.3.5 end()

```
template<class T >
iterator util::list< T >::end ( )  [inline]
```

Returns a special iterator which points to a placeholder node which is the successor of the last node stored in the list. Calling method end() of any STL container never returns an iterator pointing to the last object but an iterator which does not point to a valid object. As this element acts as a placeholder any attempt to access its object results in undefined behavior.

Definition at line 200 of file utillist.h.

6.2.3.6 erase()

```
template<class T >
iterator util::list< T >::erase (
            iterator position ) throw ( char *)   [inline]
```

Remove element at given position.

Parameters

position	Iterator pointing to element to be erased. Iterator must be valid and de-referencible. For that reason, the end() iterator cannot be used as a parameter.

Returns

An iterator pointing to the next element (the successor of the erased object) or end().

@description This function will erase the element at the given position and thus shorten the list by one.

Definition at line 245 of file utillist.h.

6.2.3.7 front()

```
template<class T >
T & util::list< T >::front ( )  [inline]
```

Gives access to the first element of the list

Definition at line 95 of file utillist.h.

6.2.3.8 insert()

```
template<class T >
iterator util::list< T >::insert (
            iterator position,
            const T & element )  [inline]
```

Adds element one position before the object iterator is currently pointing to. iterator may be the end() iterator. Returns an iterator pointing to the newly added element. create a new node with the contents given. Its next is the current position next node, and its previous is the node at current position.

Definition at line 213 of file utillist.h.

6.2.3.9 operator=()

```
template<class T >
list & util::list< T >::operator= (
            list< T > const &  )  [delete]
```

6.2.3.10 pop_back()

```
template<class T >
void util::list< T >::pop_back
```

removes the last element from the list

Definition at line 80 of file utillist.cpp.

6.2.3.11 pop_front()

```
template<class T >
void util::list< T >::pop_front throw ( char *)
```

Definition at line 41 of file utillist.cpp.

6.2.3.12 push_back()

```
template<class T >
void util::list< T >::push_back (
            const T & element )  [inline]
```

Adds element to the end of the list

Definition at line 121 of file utillist.h.

6.2.3.13 push_front()

```
template<class T >
void util::list< T >::push_front (
            const T & element )  [inline]
```

Adds element to the front of the list

Definition at line 101 of file utillist.h.

6.2.3.14 size()

```
template<class T >
size_t util::list< T >::size
```

Returns the amount of stored objects

Definition at line 115 of file utillist.cpp.

6.2.4 Friends And Related Function Documentation

6.2.4.1 operator<<

```
template<class T >
template<class U >
std::ostream & operator<< (
            std::ostream & os,
            const list< U > & theList )  [friend]
```

Definition at line 147 of file utillist.cpp.

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

- utillist.h
- utillist.cpp

6.3 Person Class Reference

Public Member Functions

- Person (std::string name="", int age=-1)
- std::string getName () const
- int getAge () const

Friends

- std::ostream & operator<< (std::ostream &iostream, const Person &person)

6.3.1 Detailed Description

Definition at line 34 of file main.cpp.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Person()

```
Person::Person (
            std::string name = "",
            int age = -1 )  [inline]
```

Definition at line 39 of file main.cpp.

6.3.3 Member Function Documentation

6.3.3.1 getAge()

```
int Person::getAge ( ) const  [inline]
```

Definition at line 41 of file main.cpp.

6.3.3.2 getName()

```
std::string Person::getName ( ) const  [inline]
```

Definition at line 40 of file main.cpp.

6.3.4 Friends And Related Function Documentation

6.3.4.1 operator<<

```
std::ostream & operator<< (
            std::ostream & iostream,
            const Person & person )  [friend]
```

Definition at line 43 of file main.cpp.

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

- main.cpp

Chapter 7

File Documentation

7.1 main.cpp File Reference

: test of own implementation of list class

```
#include <iostream>
#include <string>
#include "my_extras.h"
#include "utillist.cpp"
#include "utillist.h"
```

Classes

- class Person

Functions

- int main ()

7.1.1 Detailed Description

: test of own implementation of list class

==========

Author

: Nour Ahmed @email : nahmed@stud.hs-bremen.de, nour @repo : https://github.com/nouremara/cpp_my_Doubly_Linked_List @repo : @createdOn : 08.12.2022

Version

: 1.0.0 @description :

Defines the entry point for the NourUtilList application In this application the class util::list is used and tested. Each method and operator is tested with all possible usages (e.g., pushing and popping elements etc.) Note: For this task some error handling is implemented. An exception-based error handling for various error cases. Example: Calling pop_front on an empty list shall result

7.1.2 in an appropriate exception.

Definition in file main.cpp.

7.1.3 Function Documentation

7.1.3.1 main()

```
int main ( )
```

Definition at line 48 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 - Doubly Linked List
00004
00026 #include <iostream>
00027 #include <string>
00028
00029 #include "my_extras.h"
00030 #include "utillist.cpp"  // to avoid link errors we may get while creating an object of list class
00031 #include "utillist.h"
00032
00033 // mockup class used in testing
00034 class Person {
00035     std::string m_name;
00036     int m_age;
00037
00038   public:
00039     Person(std::string name = "", int age = -1) : m_name(name), m_age(age){};
00040     std::string getName() const { return m_name; }
00041     int getAge() const { return m_age; }
00042
00043     friend std::ostream& operator«(std::ostream& iostream, const Person& person) {
00044         return (iostream « person.getName() « "\t" « person.getAge() « std::endl);
00045     }
00046 };
00047
00048 int main() {
00049     util::printHeader("NourUtilList (Doubly Linked Lists) Application");
00050
00051     util::printSubHeader("Variable used for testing and their values");
00052     util::list<int> myList;
00053     std::cout « "myList: " « myList « "\n";
00054     std::cout « "--------------------------------------------------------------------------------\n\n";
00055
00056     util::printSubHeader("Test Member Methods");
00057     util::printTestCase("push_front() and push_back()");
00058     myList.push_front(3);
00059     myList.push_back(5);
00060     myList.push_back(12);
00061     myList.push_back(9);
00062     myList.push_back(12);
00063     myList.push_front(6);
00064     myList.push_back(88);
00065     std::cout « "\tmyList: " « myList « "\n";
00066
00067     util::printTestCase("front()");
00068     std::cout « "front(): " « myList.front() « "\n";
00069
00070     myList.front() = 100;
00071     std::cout « "\tset front(): " « myList.front() « "\n";
00072     std::cout « "\tmyList: " « myList « "\n";
00073
```

```
00074     util::printTestCase("back()");
00075     std::cout « "\tback() : " « myList.back() « "\n";
00076
00077     myList.back() = 200;
00078     std::cout « "\tset back(): " « myList.back() « "\n";
00079     std::cout « "\tmyList: " « myList « "\n";
00080     std::cout « "--------------------------------------------------------------------------------\n\n";
00081
00082     util::printTestCase("pop_front()");
00083
00084     myList.pop_front();
00085     std::cout « "\tmyList: " « myList « "\n";
00086
00087     util::printTestCase("pop_back()");
00088     myList.pop_back();
00089     std::cout « "\tmyList: " « myList « "\n";
00090
00091     std::cout « "--------------------------------------------------------------------------------\n\n";
00092
00093     util::printSubHeader("Test Iterators");
00094
00095     // Finally, print your list as shown:
00096     util::list<int>::iterator myItB = myList.begin();
00097     util::list<int>::iterator myItE = myList.end();
00098     while (myItB != myItE) {
00099         std::cout « "\t" « (*myItB) « std::endl;
00100         ++myItB;
00101     }
00102
00103     util::printTestCase("Test erase()");
00104
00105     // set the iterator to the begin of the list again
00106     myItB = myList.begin();
00107     ++myItB;  // advance the iterator one element ahead
00108
00109     myList.erase(myItB);
00110     std::cout « "\tmyList: " « myList « "\n";
00111     std::cout « "--------------------------------------------------------------------------------\n\n";
00112
00113     util::printSubHeader("Test Other Methods");
00114     util::printTestCase("Test size() and empty()");
00115     std::cout « "\tmyList: " « myList « "\n";
00116     std::cout « "\tmyList size now is: " « myList.size() « "\n";
00117     std::cout « "\tis myList empty? " « (myList.empty() ? "true" : "false") « "\n";
00118
00119     util::printTestCase("Test clear()");
00120     myList.clear();
00121     std::cout « "\tmyList: " « myList « "\n";
00122     std::cout « "\tmyList size now is: " « myList.size() « "\n";
00123     std::cout « "\tis myList empty? " « (myList.empty() ? "true" : "false") « "\n";
00124
00125     util::printTestCase("pop_front() on the empty list");
00126     try {
00127         myList.pop_front();
00128     } catch (char* e) {
00129         std::cerr « "Caught exception:\n"
00130                   « e « "\n\n";
00131     }
00132     std::cout « "\tmyList: " « myList « "\n";
00133
00134     std::cout « "--------------------------------------------------------------------------------\n\n";
00135
00136     util::list<Person> l;  // Store a type of your choice
00137
00138     // Call push_back a few times to add elements
00139     l.push_back(Person("AB", 10));
00140     l.push_back(Person("CD", 20));
00141     l.push_back(Person("EF", 30));
00142     l.push_back(Person("GH", 40));
00143
00144     util::printSubHeader("Variable used for testing and their values");
00145     std::cout « "List l: " « l;
00146     std::cout « "--------------------------------------------------------------------------------\n\n";
00147
00148     util::printSubHeader("Test Iterators");
00149     // Finally, print your list as shown:
00150     util::list<Person>::iterator itB = l.begin();
00151     util::list<Person>::iterator itE = l.end();
00152     while (itB != itE) {
00153         std::cout « "\t" « (*itB).getName() « "\t" « itB->getAge() « std::endl;
00154         ++itB;
00155     }
00156     std::cout « "--------------------------------------------------------------------------------\n\n";
00157
00158     util::printSubHeader("Test erase()");
00159     util::printTestCase("Test erase()");
00160
```

```
00161     // set the iterator to the begin of the list again
00162     itB = l.begin();
00163     ++itB;  // advance the iterator one element ahead
00164
00165     l.erase(itB);
00166     std::cout « " List l: " « l « "\n";
00167
00168     util::printTestCase("Test erase() the end() element");
00169     try {
00170         itE = l.end();
00171         l.erase(itE);
00172     } catch (char* e) {
00173         std::cerr « "Caught exception:\n"
00174                   « e « "\n\n";
00175     }
00176
00177     std::cout « " List l: " « l « "\n";
00178
00179     std::cout « "--------------------------------------------------------------------------------\n\n";
00180
00181     util::printSubHeader("Test insert()");
00182     util::printTestCase("Test insert()");
00183
00184     // set the iterator to the begin of the list again
00185     itB = l.begin();
00186     ++itB;  // advance the iterator one element ahead
00187     l.insert(itB, Person("XY", 100));
00188     std::cout « " List l: " « l « "\n";
00189
00190     util::printTestCase("Test insert() at end()");
00191     l.insert(l.end(), Person("ZZ", 200));
00192     std::cout « " List l: " « l « "\n";
00193
00194     return 0;
00195 }
```

7.3 my_extras.h File Reference

some extra functions and utilities.

`#include <iostream>`

Namespaces

- namespace util

Functions

- void util::printHeader (const char *text)
- void util::printSubHeader (const char *text)
- void util::printTestCase (const char *text)

7.3.1 Detailed Description

some extra functions and utilities.

==========

Author

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

Version

1.0.0 @description

7.3.2 This file presents an implementation of some extra functions and utilities.

Definition in file my_extras.h.

7.4 my_extras.h

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003
00017 #ifndef MY_EXTRAS_H
00018 #define MY_EXTRAS_H
00019
00020 #include <iostream>
00021
00022 namespace util {
00023
00024 /*=========================================================================*
00025  *      Some Utility functions for printing nice text output               *
00026  *=========================================================================*/
00027
00037 void printHeader(const char* text) {
00038     size_t spaces_needed = (80 - std::strlen(text)) / 2 - 2;
00039
00040     std::cout << "\033[1;30;106m";  // set text and background colors
00041     std::cout << "--------------------------------------------------------------------------------\n-";
00042     for (int i = 0; i < spaces_needed; ++i) {
00043         std::cout << " ";
00044     }
00045     std::cout << text;
00046     for (int i = 0; i < spaces_needed; ++i) {
00047         std::cout << " ";
00048     }
00049     std::cout << " -\n";
00050     std::cout << "--------------------------------------------------------------------------------\n";
00051     std::cout << "\033[0m\n";  // reset text and background colors
00052 }
00053
00054 void printSubHeader(const char* text) {
00055     std::cout << "\033[32m";  // set text and background colors
00056     std::cout << text;
00057     std::cout << "\033[0m\n";  // reset text and background colors
00058 }
00059
00060 void printTestCase(const char* text) {
00061     std::cout << "\033[93m  > [";  // set text and background colors
00062     std::cout << text;
00063     std::cout << "]\033[0m \n";  // reset text and background colors
00064 }
00065
00066 }  // namespace util
00067
00068 #endif /* MY_EXTRAS_H */
```

7.5 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

7.5.1 Macro Definition Documentation

7.5.1.1 __has_include

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

Definition at line 17 of file CMakeCCompilerId.c.

7.5.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 668 of file CMakeCCompilerId.c.

7.5.1.3 C_DIALECT

```
#define C_DIALECT
```

Definition at line 757 of file CMakeCCompilerId.c.

7.5.1.4 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 412 of file CMakeCCompilerId.c.

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 672 of file CMakeCCompilerId.c.

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 683 of file CMakeCCompilerId.c.

7.5.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 540 of file CMakeCCompilerId.c.

7.5.1.8 STRINGIFY

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

Definition at line 433 of file CMakeCCompilerId.c.

7.5.1.9 STRINGIFY_HELPER

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

Definition at line 432 of file CMakeCCompilerId.c.

7.5.2 Function Documentation

7.5.2.1 main()

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

Definition at line 781 of file CMakeCCompilerId.c.

7.5.3 Variable Documentation

7.5.3.1 info_arch

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

Definition at line 749 of file CMakeCCompilerId.c.

7.5.3.2 info_compiler

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

Definition at line 419 of file CMakeCCompilerId.c.

7.5.3.3 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.

7.5.3.4 info_platform

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

Definition at line 748 of file CMakeCCompilerId.c.

7.6 CMakeCCompilerId.c

Go to the documentation of this file.

```
00001 #ifdef __cplusplus
00002 # error "A C++ compiler has been selected for C."
00003 #endif
00004
00005 #if defined(__18CXX)
00006 # define ID_VOID_MAIN
00007 #endif
00008 #if defined(__CLASSIC_C__)
00009 /* cv-qualifiers did not exist in K&R C */
00010 # define const
00011 # define volatile
00012 #endif
00013
00014 #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
```

7.7 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

7.7.1 Macro Definition Documentation

7.7.1.1 __has_include

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

Definition at line 11 of file CMakeCXXCompilerId.cpp.

7.7.1.2 ARCHITECTURE_ID

```
#define ARCHITECTURE_ID
```

Definition at line 653 of file CMakeCXXCompilerId.cpp.

7.7.1.3 COMPILER_ID

```
#define COMPILER_ID ""
```

Definition at line 397 of file CMakeCXXCompilerId.cpp.

7.7.1.4 CXX_STD

```
#define CXX_STD __cplusplus
```

Definition at line 751 of file CMakeCXXCompilerId.cpp.

7.7.1.5 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.

7.7.1.6 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.

7.7.1.7 PLATFORM_ID

```
#define PLATFORM_ID
```

Definition at line 525 of file CMakeCXXCompilerId.cpp.

7.7.1.8 STRINGIFY

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

Definition at line 418 of file CMakeCXXCompilerId.cpp.

7.7.1.9 STRINGIFY_HELPER

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

Definition at line 417 of file CMakeCXXCompilerId.cpp.

7.7.2 Function Documentation

7.7.2.1 main()

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

Definition at line 772 of file CMakeCXXCompilerId.cpp.

7.7.3 Variable Documentation

7.7.3.1 info_arch

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

Definition at line 734 of file CMakeCXXCompilerId.cpp.

7.7.3.2 info_compiler

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

Definition at line 404 of file CMakeCXXCompilerId.cpp.

7.7.3.3 info_language_dialect_default

```
const char* info_language_dialect_default
```

Initial value:

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

Definition at line 754 of file CMakeCXXCompilerId.cpp.

7.7.3.4 info_platform

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

Definition at line 733 of file CMakeCXXCompilerId.cpp.

7.8 CMakeCXXCompilerId.cpp

Go to the documentation of this file.

```
00001 /* This source file must have a .cpp extension so that all C++ compilers
00002    recognize the extension without flags.  Borland does not know .cxx for
00003    example.  */
00004 #ifndef __cplusplus
00005 # error "A C compiler has been selected for C++."
00006 #endif
00007
00008 #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 }
```

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

7.10 foo.h

Go to the documentation of this file.

```
00001
```

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

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

Functions

- int main ()

7.11.1 Function Documentation

7.11.1.1 main()

```
int main ( )
```

Definition at line 2 of file main.c.

7.12 main.c

Go to the documentation of this file.

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

7.13 README.md File Reference

7.14 utillist.cpp File Reference

```
#include "utillist.h"
```

Namespaces

- namespace util

Functions

- template<class U >
  std::ostream & util::operator<< (std::ostream &os, const util::list< U > &theList)

7.15 utillist.cpp

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 2 – Doubly Linked List
00004
00023 #include "utillist.h"
00024
00025 // namespace util to contain the Class list
00026 namespace util {
00027
00028 // destructor
00029 template <class T>
00030 list<T>::~list() {
00031     Node* tmp = nullptr;
00032     while (head) {
00033         tmp = head;
00034         head = head->next;
00035         delete tmp;
00036     }
00037     head = nullptr;
00038 }
00039
00040 template <class T>
00041 void list<T>::pop_front() throw(char*) {
```

```
00042     // check whether the doubly linked list is empty or not.
00043     // If it is empty, then do nothing.
00044     if (empty()) {
00045         // return end();
00046         throw "Exception from util::list::pop_front(): pop_front can not be called on an empty list";
00047     }
00048
00049     // check whether the doubly linked list is empty or not.
00050     // If it is empty, then do nothing.
00051     // if (head == nullptr) { return;}
00052
00053     // check whether the doubly linked List has just one node.If it does,
00054     // we just delete that node and set the head and tail pointers to NULL.
00055     if (head->next == nullptr) {
00056         delete head;
00057         head = nullptr;
00058         tail = nullptr;
00059         return;
00060     }
00061
00062     // create a temporary node to be the current head (before deletion)
00063     // Create a new node, current, which is the second node in the list.
00064     Node* current = head->next;
00065
00066     // delete the front node
00067     delete head;
00068
00069     // Make head point to the second node, i.e., current.
00070     // point the head pointer to the previous node
00071     head = current;
00072
00073     // adjust the next pointer of the head node
00074     // Set this node's prev to NULL.
00075     head->prev = nullptr;
00076 }
00077
00078 // removes the last element from the list
00079 template <class T>
00080 void list<T>::pop_back() {
00081     Node* current = tail->prev;
00082     if (head == nullptr) {
00083         return;
00084     }
00085
00086     if (head->next == nullptr) {
00087         delete head;
00088         head = nullptr;
00089         tail = nullptr;
00090         return;
00091     }
00092
00093     // Node* current = tail->prev;
00094     delete tail;
00095     tail = current;
00096     tail->next = nullptr;
00097 }
00098
00099 // Clears the list
00100 template <class T>
00101 void list<T>::clear() {
00102     Node* tmp = tail;
00103
00104     while (tail->prev != nullptr) {
00105         tmp = tail->prev;
00106         delete tail;
00107         tail = tmp;
00108     }
00109     head = nullptr;
00110     tail = nullptr;
00111 }
00112
00113 // Returns the amount of stored objects
00114 template <class T>
00115 size_t list<T>::size() const {
00116     size_t count = 0;
00117     if (head == nullptr) {
00118         return count;
00119     }
00120
00121     Node* current = head;
00122     while (current != nullptr) {
00123         count++;
00124         current = current->next;
00125     }
00126
00127     return count;
00128 }
```

```
00129
00130 template <class T>
00131 void list<T>::display(std::ostream& out) const {
00132     out « " [size: " « size() « "], contents: ";
00133
00134     if (head == nullptr) {
00135         out « " <empty list> ";
00136         return;
00137     }
00138
00139     Node* node = head;
00140     while (node != nullptr) {
00141         out « node->data « " ";
00142         node = node->next;
00143     }
00144 }
00145
00146 template <class U>
00147 std::ostream& operator«(std::ostream& os, const util::list<U>& theList) {
00148     theList.display(os);
00149     return os;
00150 }
00151
00152 } /* namespace util */
```

7.16 utillist.h File Reference

implementation of own doubly linked list class.

`#include <iostream>`

Classes

- class util::list< T >

  Implementation of of own doubly linked list class.
- class util::list< T >::iterator

Namespaces

- namespace util

7.16.1 Detailed Description

implementation of own doubly linked list class.

==========

Author

Nour Ahmed @email nahmed@stud.hs-bremen.de, nourbrm02@gmail.com @repo https://github.com/nouremara/my_cpp_doubly_linked_list @createdOn 08.12.2022

Version

1.0.0 @description implementation of own Doubly Linked List class

This file presents an implementation of a class named list. Class list represents a container which organizes stored objects with a so-called doubly linked list. A doubly linked list is basically a list of nodes which are connected among each other This class behavior will be similar to the std::list (for the implemented

7.16.2 part of its functionality).

==========

Author

Nour Ahmed @email nahmed@stud.hs-bremen.de, nourbrm02@gmail.com @repo https://github.com/nouremara/my_cpp_doubly_linked_list @createdOn 08.12.2022

Version

1.0.0 @description implementation of own Doubly Linked List class

This file presents an implementation of a class named list. Class list represents a container which organizes stored objects with a so-called doubly linked list. A doubly linked list is basically a list of nodes which are connected among each other This class behavior will be similar to the std::list (for the implemented

7.16.3 part of its functionality).

Definition in file utillist.h.

7.17 utillist.h

Go to the documentation of this file.

```
00001 // Nour Ahmed
00002 // Matrikal-Nr.: 5200991
00003 // Assignment 2 - Doubly Linked List
00004
00023 #ifndef UTILLIST_CPP
00024 #define UTILLIST_CPP
00025
00026 #include <iostream>
00027
00031 namespace util {
00032
00037 template <class T>
00051 class list {
00058     struct Node {
00059         T data;
00060         Node* prev;
00061         Node* next;
00062
00064         Node() : next(nullptr), prev(nullptr) {}
00065
00070         Node(const T& element, Node* next_node_ptr = nullptr, Node* prev_node_ptr = nullptr)
00071             : next(next_node_ptr),
00072               prev(prev_node_ptr),
00073               data(element) {}
00074     };
00075
00076     Node *head, *tail;
00077
00082     Node* beyond_tail;
00083
00084   public:
00086     list() : head(nullptr), tail(nullptr), beyond_tail(nullptr) {}
00087
00089     ~list();
00090
00091     list(const list<T>& other_doubly_linked_list) = delete;
00092     list& operator=(list const&) = delete;
00093
00095     T& front() { return head->data; }
00096
00098     T& back() { return tail->data; }
00099
```

```
00101     void push_front(const T& element) {
00102         Node* node = new Node(element);
00103 
00104         if (head == nullptr) {
00105             head = node;
00106             tail = head;
00107             return;
00108         }
00109 
00110         head->prev = node;
00111         node->next = head;
00112         head = node;
00113     }
00114 
00115     void pop_front() throw(char*);
00116 
00118     void pop_back();
00119 
00121     void push_back(const T& element) {
00122         Node* node = new Node(element);
00123         if (tail == nullptr) {
00124             tail = node;
00125             head = tail;
00126             return;
00127         }
00128 
00129         tail->next = node;
00130         node->prev = tail;
00131         tail = node;
00132     }
00133 
00135     void clear();
00136 
00138     size_t size() const;
00139 
00141     bool empty() const { return size() ? false : true; }
00142 
00143     template <class U>
00144     friend std::ostream& operator<<(std::ostream& os, const list<U>& theList);
00145 
00146    private:
00147     void display(std::ostream& out = std::cout) const;
00148 
00149    public:
00153     class iterator {
00154         friend class list;
00155 
00156         Node* m_pNode;
00157 
00158        public:
00160         iterator(Node* pNode) : m_pNode(pNode) {}
00161 
00163         inline bool operator==(const iterator& it) const { return this->m_pNode == it.m_pNode; }
00164 
00166         inline bool operator!=(const iterator& it) const { return this->m_pNode != it.m_pNode; }
00167 
00169         iterator& operator++() {
00170             // Point/Go to the next value of m_pNode
00171             m_pNode = static_cast<Node*>(m_pNode)->next;
00172             return *this;
00173         }
00174 
00176         iterator operator++(int) {
00177             iterator tmp(*this);
00178             ++*this;
00179             return tmp;
00180         }
00181 
00183         T& operator*() const { return static_cast<Node*>(m_pNode)->data; }
00184 
00186         T* operator->() const { return &(static_cast<Node*>(m_pNode)->data); }
00187     };
00188 
00190     iterator begin() { return iterator(head); }
00191 
00200     iterator end() {
00201         // create a circular link between the the tail and beyond_tail nodes
00202         //  tail <=> beyond_tail
00203         tail->next = beyond_tail;
00204 
00205         return iterator(beyond_tail);
00206     }
00207 
00213     iterator insert(iterator position, const T& element) {
00214         // check if it is the end iterator
00215         if (position == end()) {
00216             push_back(element);
```

```
00217             return iterator(tail);
00218         }
00219
00225         Node* newNode = new Node(element, position.m_pNode->next, position.m_pNode);
00226         if (position.m_pNode == tail) tail = newNode;
00227         position.m_pNode->next = newNode;
00228         position.m_pNode->next->prev = newNode;
00229         return iterator(newNode);
00230     }
00231
00245     iterator erase(iterator position) throw(char*) {
00246         // the end() iterator cannot be used as a parameter
00247         if (position == end()) {
00248             // return end();
00249             throw "Exception from util::list::erase(): iterator must be valid and dereferenceable.
       end() iterator cannot be used as a parameter";
00250         }
00251
00252         if (position.m_pNode != head && position.m_pNode != tail->next) {
00253             Node* successor = position.m_pNode->next;
00254             position.m_pNode->prev->next = successor;
00255             position.m_pNode->next->prev = position.m_pNode->prev;
00256             delete position.m_pNode;
00257             return (successor == tail) ? end() : iterator(successor);
00258         }
00259
00260         if (position.m_pNode == head) {
00261             this->pop_front();
00262             return iterator(head);
00263         }
00264
00265         this->pop_back();
00266         return end();
00267     }
00268
00269 };  // end class list
00270
00271 }  // end namespace util
00272
00273 #endif /* UTILLIST_CPP */
```