Lab

Generated by Doxygen 1.9.3

Chapter 1

Porownanie wydajnosci kontenerow standardowych:

Klasa-wrapper ma zaimplementowac metody, ktore zawiera klasa abstrakcyjna IContainerWrapper. Implementujac kazda z metod nalezy zmieniac odpowiadajace im stale preprocesora z 0 na 1. Dzieki temu kolejne testy beda przechodzic.

Zaleznie od uzywanego pod spodem kontenera bedziemy mieli dostepne rozne metody, czesc metod do zaimplementowania mozna w sposob przezroczysty przekazac do wykonania uzywanemu kontenerowi, ktory dana metode juz ma zaimplementowana. Jednakze czesc metod nie jest zaimplementowana, do tego trzeba juz uzyc internetu.

Odsylam do `dokumentacji standardowych kontenerow`, oraz do `artykulu robiacego benchmark`.

1.0.1 Do rozwazenia dla zaawansowanych:

1. Gdzie by sie przydal std::optional?
2. Co by nam dala pula pamieci?
3. Zachecam do poeksperymentowania z typem `value_type` i z rozmiarem elementow - czy cos to zmienia?
4. Dlaczego `pop_front` z biblioteki standardowej zwraca `void`?

1.0.2 Uwaga:

1. Zachecam do uzywania czego sie da z biblioteki standardowej.
2. Metody większe niż 1-linijkowe powinny być zadeklarowane w klasie, zdefiniowane poza klasą w pliku zrodlowym.
3. Obiekty typów klasowych powinny być w miarę możliwości przekazywane w argumentach funkcji przez referencję do stalej,
4. Proszę stosować słówko "const" w odpowiednich miejscach.
5. W pliku naglowkowym prosze nie wlaczac dodatkowych naglowkow typu: <iostream>, <algorithm> - takie rzeczy powinny byc w pliku zrodlowym
6. Prosze aby w pliku naglowkowym nie bylo `using namespace std;`, w zrodlowym moze.
7. Testy nie testuja przypadkow wyjatkowych, ale jak ktos umie warto zabezpieczyc metody na dziwne uzycie.
8. W kodzie sa fragmenty -niespodzianki. Powiem o tym na zajeciach (a w razie czego prosze o przypomnienie)

Mozna tworzyc dowolna ilosc metod pomocniczych, jednakze aby byly one prywatne.

1.0.3 Punktacja:

Liczy sie przejście testów, aczkolwiek dobrze jakby tez nie bylo warningow i wyciekow pamieci

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

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

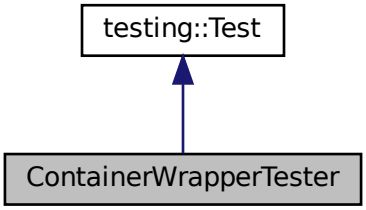
Here is a list of all files with brief descriptions:

Chapter 5

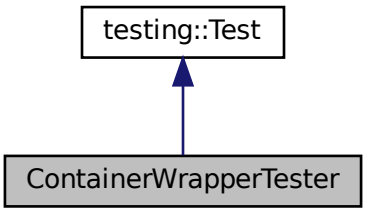
Class Documentation

5.1 ContainerWrapperTester Struct Reference

Inheritance diagram for ContainerWrapperTester:



Collaboration diagram for ContainerWrapperTester:



Public Types

- using value_type = IContainerWrapper::value_type

Static Public Member Functions

- static auto prepareSourceContainer ()

5.1.1 Detailed Description

Definition at line 21 of file containerWrapperTests.cpp.

5.1.2 Member Typedef Documentation

5.1.2.1 value_type

```
using ContainerWrapperTester::value_type = IContainerWrapper::value_type
```

Definition at line 23 of file containerWrapperTests.cpp.

5.1.3 Member Function Documentation

5.1.3.1 prepareSourceContainer()

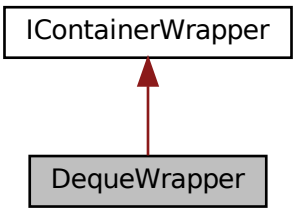
```
static auto ContainerWrapperTester::prepareSourceContainer ( )  [inline], [static]
```

Definition at line 25 of file containerWrapperTests.cpp.
The documentation for this struct was generated from the following file:

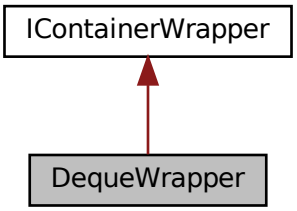
- containerWrapperTests.cpp

5.2 DequeWrapper Class Reference

Wrapper do std::deque.
`#include <containerWrapper.h>`
Inheritance diagram for DequeWrapper:



Collaboration diagram for DequeWrapper:



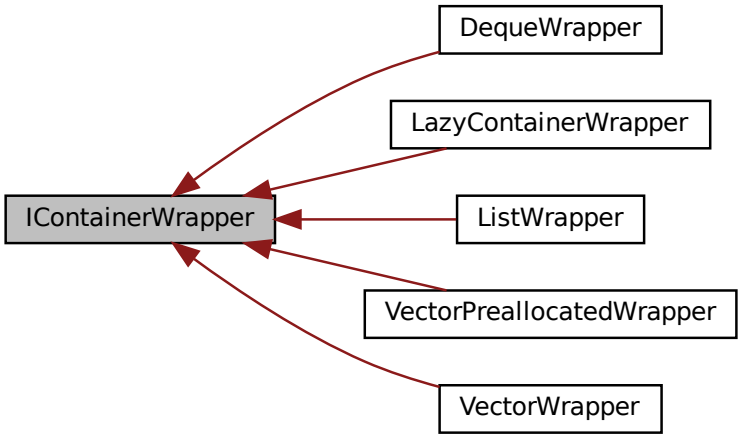
5.2.1 Detailed Description

Wrapper do std::deque.
Definition at line 155 of file containerWrapper.h.
The documentation for this class was generated from the following file:

- containerWrapper.h

5.3 IContainerWrapper Class Reference

Klasa abstrakcyjna - ma nam przypominac co zaimplementowac.
`#include <containerWrapper.h>`
Inheritance diagram for IContainerWrapper:



Public Types

- using value_type = std::int64_t

Public Member Functions

- virtual ~IContainerWrapper ()=0
- virtual void push_back (const value_type &element)=0
- virtual void push_front (const value_type &element)=0
- virtual void insert (const value_type &element, size_t position)=0
- virtual size_t size () const =0
- virtual value_type & at (size_t position)=0
- virtual void sort ()=0
- virtual void erase (size_t position)=0
- virtual value_type count () const =0
- virtual size_t find (const value_type &needle) const =0
- virtual value_type pop_front ()=0

5.3.1 Detailed Description

Klasa abstrakcyjna - ma nam przypominac co zaimplementowac.
Definition at line 66 of file containerWrapper.h.

5.3.2 Member Typedef Documentation

5.3.2.1 value_type

```
using IContainerWrapper::value_type = std::int64_t
```

Definition at line 69 of file containerWrapper.h.

5.3.3 Constructor & Destructor Documentation

5.3.3.1 ~IContainerWrapper()

```
IContainerWrapper::~IContainerWrapper ( )  [pure virtual], [default]
```

5.3.4 Member Function Documentation

5.3.4.1 at()

```
virtual value_type & IContainerWrapper::at (
            size_t position )  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.2 count()

```
virtual value_type IContainerWrapper::count ( ) const  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.3 erase()

```
virtual void IContainerWrapper::erase (
            size_t position )  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.4 find()

```
virtual size_t IContainerWrapper::find (
            const value_type & needle ) const  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.5 insert()

```
virtual void IContainerWrapper::insert (
            const value_type & element,
            size_t position )  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.6 pop_front()

```
virtual value_type IContainerWrapper::pop_front ( )  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.7 push_back()

```
virtual void IContainerWrapper::push_back (
            const value_type & element )  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.8 push_front()

```
virtual void IContainerWrapper::push_front (
            const value_type & element )  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.9 size()

```
virtual size_t IContainerWrapper::size ( ) const  [pure virtual]
```

Implemented in LazyContainerWrapper.

5.3.4.10 sort()

```
virtual void IContainerWrapper::sort ( )  [pure virtual]
```

Implemented in LazyContainerWrapper.

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

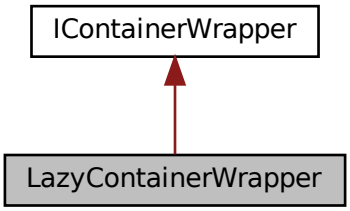
- containerWrapper.h
- containerWrapper.cpp

5.4 LazyContainerWrapper Class Reference

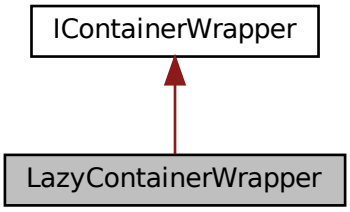
Klasa ktora nic nie robi - aby sie kompilowalo.

`#include <containerWrapper.h>`

Inheritance diagram for LazyContainerWrapper:



Collaboration diagram for LazyContainerWrapper:



Public Member Functions

- LazyContainerWrapper ()=default
- LazyContainerWrapper (const value_type[ ], size_t)
- void push_back (const value_type &) override
- void push_front (const value_type &) override
- void insert (const value_type &, size_t) override
- size_t size () const override
- value_type & at (size_t) override
- void sort () override
- void erase (size_t) override
- value_type count () const override
- size_t find (const value_type &) const override
- value_type pop_front () override

5.4.1 Detailed Description

Klasa ktora nic nie robi - aby sie kompilowalo.
Definition at line 94 of file containerWrapper.h.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 LazyContainerWrapper() [1/2]

```
LazyContainerWrapper::LazyContainerWrapper ( )  [default]
```

5.4.2.2 LazyContainerWrapper() [2/2]

```
LazyContainerWrapper::LazyContainerWrapper (
            const value_type[],
            size_t  )  [inline]
```

Definition at line 98 of file containerWrapper.h.

5.4.3 Member Function Documentation

5.4.3.1 at()

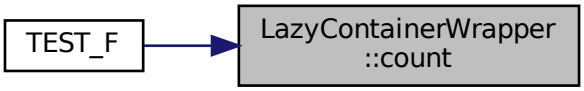
```
value_type & LazyContainerWrapper::at (
            size_t  )  [inline], [override], [virtual]
```

Implements IContainerWrapper.
Definition at line 108 of file containerWrapper.h.
Here is the caller graph for this function:

5.4.3.2 count()

```
value_type LazyContainerWrapper::count ( ) const  [inline], [override], [virtual]
```

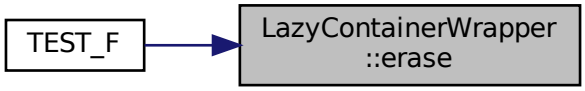
Implements IContainerWrapper.
Definition at line 118 of file containerWrapper.h.
Here is the caller graph for this function:



5.4.3.3 erase()

```
void LazyContainerWrapper::erase (
            size_t  )  [inline], [override], [virtual]
```

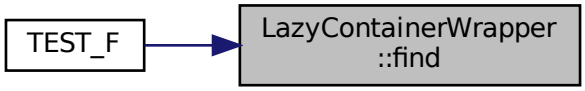
Implements IContainerWrapper.
Definition at line 116 of file containerWrapper.h.
Here is the caller graph for this function:



5.4.3.4 find()

```
size_t LazyContainerWrapper::find (
            const value_type &  ) const  [inline], [override], [virtual]
```

Implements IContainerWrapper.
Definition at line 120 of file containerWrapper.h.
Here is the caller graph for this function:

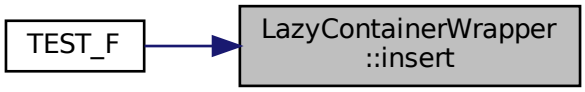


5.4.3.5 insert()

```
void LazyContainerWrapper::insert (
            const value_type & ,
            size_t  )  [inline], [override], [virtual]
```

Implements IContainerWrapper.
Definition at line 104 of file containerWrapper.h.

Here is the caller graph for this function:



5.4.3.6 pop_front()

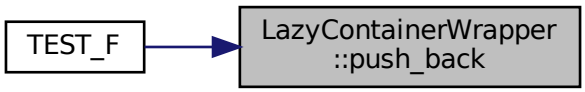
```
value_type LazyContainerWrapper::pop_front ( )  [inline], [override], [virtual]
```

Implements IContainerWrapper.
Definition at line 122 of file containerWrapper.h.

5.4.3.7 push_back()

```
void LazyContainerWrapper::push_back (
            const value_type &  )  [inline], [override], [virtual]
```

Implements IContainerWrapper.
Definition at line 100 of file containerWrapper.h.
Here is the caller graph for this function:

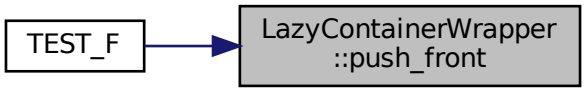


5.4.3.8 push_front()

```
void LazyContainerWrapper::push_front (
            const value_type &  )  [inline], [override], [virtual]
```

Implements IContainerWrapper.
Definition at line 102 of file containerWrapper.h.

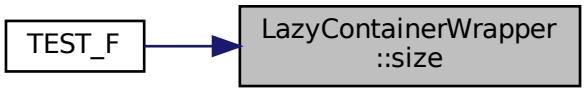
Here is the caller graph for this function:



5.4.3.9 size()

```
size_t LazyContainerWrapper::size ( ) const  [inline], [override], [virtual]
```

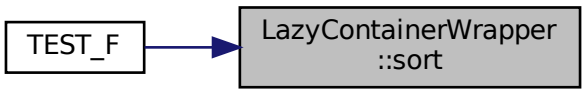
Implements IContainerWrapper.
Definition at line 106 of file containerWrapper.h.
Here is the caller graph for this function:



5.4.3.10 sort()

```
void LazyContainerWrapper::sort ( )  [inline], [override], [virtual]
```

Implements IContainerWrapper.
Definition at line 114 of file containerWrapper.h.
Here is the caller graph for this function:

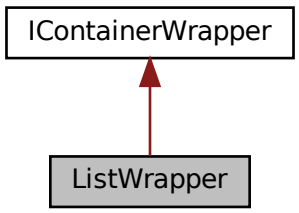


The documentation for this class was generated from the following file:

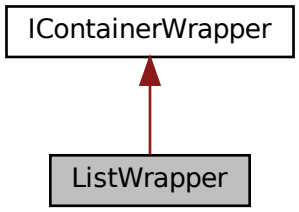
- containerWrapper.h

5.5 ListWrapper Class Reference

Wrapper do std::list.
`#include <containerWrapper.h>`
Inheritance diagram for ListWrapper:



Collaboration diagram for ListWrapper:



5.5.1 Detailed Description

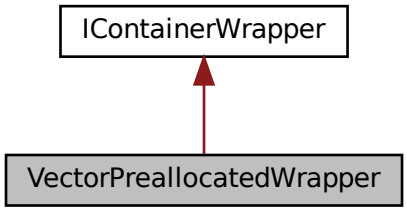
Wrapper do std::list.
Definition at line 146 of file containerWrapper.h.
The documentation for this class was generated from the following file:

- containerWrapper.h

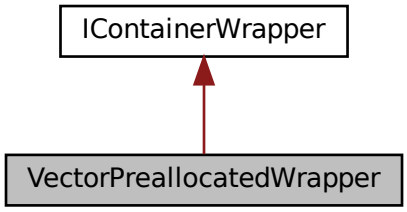
5.6 VectorPreallocatedWrapper Class Reference

Wrapper do std::vector, ktory dokonuje pre-allokacji w konstruktorze.
`#include <containerWrapper.h>`

Inheritance diagram for VectorPreallocatedWrapper:



Collaboration diagram for VectorPreallocatedWrapper:



5.6.1 Detailed Description

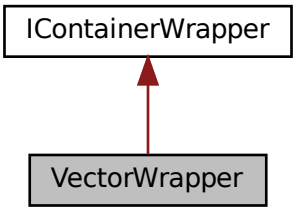
Wrapper do std::vector, ktory dokonuje pre-allokacji w konstruktorze.
Definition at line 135 of file containerWrapper.h.
The documentation for this class was generated from the following file:

- containerWrapper.h

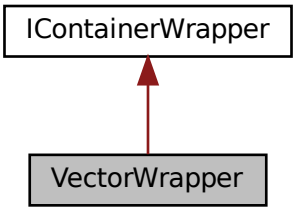
5.7 VectorWrapper Class Reference

Wrapper do std::vector.
`#include <containerWrapper.h>`

Inheritance diagram for VectorWrapper:



Collaboration diagram for VectorWrapper:



5.7.1 Detailed Description

Wrapper do std::vector.
Definition at line 126 of file containerWrapper.h.
The documentation for this class was generated from the following file:

- containerWrapper.h

Chapter 6

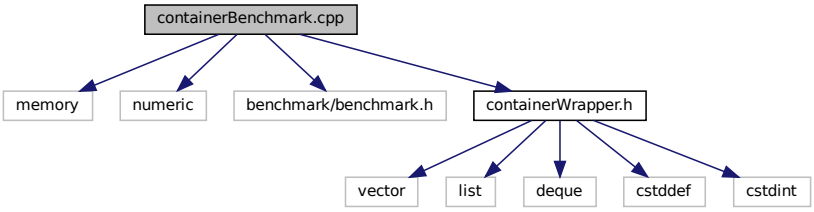
File Documentation

6.1 CMakeLists.txt File Reference

6.2 containerBenchmark.cpp File Reference

```
#include <memory>
#include <numeric>
#include <benchmark/benchmark.h>
#include "containerWrapper.h"
```

Include dependency graph for containerBenchmark.cpp:



Functions

- BENCHMARK (BM_constructionLotsOfElementsWithDefaultConstructor)
- BENCHMARK (BM_constructionFromArray)
- BENCHMARK (BM_pushBackManyElements)
- BENCHMARK (BM_pushFrontManyElements)
- BENCHMARK (BM_insertingInRandomPositionManyElements)
- BENCHMARK (BM_randomAccess)
- BENCHMARK (BM_sorting)
- BENCHMARK (BM_randomErase)
- BENCHMARK (BM_count_expectedAllElementsCounted)
- BENCHMARK (BM_findElement)

6.2.1 Function Documentation

6.2.1.1 BENCHMARK() [1/10]

```
BENCHMARK (
            BM_constructionFromArray  )
```

6.2.1.2 BENCHMARK() [2/10]

```
BENCHMARK (
            BM_constructionLotsOfElementsWithDefaultConstructor  )
```

6.2.1.3 BENCHMARK() [3/10]

```
BENCHMARK (
            BM_count_expectedAllElementsCounted  )
```

6.2.1.4 BENCHMARK() [4/10]

```
BENCHMARK (
            BM_findElement  )
```

6.2.1.5 BENCHMARK() [5/10]

```
BENCHMARK (
            BM_insertingInRandomPositionManyElements  )
```

6.2.1.6 BENCHMARK() [6/10]

```
BENCHMARK (
            BM_pushBackManyElements  )
```

6.2.1.7 BENCHMARK() [7/10]

```
BENCHMARK (
            BM_pushFrontManyElements  )
```

6.2.1.8 BENCHMARK() [8/10]

```
BENCHMARK (
            BM_randomAccess  )
```

6.2.1.9 BENCHMARK() [9/10]

```
BENCHMARK (
            BM_randomErase  )
```

6.2.1.10 BENCHMARK() [10/10]

```
BENCHMARK (
            BM_sorting  )
```

6.3 containerBenchmark.cpp

Go to the documentation of this file.

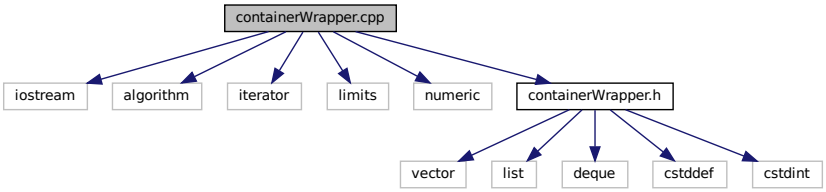
```
00001 #include <memory>
00002 #include <numeric> // std::iota()
00003 #include <benchmark/benchmark.h>
00004 #include "containerWrapper.h"
00005
00007
00008 namespace
00009 {
00010     constexpr size_t N = 100'000;
00011
00012     using value_type = IContainerWrapper::value_type;
00013
00014     static auto prepareSourceContainer()
00015     {
00016         std::array<value_type, N> sourceElements;
00017
00018         std::iota(sourceElements.begin(), sourceElements.end(), 0);
00019         return sourceElements;
00020     }
00021 } // namespace
00022
00023
00024 static void BM_constructionLotsOfElementsWithDefaultConstructor(benchmark::State& state)
00025 {
00026     for (auto _ : state)
00027     {
00028         std::unique_ptr<ContainerWrapper[]> wrappers(new ContainerWrapper[1000]);
00029         static_cast<void>(wrappers); // to disable warning
00030     }
00031 }
00032 BENCHMARK(BM_constructionLotsOfElementsWithDefaultConstructor);
00033
00034
00035 static void BM_constructionFromArray(benchmark::State& state)
00036 {
00037     const auto sourceElements = prepareSourceContainer();
00038
00039     for (auto _ : state)
00040     {
00041         ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00042         static_cast<void>(wrapper); // to disable warning
00043     }
00044 }
00045 BENCHMARK(BM_constructionFromArray);
00046
00047
00048 static void BM_pushBackManyElements(benchmark::State& state)
00049 {
00050     const auto sourceElements = prepareSourceContainer();
00051
00052     for (auto _ : state)
00053     {
00054         ContainerWrapper wrapper;
00055         for (const auto& element : sourceElements)
00056         {
00057             wrapper.push_back(element);
00058         }
00059     }
00060 }
00061 BENCHMARK(BM_pushBackManyElements);
00062
00063
00064 static void BM_pushFrontManyElements(benchmark::State& state)
00065 {
00066     const auto sourceElements = prepareSourceContainer();
00067
00068     for (auto _ : state)
00069     {
00070         ContainerWrapper wrapper;
00071         for (const auto& element : sourceElements)
00072         {
00073             wrapper.push_front(element);
00074         }
00075     }
00076 }
00077 BENCHMARK(BM_pushFrontManyElements);
00078
00079
00080 static void BM_insertingInRandomPositionManyElements(benchmark::State& state)
00081 {
00082     const auto sourceElements = prepareSourceContainer();
00083
00084     for (auto _ : state)
```

```
00085     {
00086         ContainerWrapper wrapper;
00087         for (const auto& element : sourceElements)
00088         {
00089             const size_t position = wrapper.size() > 0 ? rand() % wrapper.size() : 0;
00090             wrapper.insert(element, position);
00091         }
00092     }
00093 }
00094 BENCHMARK(BM_insertingInRandomPositionManyElements);
00095
00096 static void BM_randomAccess(benchmark::State& state)
00097 {
00098     const auto sourceElements = prepareSourceContainer();
00099
00100     ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00101
00102     for (auto _ : state)
00103     {
00104         const size_t position = rand() % wrapper.size();
00105         wrapper.at(position);
00106     }
00107 }
00108 BENCHMARK(BM_randomAccess);
00109
00110 static void BM_sorting(benchmark::State& state)
00111 {
00112     auto sourceElements = prepareSourceContainer();
00113     decltype(sourceElements) reversedElements;
00114     std::reverse_copy(begin(sourceElements), end(sourceElements), begin(reversedElements));
00115
00116     for (auto _ : state)
00117     {
00118         ContainerWrapper wrapper(reversedElements.data(), sourceElements.size());
00119         wrapper.sort();
00120     }
00121 }
00122 BENCHMARK(BM_sorting);
00123
00124 static void BM_randomErase(benchmark::State& state)
00125 {
00126     const auto sourceElements = prepareSourceContainer();
00127
00128     for (auto _ : state)
00129     {
00130         ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00131
00132         for (size_t i=1; i < N; ++i)
00133         {
00134             const size_t erasePosition = wrapper.size() > 0 ? rand() % wrapper.size() : 0;
00135             wrapper.erase(erasePosition);
00136         }
00137     }
00138 }
00139 BENCHMARK(BM_randomErase);
00140
00141 static void BM_count_expectedAllElementsCounted(benchmark::State& state)
00142 {
00143     const auto sourceElements = prepareSourceContainer();
00144
00145     const ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00146
00147     for (auto _ : state)
00148     {
00149         wrapper.count();
00150     }
00151 }
00152 BENCHMARK(BM_count_expectedAllElementsCounted);
00153
00154 static void BM_findElement(benchmark::State& state)
00155 {
00156     const auto sourceElements = prepareSourceContainer();
00157
00158     const ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00159
00160     for (auto _ : state)
00161     {
00162         for (size_t i=0; i < N / 10; ++i)
00163         {
00164             const size_t expectedPosition = rand() % sourceElements.size();
00165             const auto element2Find = sourceElements[expectedPosition];
00166             wrapper.find(element2Find);
00167         }
00168     }
00169 }
00170 BENCHMARK(BM_findElement);
```

6.4 containerWrapper.cpp File Reference

```
#include <iostream>
#include <algorithm>
#include <iterator>
#include <limits>
#include <numeric>
#include "containerWrapper.h"
```

Include dependency graph for containerWrapper.cpp:



6.5 containerWrapper.cpp

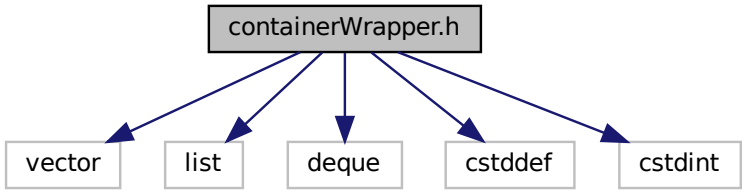
Go to the documentation of this file.

```
00001 #include <iostream>
00002 #include <algorithm> // std::sort, std::find, std::copy
00003 #include <iterator> // std::distance, std::advance, std::back_inserter
00004 #include <limits>   // std::numeric_limits<size_t>::max()
00005 #include <numeric>  // std::accumulate()
00006
00007 using namespace std;
00008
00009 #include "containerWrapper.h"
00010
00011 #ifndef _MSC_FULL_VER // if not Visual Studio Compiler
00012     #warning "Klasa jest do zaimplementowania. Instrukcja w pliku naglowkowym"
00013 #else
00014     #pragma message ("Klasa jest do zaimplementowania. Instrukcja w pliku naglowkowym")
00015 #endif
00016
00017 IContainerWrapper::~IContainerWrapper() = default;
00018
00019
```

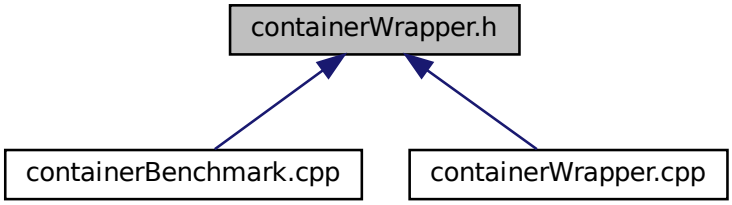
6.6 containerWrapper.h File Reference

```
#include <vector>
#include <list>
#include <deque>
#include <cstddef>
#include <cstdint>
```

Include dependency graph for containerWrapper.h:



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



Classes

- class IContainerWrapper

  Klasa abstrakcyjna - ma nam przypominac co zaimplementowac.
- class LazyContainerWrapper

  Klasa ktora nic nie robi - aby sie kompilowalo.
- class VectorWrapper

  Wrapper do std::vector.
- class VectorPreallocatedWrapper

  Wrapper do std::vector, ktory dokonuje pre-allokacji w konstruktorze.
- class ListWrapper

  Wrapper do std::list.
- class DequeWrapper

  Wrapper do std::deque.

Macros

- #define IMPLEMENTED_DEFAULT_CONSTRUCTOR 0
- #define IMPLEMENTED_CONSTRUCTOR_COPYING_FROM_ARRAY 0
- #define IMPLEMENTED_PUSH_BACK 0
- #define IMPLEMENTED_PUSH_FRONT 0
- #define IMPLEMENTED_INSERT 0
- #define IMPLEMENTED_AT 0

- #define IMPLEMENTED_SORT 0
- #define IMPLEMENTED_ERASE 0
- #define IMPLEMENTED_COUNT 0
- #define IMPLEMENTED_FIND 0
- #define IMPLEMENTED_POP_FRONT 0

Typedefs

- using ContainerWrapper = LazyContainerWrapper

  Alias typu, ktory bedzie podlegac testom.

6.6.1 Macro Definition Documentation

6.6.1.1 IMPLEMENTED_AT

```
#define IMPLEMENTED_AT 0
```

Definition at line 57 of file containerWrapper.h.

6.6.1.2 IMPLEMENTED_CONSTRUCTOR_COPYING_FROM_ARRAY

```
#define IMPLEMENTED_CONSTRUCTOR_COPYING_FROM_ARRAY 0
```

Definition at line 53 of file containerWrapper.h.

6.6.1.3 IMPLEMENTED_COUNT

```
#define IMPLEMENTED_COUNT 0
```

Definition at line 60 of file containerWrapper.h.

6.6.1.4 IMPLEMENTED_DEFAULT_CONSTRUCTOR

```
#define IMPLEMENTED_DEFAULT_CONSTRUCTOR 0
```

Definition at line 52 of file containerWrapper.h.

6.6.1.5 IMPLEMENTED_ERASE

```
#define IMPLEMENTED_ERASE 0
```

Definition at line 59 of file containerWrapper.h.

6.6.1.6 IMPLEMENTED_FIND

```
#define IMPLEMENTED_FIND 0
```

Definition at line 61 of file containerWrapper.h.

6.6.1.7 IMPLEMENTED_INSERT

```
#define IMPLEMENTED_INSERT 0
```

Definition at line 56 of file containerWrapper.h.

6.6.1.8 IMPLEMENTED_POP_FRONT

```
#define IMPLEMENTED_POP_FRONT 0
```
Definition at line 62 of file containerWrapper.h.

6.6.1.9 IMPLEMENTED_PUSH_BACK

```
#define IMPLEMENTED_PUSH_BACK 0
```
Definition at line 54 of file containerWrapper.h.

6.6.1.10 IMPLEMENTED_PUSH_FRONT

```
#define IMPLEMENTED_PUSH_FRONT 0
```
Definition at line 55 of file containerWrapper.h.

6.6.1.11 IMPLEMENTED_SORT

```
#define IMPLEMENTED_SORT 0
```
Definition at line 58 of file containerWrapper.h.

6.6.2 Typedef Documentation

6.6.2.1 ContainerWrapper

```
using ContainerWrapper = LazyContainerWrapper
```
Alias typu, ktory bedzie podlegac testom.
Definition at line 164 of file containerWrapper.h.

6.7 containerWrapper.h

Go to the documentation of this file.

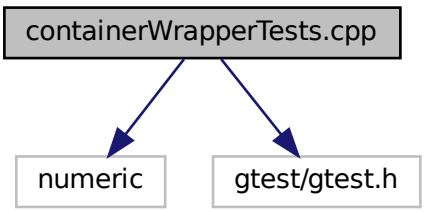
```
00001 #ifndef MATRIX_H
00002 #define MATRIX_H
00003
00046 #include <vector>
00047 #include <list>
00048 #include <deque>
00049 #include <cstddef> // std::size_t
00050 #include <cstdint> // std::int64_t
00051
00052 #define IMPLEMENTED_DEFAULT_CONSTRUCTOR 0
00053 #define IMPLEMENTED_CONSTRUCTOR_COPYING_FROM_ARRAY 0
00054 #define IMPLEMENTED_PUSH_BACK 0
00055 #define IMPLEMENTED_PUSH_FRONT 0
00056 #define IMPLEMENTED_INSERT 0
00057 #define IMPLEMENTED_AT 0
00058 #define IMPLEMENTED_SORT 0
00059 #define IMPLEMENTED_ERASE 0
00060 #define IMPLEMENTED_COUNT 0
00061 #define IMPLEMENTED_FIND 0
00062 #define IMPLEMENTED_POP_FRONT 0
00063
00064
00066 class IContainerWrapper
00067 {
00068 public:
00069     using value_type = std::int64_t;
00070
00071     virtual ~IContainerWrapper() = 0;
00072
00073     virtual void push_back(const value_type& element) = 0;
00074     virtual void push_front(const value_type& element) = 0;
00075
00076     virtual void insert(const value_type& element, size_t position) = 0;
00077
```

```
00078     virtual size_t size() const = 0;
00079
00080     virtual value_type& at(size_t position) = 0;
00081
00082     virtual void sort() = 0;
00083
00084     virtual void erase(size_t position) = 0;
00085
00086     virtual value_type count() const = 0;
00087
00088     virtual size_t find(const value_type& needle) const = 0;
00089
00090     virtual value_type pop_front() = 0;
00091 };
00092
00094 class LazyContainerWrapper: IContainerWrapper
00095 {
00096 public:
00097     LazyContainerWrapper() = default;
00098     LazyContainerWrapper(const value_type /*elements*/[], size_t /*N*/) {}
00099
00100     void push_back(const value_type& /*element*/) override {}
00101
00102     void push_front(const value_type& /*element*/) override {}
00103
00104     void insert(const value_type& /*element*/, size_t /*position*/) override {}
00105
00106     size_t size() const override { return {}; }
00107
00108     value_type& at(size_t /*position*/) override
00109     {
00110         static value_type zero{};
00111         return zero;
00112     }
00113
00114     void sort() override {}
00115
00116     void erase(size_t /*position*/) override {}
00117
00118     value_type count() const override { return {}; }
00119
00120     size_t find(const value_type& /*needle*/) const override { return {}; }
00121
00122     value_type pop_front() override { return {}; }
00123 };
00124
00126 class VectorWrapper: IContainerWrapper
00127 {
00128 public:
00129     // TODO: ...
00130 private:
00131     std::vector<value_type> impl_;
00132 };
00133
00135 class VectorPreallocatedWrapper: IContainerWrapper
00136 {
00137     constexpr static std::size_t preallocationSize = 1'000'000;
00138
00139 public:
00140     // TODO: ...
00141 private:
00142     std::vector<value_type> impl_;
00143 };
00144
00146 class ListWrapper: IContainerWrapper
00147 {
00148 public:
00149     // TODO: ...
00150 private:
00151     std::list<value_type> impl_;
00152 };
00153
00155 class DequeWrapper: IContainerWrapper
00156 {
00157 public:
00158     // TODO: ...
00159 private:
00160     std::deque<value_type> impl_;
00161 };
00162
00164 using ContainerWrapper = LazyContainerWrapper;
00165
00166 #endif // MATRIX_H
```

6.8 containerWrapperTests.cpp File Reference

```
#include <numeric>
#include <gtest/gtest.h>
```

Include dependency graph for containerWrapperTests.cpp:



Classes

- struct ContainerWrapperTester

Functions

- TEST_F (ContainerWrapperTester, constructionLotsOfElementsWithDefaultConstructor)
- TEST_F (ContainerWrapperTester, constructionFromArray)
- TEST_F (ContainerWrapperTester, pushBackManyElements)
- TEST_F (ContainerWrapperTester, pushFrontManyElements)
- TEST_F (ContainerWrapperTester, insertingInRandomPositionManyElements)
- TEST_F (ContainerWrapperTester, randomAccess)
- TEST_F (ContainerWrapperTester, sorting)
- TEST_F (ContainerWrapperTester, randomErase)
- TEST_F (ContainerWrapperTester, count_expectedAllElementsCounted)
- TEST_F (ContainerWrapperTester, findElement)

6.8.1 Function Documentation

6.8.1.1 TEST_F() [1/10]

```
TEST_F (
            ContainerWrapperTester ,
            constructionFromArray  )
```

Definition at line 49 of file containerWrapperTests.cpp.

6.8.1.2 TEST_F() [2/10]

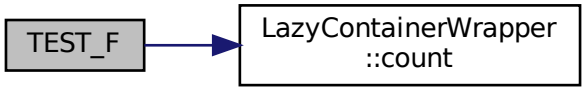
```
TEST_F (
            ContainerWrapperTester ,
            constructionLotsOfElementsWithDefaultConstructor  )
```

Definition at line 34 of file containerWrapperTests.cpp.

6.8.1.3 TEST_F() [3/10]

```
TEST_F (
            ContainerWrapperTester ,
            count_expectedAllElementsCounted  )
```

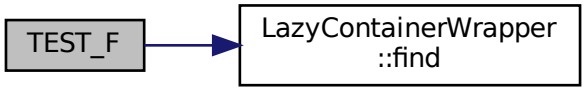
Definition at line 160 of file containerWrapperTests.cpp.
Here is the call graph for this function:



6.8.1.4 TEST_F() [4/10]

```
TEST_F (
            ContainerWrapperTester ,
            findElement  )
```

Definition at line 174 of file containerWrapperTests.cpp.
Here is the call graph for this function:

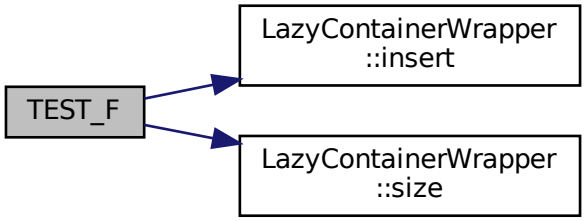


6.8.1.5 TEST_F() [5/10]

```
TEST_F (
            ContainerWrapperTester ,
            insertingInRandomPositionManyElements  )
```

Definition at line 91 of file containerWrapperTests.cpp.

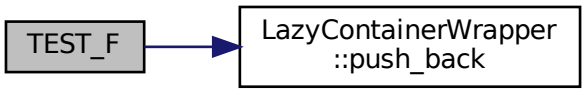
Here is the call graph for this function:



6.8.1.6 TEST_F() [6/10]

```
TEST_F (
            ContainerWrapperTester ,
            pushBackManyElements  )
```

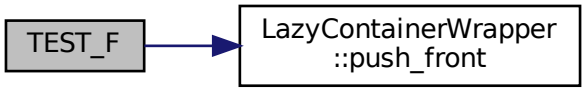
Definition at line 61 of file containerWrapperTests.cpp.
Here is the call graph for this function:



6.8.1.7 TEST_F() [7/10]

```
TEST_F (
            ContainerWrapperTester ,
            pushFrontManyElements  )
```

Definition at line 76 of file containerWrapperTests.cpp.
Here is the call graph for this function:



6.8.1.8 TEST_F() [8/10]

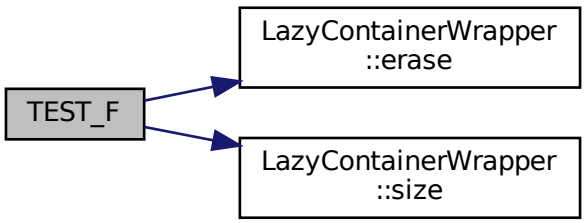
```
TEST_F (
            ContainerWrapperTester ,
            randomAccess  )
```

Definition at line 107 of file containerWrapperTests.cpp.
Here is the call graph for this function:

6.8.1.9 TEST_F() [9/10]

```
TEST_F (
            ContainerWrapperTester ,
            randomErase  )
```

Definition at line 142 of file containerWrapperTests.cpp.
Here is the call graph for this function:

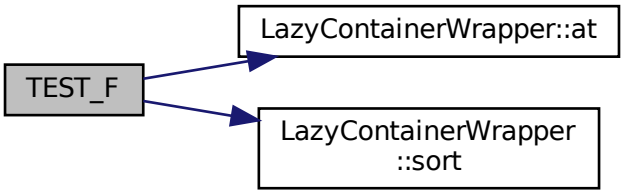


6.8.1.10 TEST_F() [10/10]

```
TEST_F (
            ContainerWrapperTester ,
            sorting  )
```

Definition at line 123 of file containerWrapperTests.cpp.

Here is the call graph for this function:



6.9 containerWrapperTests.cpp

Go to the documentation of this file.

```
00001 #include <numeric> // std::iota()
00002 #include <gtest/gtest.h>
00003
00004 #if __has_include("../containerWrapper.h")
00005     #include "../containerWrapper.h"
00006 #elif __has_include("containerWrapper.h")
00007     #include "containerWrapper.h"
00008 #else
00009     #error "File 'containerWrapper.h' not found!"
00010 #endif
00011
00012 using namespace std;
00013 using namespace ::testing;
00014
00015
00016 namespace
00017 {
00018     constexpr size_t N = 100'000;
00019 } // namespace
00020
00021 struct ContainerWrapperTester : ::testing::Test
00022 {
00023     using value_type = IContainerWrapper::value_type;
00024
00025     static auto prepareSourceContainer()
00026     {
00027         std::array<value_type, N> sourceElements;
00028
00029         std::iota(sourceElements.begin(), sourceElements.end(), 0);
00030         return sourceElements;
00031     }
00032 };
00033
00034 TEST_F(ContainerWrapperTester, constructionLotsOfElementsWithDefaultConstructor)
00035 {
00036 #if IMPLEMENTED_DEFAULT_CONSTRUCTOR == 1
00037     unique_ptr<ContainerWrapper[]> wrappers(new ContainerWrapper[N]);
00038     static_cast<void>(wrappers); // to disable warning
00039
00040     #ifndef DOMJUDGE
00041     clog « "[INFO] Created " « N « " obects, with size: "
00042          « (N*sizeof(value_type)/1024) « " kb" « endl;
00043     #endif // #ifndef DOMJUDGE
00044 #else
00045     ADD_FAILURE() « "Default constructor not implemented!";
00046 #endif
00047 }
00048
00049 TEST_F(ContainerWrapperTester, constructionFromArray)
00050 {
00051 #if IMPLEMENTED_CONSTRUCTOR_COPYING_FROM_ARRAY == 1
00052     const auto sourceElements = prepareSourceContainer();
00053
00054     ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00055     static_cast<void>(wrapper); // to disable warning
00056 #else
00057     ADD_FAILURE() « "Constructor, which is copying from array not implemented!";
```

```
00058 #endif
00059 }
00060
00061 TEST_F(ContainerWrapperTester, pushBackManyElements)
00062 {
00063 #if IMPLEMENTED_PUSH_BACK == 1
00064     const auto sourceElements = prepareSourceContainer();
00065
00066     ContainerWrapper wrapper;
00067     for (const auto& element : sourceElements)
00068     {
00069         wrapper.push_back(element);
00070     }
00071 #else
00072     ADD_FAILURE() << "Method push_back() not implemented!";
00073 #endif
00074 }
00075
00076 TEST_F(ContainerWrapperTester, pushFrontManyElements)
00077 {
00078 #if IMPLEMENTED_PUSH_FRONT == 1
00079     const auto sourceElements = prepareSourceContainer();
00080
00081     ContainerWrapper wrapper;
00082     for (const auto& element : sourceElements)
00083     {
00084         wrapper.push_front(element);
00085     }
00086 #else
00087     ADD_FAILURE() << "Method push_back() not implemented!";
00088 #endif
00089 }
00090
00091 TEST_F(ContainerWrapperTester, insertingInRandomPositionManyElements)
00092 {
00093 #if IMPLEMENTED_INSERT == 1
00094     const auto sourceElements = prepareSourceContainer();
00095
00096     ContainerWrapper wrapper;
00097     for (const auto& element : sourceElements)
00098     {
00099         const size_t position = wrapper.size() > 0 ? rand() % wrapper.size() : 0;
00100         wrapper.insert(element, position);
00101     }
00102 #else
00103     ADD_FAILURE() << "Method insert() not implemented!";
00104 #endif
00105 }
00106
00107 TEST_F(ContainerWrapperTester, randomAccess)
00108 {
00109 #if IMPLEMENTED_AT == 1
00110     const auto sourceElements = prepareSourceContainer();
00111
00112     ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00113
00114     for (size_t i=0; i < N; ++i)
00115     {
00116         ASSERT_EQ(sourceElements[i], wrapper.at(i)) << "i = " << i;
00117     }
00118 #else
00119     ADD_FAILURE() << "Method at() not implemented!";
00120 #endif
00121 }
00122
00123 TEST_F(ContainerWrapperTester, sorting)
00124 {
00125 #if IMPLEMENTED_SORT == 1
00126     auto sourceElements = prepareSourceContainer();
00127     decltype(sourceElements) reversedElements;
00128     std::reverse_copy(begin(sourceElements), end(sourceElements), begin(reversedElements));
00129
00130     ContainerWrapper wrapper(reversedElements.data(), sourceElements.size());
00131     wrapper.sort();
00132
00133     for (size_t i=0; i < N; ++i)
00134     {
00135         ASSERT_EQ(sourceElements[i], wrapper.at(i)) << "i = " << i;
00136     }
00137 #else
00138     ADD_FAILURE() << "Method sort() not implemented!";
00139 #endif
00140 }
00141
00142 TEST_F(ContainerWrapperTester, randomErase)
00143 {
00144 #if IMPLEMENTED_ERASE == 1
```

```
00145     const auto sourceElements = prepareSourceContainer();
00146
00147     ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00148
00149     for (size_t i=1; i < N; ++i)
00150     {
00151         const size_t erasePosition = wrapper.size() > 0 ? rand() % wrapper.size() : 0;
00152         wrapper.erase(erasePosition);
00153         ASSERT_EQ(N - i, wrapper.size()) << "i = " << i;
00154     }
00155 #else
00156     ADD_FAILURE() << "Method erase() not implemented!";
00157 #endif
00158 }
00159
00160 TEST_F(ContainerWrapperTester, count_expectedAllElementsCounted)
00161 {
00162 #if IMPLEMENTED_COUNT == 1
00163     const auto sourceElements = prepareSourceContainer();
00164     const auto expectedCount = std::accumulate(sourceElements.begin(), sourceElements.end(),
      value_type{});
00165
00166     const ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00167
00168     ASSERT_EQ(expectedCount, wrapper.count());
00169 #else
00170     ADD_FAILURE() << "Method count() not implemented!";
00171 #endif
00172 }
00173
00174 TEST_F(ContainerWrapperTester, findElement)
00175 {
00176 #if IMPLEMENTED_FIND == 1
00177     const auto sourceElements = prepareSourceContainer();
00178
00179     const ContainerWrapper wrapper(sourceElements.data(), sourceElements.size());
00180
00181     const value_type notExistingElement = 2 * N;
00182     ASSERT_EQ(std::numeric_limits<size_t>::max(), wrapper.find(notExistingElement));
00183
00184     for (size_t i=0; i < N / 10; ++i)
00185     {
00186         const size_t expectedPosition = rand() % sourceElements.size();
00187         const auto element2Find = sourceElements[expectedPosition];
00188         ASSERT_EQ(expectedPosition, wrapper.find(element2Find)) << "i = " << i << ", element2Find = " <<
      element2Find;
00189     }
00190 #else
00191     ADD_FAILURE() << "Method find() not implemented!";
00192 #endif
00193 }
```

Index