mc-homology

Generated by Doxygen 1.16.1

Chapter 1

Concept Index

1.1 Concepts

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

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 documented files with brief descriptions:

Chapter 5

Concept Documentation

5.1 algebra::AdditiveGroup Concept Reference

An additive group concept.

```
#include <algebraic_concepts.h>
```

5.1.1 Concept definition

```
template<class T>
concept AdditiveGroup = std::regular<T> && std::default_initializable<T>
    && Commutative<T> && requires(T x, T y) {
            { x + y } -> std::convertible_to<T>;
            { x - y } -> std::convertible_to<T>;
            { x += y } -> std::same_as<T&>;
            { x -= y } -> std::same_as<T&>;
            { +x } -> std::convertible_to<T>;
            { -x } -> std::convertible_to<T>;
            { T::zero() } -> std::convertible_to<T>;
        }
```

5.1.2 Detailed Description

An additive group concept.

`AdditiveGroup` concept declares that a type models a mathematical definition of an additive group, that is an abelian group that, in which the group operation is expressed with '+'.

5.1.3 Semantic requirements

For `x`, `y`, `z` of type `T`

1. `(x + y) + z == x + (y + z)`
2. `x + T::zero() == T::zero() + x == x`
3. `x + (-x) == (-x) + x == T::zero()`
4. `x + y == y + x`

5.2 algebra::Commutative Concept Reference

An abelian structure concept.

`#include <algebraic_concepts.h>`

5.2.1 Concept definition

```
template<class T>
concept Commutative = is_commutative_v<T>
```

5.2.2 Detailed Description

An abelian structure concept.

`Abelian` concept declares that a type's main operation is abelian, that is, for any `T x`, `T y` and operation `@` (+ lub −) we have `x @ y == y @ x`.

5.2.3 Semantic requirements

For `x` and `y` and operation `@`

1. `x @ y == y @ x`

5.3 algebra::CommutativeRing Concept Reference

An commutative ring concept.

`#include <algebraic_concepts.h>`

5.3.1 Concept definition

```
template<class T>
concept CommutativeRing = Ring<T> && Commutative<T>
```

5.3.2 Detailed Description

An commutative ring concept.

`CommutativeRing` concept declares that a type models a mathematical definition of a commutative ring, where addition is denoted by + and multiplication by ∗.

5.3.3 Sematic requirements

Type `T` satisfies all semantic requirements for `Ring` and additionally, for `x`, `y` of type `T`

1. `x * y == y * x`

5.4 algebra::EuclideanDomain Concept Reference

An euclidean domain concept.

`#include <algebraic_concepts.h>`

5.4.1 Concept definition

```
template<class T>
concept EuclideanDomain = CommutativeRing<T> && requires(T a, T b, T x, T y) {
    { divide(x, y) } -> std::same_as<DivResult<T»;
    { x.euclidean_function() } -> std::same_as<int>;
}
```

5.4.2 Detailed Description

An euclidean domain concept.

`EuclideanDomain` concept declares that a type models a mathematical definition of an Euclidean domain, that is a commutative ring, where we can define an euclidean function f, that satisfies two conditions:

1. For a and nonzero b there exist q and r satisfying a = q $*$ b + r and f(r) $<$ f(b)
2. For a and b nonzero we have f(a) $<$= f(b)

Additionally, we require function `divide`, which returns the numbers q and r.

5.4.3 Semantic requirements

Type 'T' satisfies all semantic requirements for `CommutativeRing` and additionally, for `x` and `y != T::`↩
`zero()`

1. `x = divide(x, y).quotient * y + divide(x, y).remainder`
2. `x.euclidean_function() <= (x * y).euclidean_function()`

5.5 algebra::Field Concept Reference

A field concept.

`#include <algebraic_concepts.h>`

5.5.1 Concept definition

```
template<class T>
concept Field = CommutativeRing<T> && requires(T x, T y) {
    { x / y } -> std::convertible_to<T>;
    { x /= y } -> std::same_as<T&>;
}
```

5.5.2 Detailed Description

A field concept.

`Field` concept declares that a type models a methematical definition of a field, where addition is denoted by + and multiplication is denoted by ∗. Additionaly, the is a divison operator '/'

5.5.3 Semantic requirements

Type `T` satisfies all semantic requirements for `EuclideanDomain` and additionally, for `x != T::zero()`

1. `x * (T::one()/x) == T::one()`

5.6 algebra::Group Concept Reference

A group concept.

`#include <algebraic_concepts.h>`

5.6.1 Concept definition

```
template<class T>
concept Group =
    std::regular<T> && std::default_initializable<T> && requires(T x, T y) {
        { x * y } -> std::convertible_to<T>;
        { x / y } -> std::convertible_to<T>;
        { x *= y } -> std::same_as<T&>;
        { x /= y } -> std::same_as<T&>;
        { T::one() } -> std::convertible_to<T>;
    }
```

5.6.2 Detailed Description

A group concept.

`Group` concept declares that a type models a mathematical definition of a group, that is an abelian group that, in which the group operation is expressed with '∗'.

5.6.3 Semantic requirements

For `x`, `y`, `z` of type `T`

1. `(x * y) * z == x * (y * z)`
2. `x * T::one() == T::one() * x == x`
3. `x * (T::one() / x) == (T::one() / x) * x == T::one()`

5.7 algebra::Ring Concept Reference

A ring concept.

`#include <algebraic_concepts.h>`

5.7.1 Concept definition

```
template<class T>
concept Ring = AdditiveGroup<T> && requires(T x, T y) {
    { x * y } -> std::convertible_to<T>;
    { x *= y } -> std::same_as<T&>;
    { T::one() } -> std::convertible_to<T>;
}
```

5.7.2 Detailed Description

A ring concept.

`Ring` concept declares that a type models a mathematical definition of a ring, where addition is denoted by + and multiplication by ∗.

5.7.3 Sematic requirements

Type `T` satisfies all semantic requirements for `AdditiveGroup` and additionally, for `x`, `y`, `z` of type `T`

1. `(x * y) * z == x * (y * z)`
2. `x * (y + z) == x * y + x * z`
3. `(x + y) * z == x * z + y * z`
4. `T::one() * x == x * T::one() == x`

Chapter 6

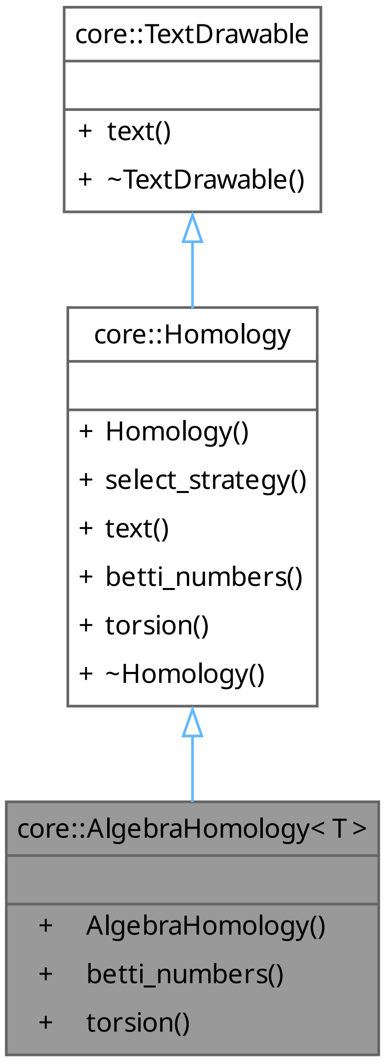
Class Documentation

6.1 core::AlgebraHomology< T > Class Template Reference

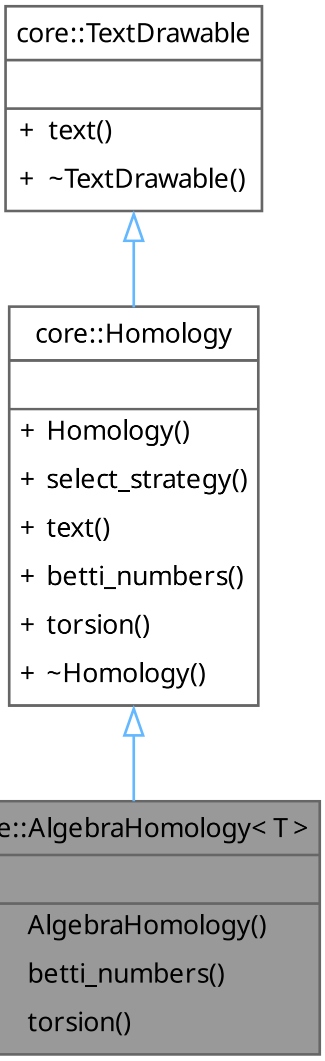
Concrete subclass of Homology, based on algebra library.

```
#include <algebra_homology.h>
```

Inheritance diagram for core::AlgebraHomology< T >:



Collaboration diagram for core::AlgebraHomology< T >:



Public Member Functions

- **AlgebraHomology** (algebra::Homology< T > homology, std::unique_ptr< HomologyPrintingStrategy > printing_strategy=std::make_unique< HomologyRawPrint >())

    Constructs homology and stores printing strategy.
- std::vector< std::size_t > betti_numbers () const override

    Returns text representation of the betti numbers.
- std::vector< std::vector< std::string > > torsion () const override

    Returns text represetnation of torsion.

Public Member Functions inherited from core::Homology

- Homology (std::unique_ptr< HomologyPrintingStrategy > printing_strategy=std::make_unique< HomologyRawPrint >())

    Constructs a new homology instance.
- void select_strategy (std::unique_ptr< HomologyPrintingStrategy > printing_strategy)

    Select a new printing_strategy.
- std::string text () const override

    Outputs the test form of the homology using given strategy.

Public Member Functions inherited from core::TextDrawable

- virtual ~**TextDrawable** ()

    Virtual destructor.

6.1.1 Detailed Description

template<class T>
class core::AlgebraHomology< T >

Concrete subclass of Homology, based on algebra library.

6.1.2 Member Function Documentation

6.1.2.1 betti_numbers()

```
template<class T>
std::vector< std::size_t > core::AlgebraHomology< T >::betti_numbers () const  [inline],
[override], [virtual]
```

Returns text representation of the betti numbers.

Implements core::Homology.

6.1.2.2 torsion()

```
template<class T>
std::vector< std::vector< std::string > > core::AlgebraHomology< T >::torsion () const  [inline],
[override], [virtual]
```

Returns text represetnation of torsion.

Implements core::Homology.

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

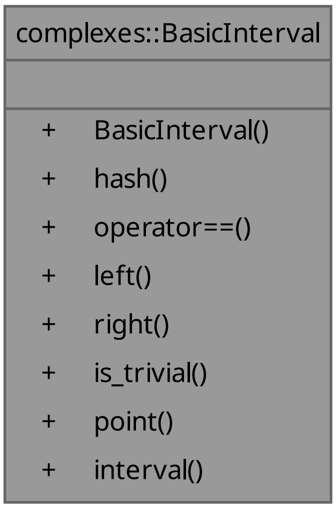
- mc-homology/core/include/core/algebra_homology.h

6.2 complexes::BasicInterval Class Reference

A struct representing an interval of length 0 or 1.

```
#include <cubical_complex.h>
```

Collaboration diagram for complexes::BasicInterval:



Public Member Functions

- **BasicInterval** ()
  Returns an singleton of 0.
- std::size_t **hash** () const
  Computer a hash of the inverval.
- bool **operator==** (BasicInterval const &) const
  Compares two intervals.
- int **left** () const
  Returns the left end of the interval.
- int **right** () const
  Returns the right end of the interval.
- bool **is_trivial** () const
  Returns true, if the interval is a singleton, false otherwise.

Static Public Member Functions

- static BasicInterval **point** (int p)
  Creates a trivial interval.
- static BasicInterval interval (int left)
  Creates a interval of length one starting in left.

6.2.1 Detailed Description

A struct representing an interval of length 0 or 1.

A struct representing an interval of a form [`left`, `right`], where `right - left < 2`

6.2.2 Member Function Documentation

6.2.2.1 interval()

```
BasicInterval complexes::BasicInterval::interval (
            int left) [static]
```

Creates a interval of length one starting in left.

Parameters

left	end of the interval

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

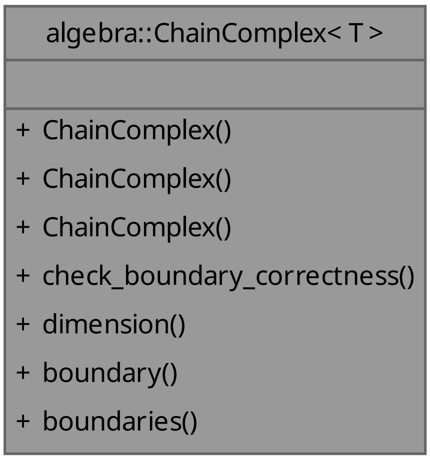
- mc-homology/complexes/include/complexes/cubical_complex.h
- mc-homology/complexes/src/cubical_complex.cpp

6.3 algebra::ChainComplex< T > Class Template Reference

Class representing a chain complex with coefficients `T`.

`#include <chain_complex.h>`

Collaboration diagram for algebra::ChainComplex< T >:



Public Member Functions

- template<std::ranges::sized_range R>
  constexpr ChainComplex (R &&boundaries)

  Constructs the chain complex from boundary matrices and checks, if they satisfy the chain complex condition.
- template<std::ranges::sized_range R>
  constexpr ChainComplex (SkipCorrectnessCheckT, R &&boundaries)

  Constructs the chain complex from the boundary matrices without checking the chain complex condition.
- constexpr bool **check_boundary_correctness** () const

  Checks, if the boundaries satisfy the chain complex condition.
- constexpr std::size_t **dimension** () const noexcept

  Return the dimension (the number of boundary matrices minus 1) of the chain complex.
- constexpr Matrix< T > const & boundary (std::size_t dim) const

  Return the boundary operator at dimension `dim`.
- constexpr std::vector< Matrix< T > > const & **boundaries** () const noexcept

  Return the vector of boundary operators.

6.3.1 Detailed Description

template<class T>
class algebra::ChainComplex< T >

Class representing a chain complex with coefficients `T`.

Chain complex is a free module over T with submodules `(C_d, ..., C_2, C_1, C_0, C_(-1))` and linear operators `B_n : C_n -> C_(n-1)` satisfying B_n*B_(n+1) == 0 (the chain complex condidition).

Template Parameters

T	Class of the coefficients

6.3.2 Constructor & Destructor Documentation

6.3.2.1 ChainComplex() [1/2]

```
template<class T>
template<std::ranges::sized_range R>
algebra::ChainComplex< T >::ChainComplex (
            R && boundaries)  [inline], [constexpr]
```

Constructs the chain complex from boundary matrices and checks, if they satisfy the chain complex condition.

Parameters

boundaries	The range containing the boundary opeartors

6.3.2.2 ChainComplex() [2/2]

```
template<class T>
template<std::ranges::sized_range R>
algebra::ChainComplex< T >::ChainComplex (
            SkipCorrectnessCheckT ,
            R && boundaries)  [inline], [constexpr]
```

Constructs the chain complex from the boundary matrices without checking the chain complex condition.

Use with caution!

Parameters

boundaries	The range containing the boundary opeartors

6.3.3 Member Function Documentation

6.3.3.1 boundary()

```
template<class T>
Matrix< T > const & algebra::ChainComplex< T >::boundary (
            std::size_t dim) const  [inline], [constexpr]
```

Return the boundary operator at dimension `dim`.

Parameters

dim	Dimension of the boundary operator

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

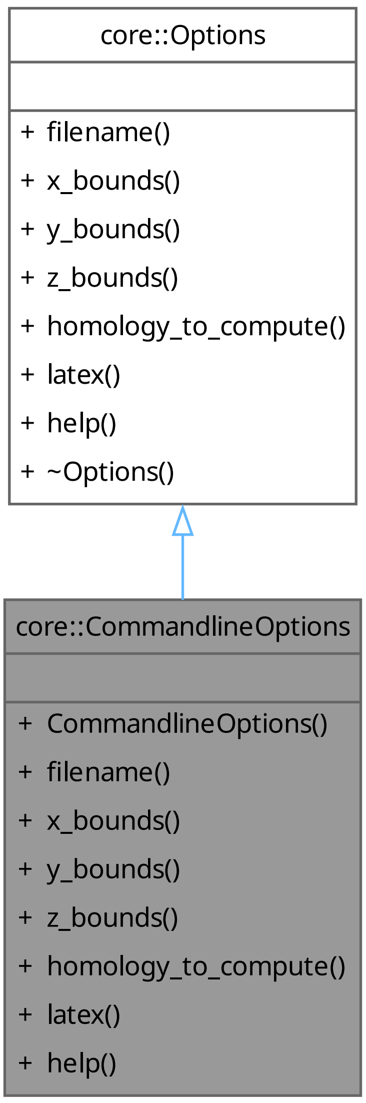
- mc-homology/algebra/include/algebra/chain_complex.h

6.4 core::CommandlineOptions Class Reference

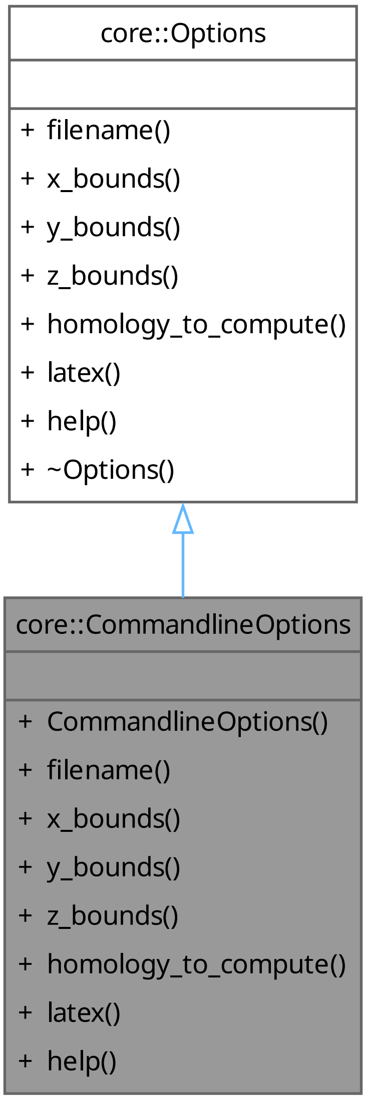
A class for storing user's options from the commandline.

```
#include <options.h>
```

Inheritance diagram for core::CommandlineOptions:



Collaboration diagram for core::CommandlineOptions:



Public Member Functions

- **CommandlineOptions** (int argc, char $**$argv)

  Parse commandline arguments.
- std::filesystem::path filename () const override

  Path to the region directory.
- std::pair< int, int > x_bounds () const override

  Bounds on x axis.
- std::pair< int, int > y_bounds () const override

  Bounds on y axis.
- std::pair< int, int > z_bounds () const override

Bounds on y axis.

- HomologyChoice homology_to_compute () const override

  Type of homology to compute.
- bool latex () const override

  Whether to print latex output.
- bool help () const override

  Whether the user requested help.

Public Member Functions inherited from core::Options

- virtual ∼**Options** ()

  virtual destructor

6.4.1 Detailed Description

A class for storing user's options from the commandline.

6.4.2 Member Function Documentation

6.4.2.1 filename()

```
std::filesystem::path core::CommandlineOptions::filename () const  [override], [virtual]
```

Path to the region directory.

Implements core::Options.

6.4.2.2 help()

```
bool core::CommandlineOptions::help () const  [override], [virtual]
```

Whether the user requested help.

Implements core::Options.

6.4.2.3 homology_to_compute()

```
HomologyChoice core::CommandlineOptions::homology_to_compute () const  [override], [virtual]
```

Type of homology to compute.

Implements core::Options.

6.4.2.4 latex()

```
bool core::CommandlineOptions::latex () const  [override], [virtual]
```

Whether to print latex output.

Implements core::Options.

6.4.2.5 x_bounds()

```
std::pair< int, int > core::CommandlineOptions::x_bounds () const  [override], [virtual]
```

Bounds on x axis.

Implements core::Options.

6.4.2.6 y_bounds()

```
std::pair< int, int > core::CommandlineOptions::y_bounds () const  [override], [virtual]
```

Bounds on y axis.

Implements core::Options.

6.4.2.7 z_bounds()

```
std::pair< int, int > core::CommandlineOptions::z_bounds () const  [override], [virtual]
```

Bounds on y axis.

Implements core::Options.

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

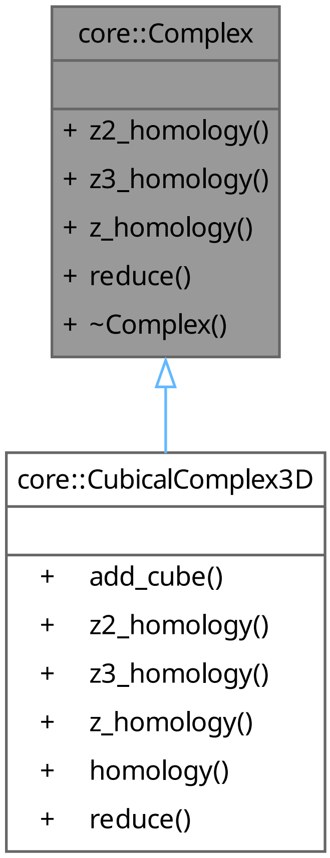
- mc-homology/core/include/core/options.h
- mc-homology/core/src/options.cpp

6.5 core::Complex Class Reference

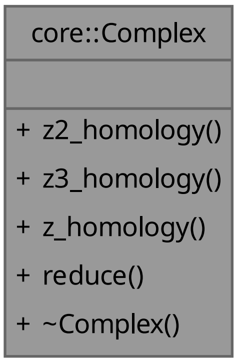
Interface for complexes.

```
#include <complex.h>
```

Inheritance diagram for core::Complex:



Collaboration diagram for core::Complex:



Public Member Functions

- virtual std::unique_ptr< Homology > z2_homology () const =0

    Computes Z2 homology of the complex.
- virtual std::unique_ptr< Homology > z3_homology () const =0

    Computes Z3 homology of the complex.
- virtual std::unique_ptr< Homology > z_homology () const =0

    Computes Z homology of the complex.
- virtual void reduce ()=0

    Decreases the complex's size without changing its homology.
- virtual ∼**Complex** ()

    Virtual destructor.

6.5.1 Detailed Description

Interface for complexes.

6.5.2 Member Function Documentation

6.5.2.1 reduce()

```
virtual void core::Complex::reduce ()  [pure virtual]
```

Decreases the complex's size without changing its homology.

Implemented in core::CubicalComplex3D.

6.5.2.2 z2_homology()

```
virtual std::unique_ptr< Homology > core::Complex::z2_homology () const  [pure virtual]
```

Computes Z2 homology of the complex.

Implemented in core::CubicalComplex3D.

6.5.2.3 z3_homology()

```
virtual std::unique_ptr< Homology > core::Complex::z3_homology () const  [pure virtual]
```

Computes Z3 homology of the complex.

Implemented in core::CubicalComplex3D.

6.5.2.4 z_homology()

```
virtual std::unique_ptr< Homology > core::Complex::z_homology () const  [pure virtual]
```

Computes Z homology of the complex.

Implemented in core::CubicalComplex3D.

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

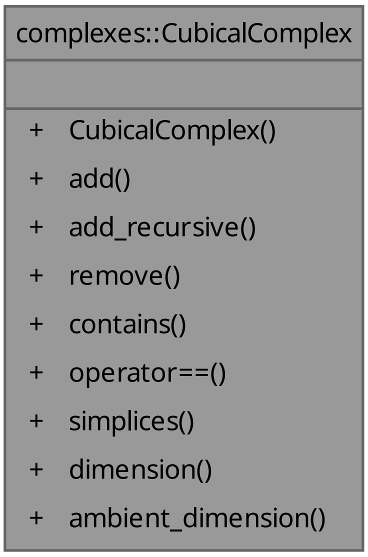
- mc-homology/core/include/core/complex.h
- mc-homology/core/src/complex.cpp

6.6 complexes::CubicalComplex Class Reference

Class representing a cubical complex.

`#include <cubical_complex.h>`

Collaboration diagram for complexes::CubicalComplex:



Public Member Functions

- CubicalComplex ()

    Default constructor for the CubicalComplex.
- bool add (CubicalSimplex simplex)

    Adds a cubical simplex to the complex.
- void add_recursive (CubicalSimplex simplex)

    Adds a cubical simplex to the complex together with its boundary.
- bool remove (CubicalSimplex const &simplex)

    Removes a cubical simplex from the complex.
- bool contains (CubicalSimplex const &simplex) const

    Checks, if a cubical complex contains simplex.
- bool **operator==** (CubicalComplex const &) const

    Equality comparison operator.
- std::vector< std::unordered_set< CubicalSimplex > > const & simplices () const

    Grants access to the simplexes of the complex.
- std::size_t **dimension** () const

    Returns the dimension of the simplex.
- std::size_t **ambient_dimension** () const

    Returns the ambient dimension of the simplex.

6.6.1 Detailed Description

Class representing a cubical complex.

A class representing a topological space constructed from cubical simplexes.

6.6.2 Constructor & Destructor Documentation

6.6.2.1 CubicalComplex()

```
complexes::CubicalComplex::CubicalComplex ()  [default]
```

Default constructor for the CubicalComplex.

Creates an empty complex

6.6.3 Member Function Documentation

6.6.3.1 add()

```
bool complexes::CubicalComplex::add (
            CubicalSimplex simplex)
```

Adds a cubical simplex to the complex.

In order to add a cubical simplex to the complex, all elements of its boundary must already be part of the complex. 0 dimensional simplices can always be added.

Parameters

simplex	Simplex to add

Returns

true, if simplex has been successfully added, false otherwise

6.6.3.2 add_recursive()

```
void complexes::CubicalComplex::add_recursive (
            CubicalSimplex simplex)
```

Adds a cubical simplex to the complex together with its boundary.

Parameters

simplex	Simplex to add

6.6.3.3 contains()

```
bool complexes::CubicalComplex::contains (
            CubicalSimplex const & simplex) const
```

Checks, if a cubical complex contains simplex.

Parameters

simplex	simplex to test

Returns

true, if simplex is a part of the complex, false otherwise

6.6.3.4 remove()

```
bool complexes::CubicalComplex::remove (
            CubicalSimplex const & simplex)
```

Removes a cubical simplex from the complex.

In order to remove a cubical complex, it has to have an empty coboundary, that is, it cannot be a part of the boundary of any other simplex

Parameters

simplex	Simplex to remove

Returns

true, if removal was successfull, false otherwise

6.6.3.5 simplices()

```
std::vector< std::unordered_set< CubicalSimplex > > const & complexes::CubicalComplex::↩
simplices () const
```

Grants access to the simplexes of the complex.

The simplices are returned in a vector, where `n`'th element is a unordered set containing simplices of dimension `n`.

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

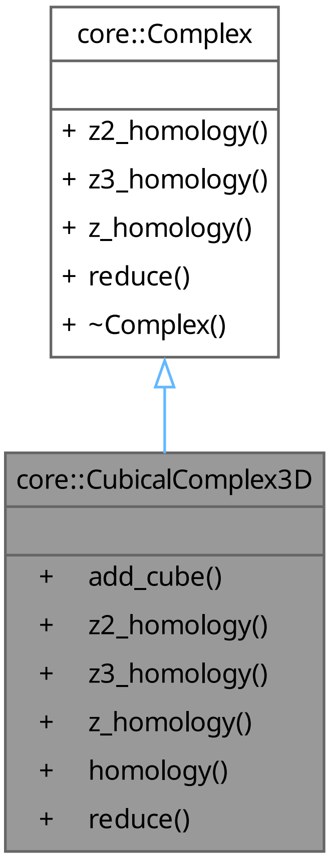
- mc-homology/complexes/include/complexes/cubical_complex.h
- mc-homology/complexes/src/cubical_complex.cpp

6.7 core::CubicalComplex3D Class Reference

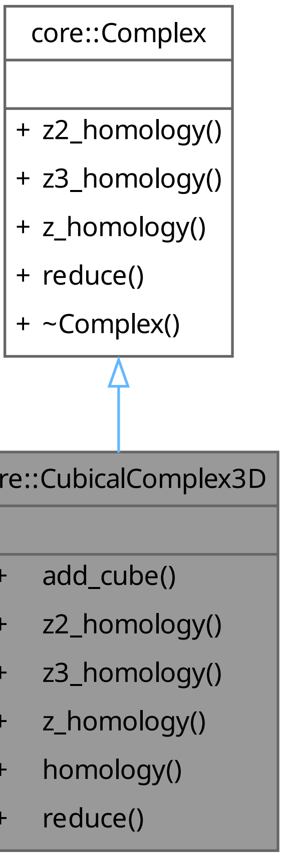
A class representing a cubical complex in 3D space.

```
#include <cubical_complex_3d.h>
```

Inheritance diagram for core::CubicalComplex3D:



Collaboration diagram for core::CubicalComplex3D:



Public Member Functions

- void **add_cube** (int x, int y, int z)

  Adds a cube to the complex.
- std::unique_ptr< Homology > z2_homology () const override

  Computes Z2 homology of the complex.
- std::unique_ptr< Homology > z3_homology () const override

  Computes Z3 homology of the complex.
- std::unique_ptr< Homology > z_homology () const override

  Computes Z homology of the complex.
- template<class T>
  std::unique_ptr< AlgebraHomology< T > > **homology** () const

  Computes homology of the complex for coefficients of type T.
- void reduce () override

  Decreases the complex's size without changing its homology.

Public Member Functions inherited from core::Complex

- virtual ∼**Complex** ()

  Virtual destructor.

6.7.1 Detailed Description

A class representing a cubical complex in 3D space.

6.7.2 Member Function Documentation

6.7.2.1 reduce()

```
void core::CubicalComplex3D::reduce ()  [override], [virtual]
```

Decreases the complex's size without changing its homology.

Implements core::Complex.

6.7.2.2 z2_homology()

```
std::unique_ptr< Homology > core::CubicalComplex3D::z2_homology () const  [override], [virtual]
```

Computes Z2 homology of the complex.

Implements core::Complex.

6.7.2.3 z3_homology()

```
std::unique_ptr< Homology > core::CubicalComplex3D::z3_homology () const  [override], [virtual]
```

Computes Z3 homology of the complex.

Implements core::Complex.

6.7.2.4 z_homology()

```
std::unique_ptr< Homology > core::CubicalComplex3D::z_homology () const  [override], [virtual]
```

Computes Z homology of the complex.

Implements core::Complex.

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

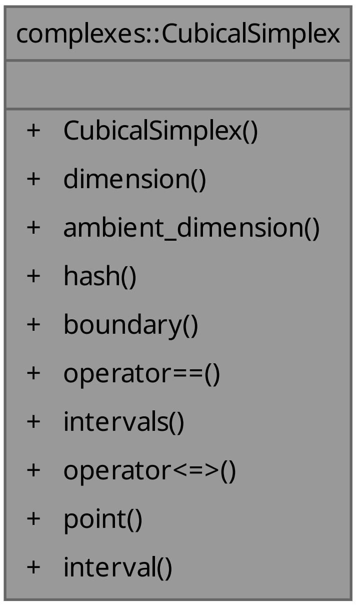
- mc-homology/core/include/core/cubical_complex_3d.h
- mc-homology/core/src/cubical_complex_3d.cpp

6.8 complexes::CubicalSimplex Class Reference

A class representing a single cubical simplex in a complex.

```
#include <cubical_complex.h>
```

Collaboration diagram for complexes::CubicalSimplex:



Public Member Functions

- CubicalSimplex (std::vector< BasicInterval > intervals)

  Constructs a simplex from a vector of intervals.
- std::size_t **dimension** () const

  Returns the dimension of the simplex.
- std::size_t **ambient_dimension** () const

  Returns the ambient dimension of the simplex.
- std::size_t **hash** () const

  Computes a hash of the simplex.
- std::vector< CubicalSimplex > boundary () const

  Returns the boundary of a simplex.
- bool **operator==** (CubicalSimplex const &) const

  Compares two simplices.
- std::vector< BasicInterval > const & **intervals** () const

  Returns the underlying intervals.
- std::strong_ordering operator<=> (CubicalSimplex const &simplex) const

  Comparison operator for simplices.

Static Public Member Functions

- static CubicalSimplex point (int p)

  Creates a cubical simplex out of a single 1d point.
- static CubicalSimplex interval (int left)

  Creates a cubical simplex out of a single 1d interval.

Friends

- CubicalSimplex product (CubicalSimplex const &s1, CubicalSimplex const &s2)

  Creates a cubical simplex by joining together two simplices.

6.8.1 Detailed Description

A class representing a single cubical simplex in a complex.

A cubical simplex is a product of basic intervals. The number of intervals is equal to the ambient dimension, while the number of non trivial intervals is equal to the dimension of the simplex.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 CubicalSimplex()

```
complexes::CubicalSimplex::CubicalSimplex (
            std::vector< BasicInterval > intervals)
```

Constructs a simplex from a vector of intervals.

Constructs a simplex from a vector of intervals. The interval must have at least one value.

Parameters

intervals	A nonempty vector of intervals

6.8.3 Member Function Documentation

6.8.3.1 boundary()

```
std::vector< CubicalSimplex > complexes::CubicalSimplex::boundary () const
```

Returns the boundary of a simplex.

The returned boundary is decresing in the sense of operator<=>.

6.8.3.2 interval()

```
CubicalSimplex complexes::CubicalSimplex::interval (
              int left)  [static]
```

Creates a cubical simplex out of a single 1d interval.

Parameters

left	Left bound of the interval

6.8.3.3 operator<=>()

```
std::strong_ordering complexes::CubicalSimplex::operator<=> (
              CubicalSimplex const & simplex) const
```

Comparison operator for simplices.

Comparison operator for simplices. A simplex A compares smaller than simplex B if and only if it has smaller dimension or the dimensions are equal and its interval vector is lexicographically smaller with order on intervals given by:

1. `[a, a + 1] < [b]`
2. `[a, a + 1] < [b, b + 1]`
3. `[a] < [b]`

for any `a < b`

6.8.3.4 point()

```
CubicalSimplex complexes::CubicalSimplex::point (
              int p)  [static]
```

Creates a cubical simplex out of a single 1d point.

Parameters

p	The point

6.8.4 Friends And Related Symbol Documentation

6.8.4.1 product

```
CubicalSimplex product (
              CubicalSimplex const & s1,
              CubicalSimplex const & s2)  [friend]
```

Creates a cubical simplex by joining together two simplices.

Creates a simplex in a higher dimension by taking a cartesian product and concatenating them

Parameters

s1	First simplex

s2	Second simplex

Returns

The result of concatenation

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

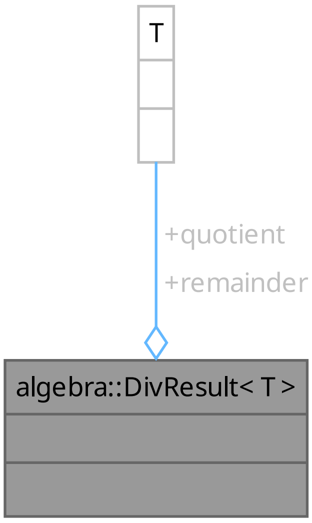
- mc-homology/complexes/include/complexes/cubical_complex.h
- mc-homology/complexes/src/cubical_complex.cpp

6.9 algebra::DivResult< T > Struct Template Reference

Result struct for the division operation.

```
#include <number_theory.h>
```

Collaboration diagram for algebra::DivResult< T >:



Public Attributes

- T **quotient** = {}

  Quotient of the division.
- T **remainder** = {}

  Remainder of the division.

6.9.1 Detailed Description

template< **class T**>
struct algebra::DivResult< **T** >

Result struct for the division operation.

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

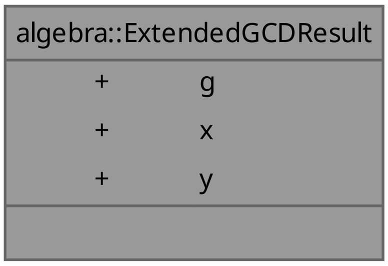
- mc-homology/algebra/include/algebra/number_theory.h

6.10 algebra::ExtendedGCDResult Struct Reference

Result struct for the extended gcd algorithm.

`#include <number_theory.h>`

Collaboration diagram for algebra::ExtendedGCDResult:



Public Attributes

- int **g** = 0

  Greatest common divisor.
- int **x** = 0

  First coefficient from the extended algorithm.
- int **y** = 0

  Second coefficient from the extended algorithm.

6.10.1 Detailed Description

Result struct for the extended gcd algorithm.

Results of the extended gcd algorithm. For input `a`, `b` satisfies `g == a * x + b * y`

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

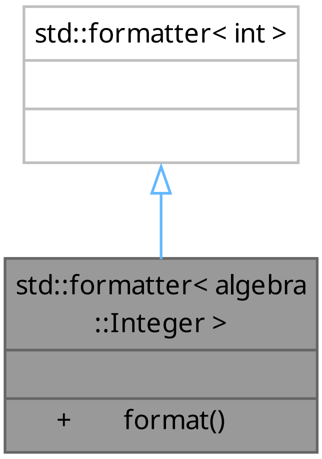
- mc-homology/algebra/include/algebra/number_theory.h

6.11 std::formatter< algebra::Integer > Struct Reference

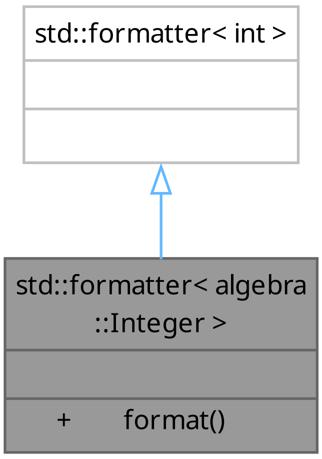
Formatter for Integer type.

`#include <integer.h>`

Inheritance diagram for std::formatter< algebra::Integer >:



Collaboration diagram for std::formatter< algebra::Integer >:



Public Member Functions

- template<class FmtContext>
  FmtContext::iterator format (algebra::Integer k, FmtContext &ctx) const

  Formats an integer.

6.11.1 Detailed Description

Formatter for Integer type.

Allows use of `std::format` with the `Integer` type. The format syntax is the same, as in the case of `int`.

6.11.2 Member Function Documentation

6.11.2.1 format()

```
template<class FmtContext>
FmtContext::iterator std::formatter< algebra::Integer >::format (
            algebra::Integer k,
            FmtContext & ctx) const  [inline]
```

Formats an integer.

Formats an integer using a formatting syntax for int.

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

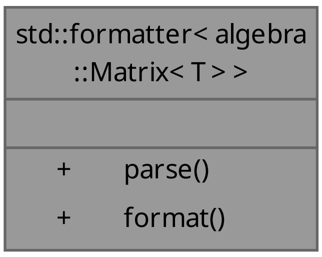
- mc-homology/algebra/include/algebra/integer.h

6.12 std::formatter< algebra::Matrix< T > > Struct Template Reference

Formatter for the `Matrix` type.

`#include <matrix.h>`

Collaboration diagram for std::formatter< algebra::Matrix< T > >:



Public Member Functions

- template<class ParseContext>
  constexpr ParseContext::iterator **parse** (ParseContext &ctx)

  Parses a context string.
- template<class FmtContext>
  FmtContext::iterator format (algebra::Matrix< T > const &matrix, FmtContext &ctx) const

  Formats a matrix.

6.12.1 Detailed Description

template<class T>
requires std::formattable<T, char>
struct std::formatter< algebra::Matrix< T > >

Formatter for the `Matrix` type.

Impletementation of the formatter for the `Matrix` type.

6.12.2 Format syntax

1. :[-|#][&ltcofficient_format_string&gt]

where &ltcoefficient_format_string&gt is the format string for the coefficient type. `-` or no specifier means that the matrix will be printed in a single line, `#` will print the matrix in multiple lines, making it easier to read.

6.12.3 Example

For `Matrix<Integer> matrix`

1. `std::format("{}", matrix)` outputs a single line matrix
2. `std::format("{:-}", matrix)` also outputs a single line matrix
3. `std::format("{:#}", matrix)` output a multi-line matrix
4. `std::format("{::b}", matrix)` output a single line matrix of integers in binary representation
5. `std::format("{:#:x}", matrix)` output a multi-line matrix of integers in hexadecimal representation

6.12.4 Member Function Documentation

6.12.4.1 format()

```
template<class T>
template<class FmtContext>
FmtContext::iterator std::formatter< algebra::Matrix< T > >::format (
            algebra::Matrix< T > const & matrix,
            FmtContext & ctx) const  [inline]
```

Formats a matrix.

Formats a matrix. Rules for the format string are specified in the class documentation.

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

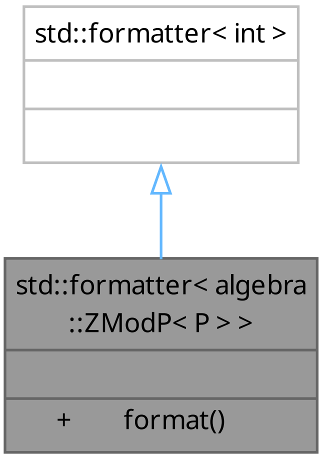
- mc-homology/algebra/include/algebra/matrix.h

6.13 std::formatter< algebra::ZModP< P > > Struct Template Reference

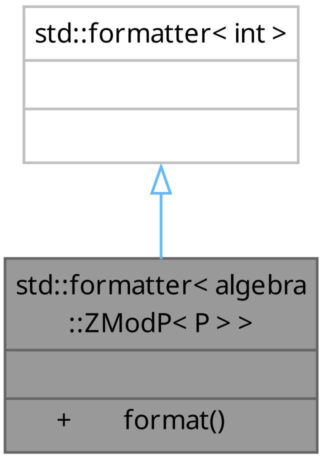
Formatter for ZModP type.

```
#include <modulo_fields.h>
```

Inheritance diagram for std::formatter< algebra::ZModP< P > >:



Collaboration diagram for std::formatter< algebra::ZModP< P > >:



Public Member Functions

- template<class FmtContext>
  FmtContext::iterator **format** (algebra::ZModP< P > x, FmtContext &ctx) const

    Formats an integer mod P.

6.13.1 Detailed Description

template<int P>
struct std::formatter< algebra::ZModP< P > >

Formatter for ZModP type.

Allows use of `std::format` with the `ZModP` type. The format syntax is the same, as in the case of `int`.

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

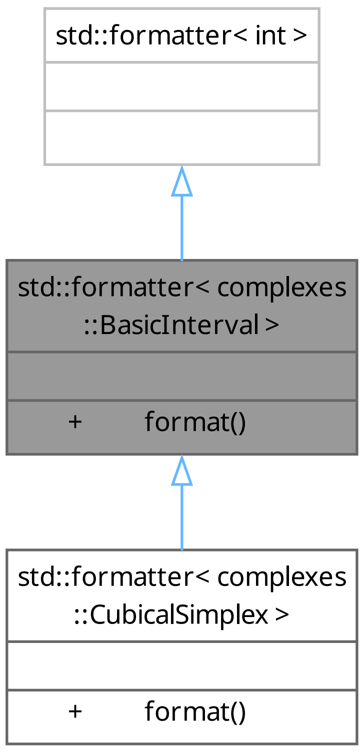
- mc-homology/algebra/include/algebra/modulo_fields.h

6.14 std::formatter< complexes::BasicInterval > Struct Reference

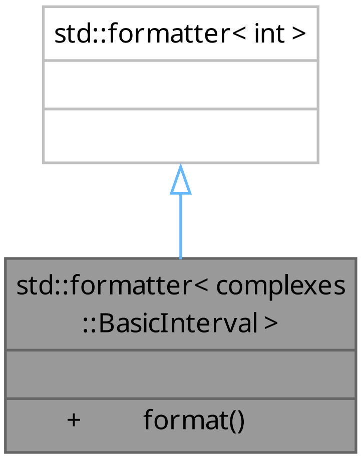
Formatter for BasicInterval type.

`#include <cubical_complex.h>`

Inheritance diagram for std::formatter< complexes::BasicInterval >:


std::formatter< int >
std::formatter< complexes ::BasicInterval >
+ format()
std::formatter< complexes ::CubicalSimplex >
+ format()

Collaboration diagram for std::formatter< complexes::BasicInterval >:



Public Member Functions

- template<class FmtContext>
  FmtContext::iterator format (complexes::BasicInterval i, FmtContext &ctx) const

  Formats an integer.

6.14.1 Detailed Description

Formatter for BasicInterval type.

Allows use of `std::format` with the `BasicInterval` type. The format syntax is the same, as in the case of `int`.

6.14.2 Member Function Documentation

6.14.2.1 format()

```
template<class FmtContext>
FmtContext::iterator std::formatter< complexes::BasicInterval >::format (
             complexes::BasicInterval i,
             FmtContext & ctx) const  [inline]
```

Formats an integer.

Formats an interval using a formatting syntax for int.

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

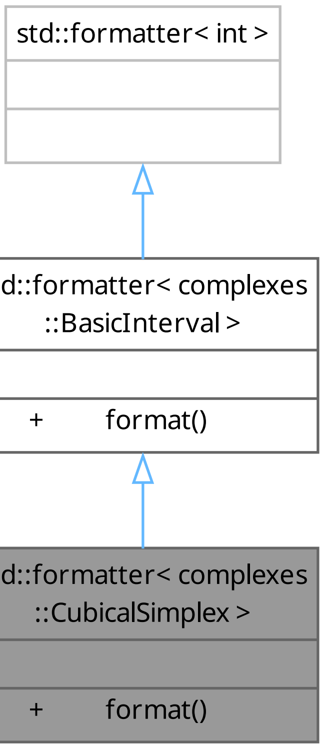
- mc-homology/complexes/include/complexes/cubical_complex.h

6.15 std::formatter< complexes::CubicalSimplex > Struct Reference

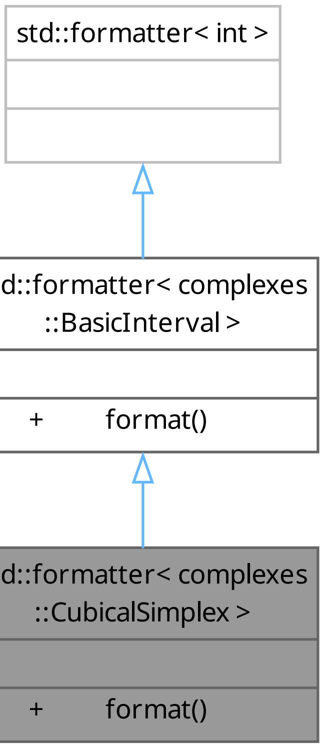
Formatter for CubicalSimplex type.

```
#include <cubical_complex.h>
```

Inheritance diagram for std::formatter< complexes::CubicalSimplex >:


std::formatter< int >
std::formatter< complexes ::BasicInterval >
+ format()
std::formatter< complexes ::CubicalSimplex >
+ format()

Collaboration diagram for std::formatter< complexes::CubicalSimplex >:


std::formatter< int >
std::formatter< complexes ::BasicInterval >
+ format()
std::formatter< complexes ::CubicalSimplex >
+ format()

Public Member Functions

- template<class FmtContext>
  FmtContext::iterator format (complexes::CubicalSimplex const &s, FmtContext &ctx) const

  Formats a cubical simplex.

Public Member Functions inherited from std::formatter< complexes::BasicInterval >

- template<class FmtContext>
  FmtContext::iterator format (complexes::BasicInterval i, FmtContext &ctx) const

  Formats an integer.

6.15.1 Detailed Description

Formatter for CubicalSimplex type.

Allows use of `std::format` with the `CubicalSimplex` type. The format syntax is the same, as in the case of `BasicInterval`.

6.15.2 Member Function Documentation

6.15.2.1 format()

```
template<class FmtContext>
FmtContext::iterator std::formatter< complexes::CubicalSimplex >::format (
            complexes::CubicalSimplex const & s,
            FmtContext & ctx) const  [inline]
```

Formats a cubical simplex.

Formats an integer using a formatting syntax for an interval.

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

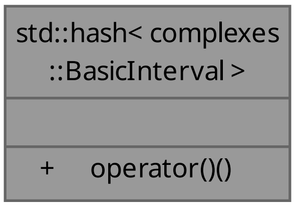
- mc-homology/complexes/include/complexes/cubical_complex.h

6.16 std::hash< complexes::BasicInterval > Struct Reference

std::hash specialization for Interval

```
#include <cubical_complex.h>
```

Collaboration diagram for std::hash< complexes::BasicInterval >:



Public Member Functions

- std::size_t **operator()** (complexes::BasicInterval const &i) const
  Returns a hash of an interval.

6.16.1 Detailed Description

std::hash specialization for Interval

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

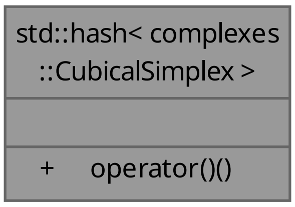
- mc-homology/complexes/include/complexes/cubical_complex.h
- mc-homology/complexes/src/cubical_complex.cpp

6.17 std::hash< complexes::CubicalSimplex > Struct Reference

std::hash specialization for CubicalSimplex

```
#include <cubical_complex.h>
```

Collaboration diagram for std::hash< complexes::CubicalSimplex >:



Public Member Functions

- std::size_t **operator()** (complexes::CubicalSimplex const &s) const

  Returns a hash of a cubical simplex.

6.17.1 Detailed Description

std::hash specialization for CubicalSimplex

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

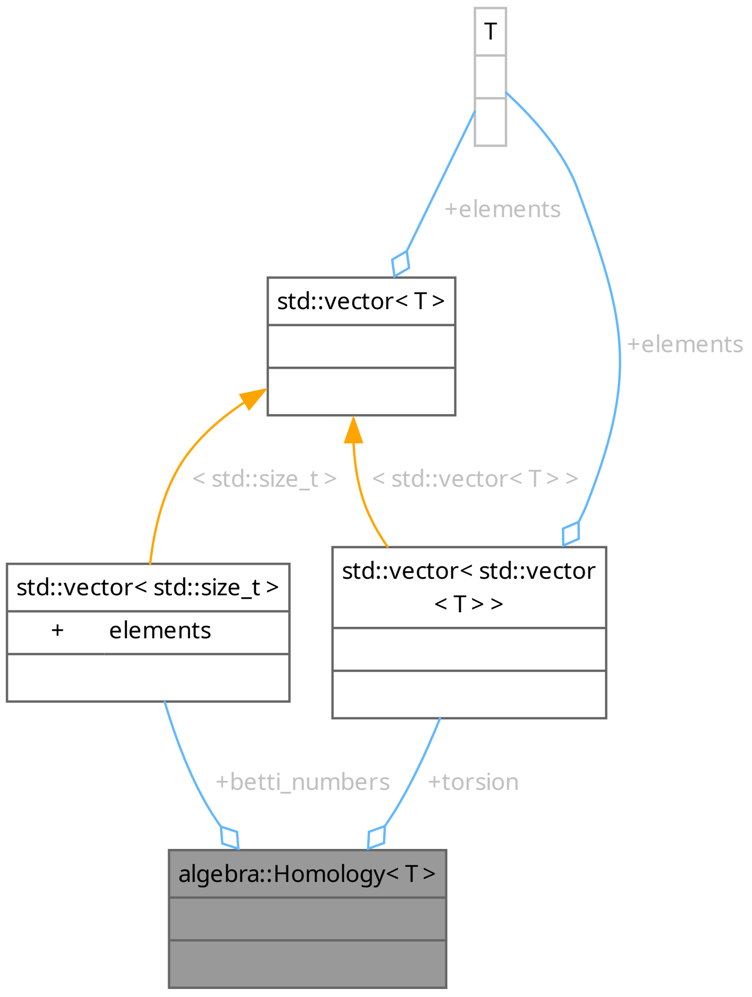
- mc-homology/complexes/include/complexes/cubical_complex.h
- mc-homology/complexes/src/cubical_complex.cpp

6.18 algebra::Homology< T > Struct Template Reference

Homology of a chain complex.

```
#include <chain_complex.h>
```

Collaboration diagram for algebra::Homology< T >:



Public Attributes

- std::vector< std::size_t > **betti_numbers** {}

  Betti numbers of a chain complex.
- std::vector< std::vector< T > > torsion {}

  Torsion of a chain complex.

6.18.1 Detailed Description

template<class T>
struct algebra::Homology< T >

Homology of a chain complex.

A struct containing homology information of a chain complex - its betti numbers and torsion group. The data is stored increasingly in dimension: dim(H_0) = betti_numbers[0], dim(H_1) = betti_numbers[1] itd.

6.18.2 Member Data Documentation

6.18.2.1 torsion

```
template<class T>
std::vector<std::vector<T> > algebra::Homology< T >::torsion {}
```

Torsion of a chain complex.

Information about the torsion group is stored in the following way: torsion[n] contains an array of elements such that torsion is equal to the simple sum of T/aT for a in the array.

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

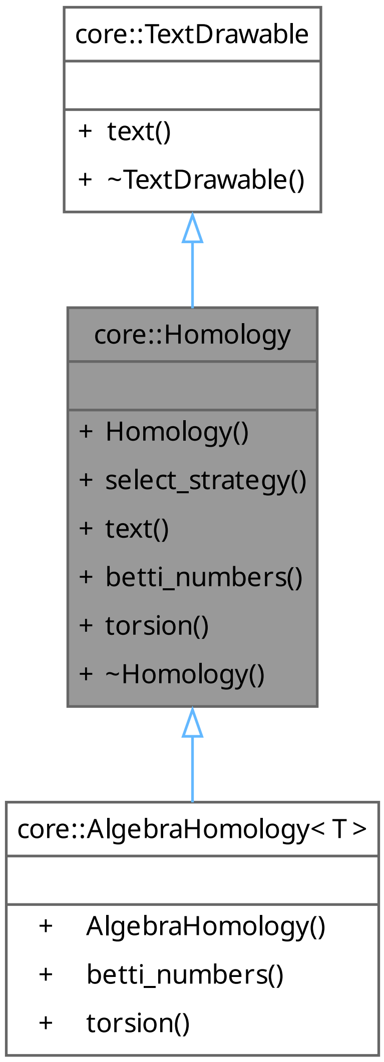
- mc-homology/algebra/include/algebra/chain_complex.h

6.19 core::Homology Class Reference

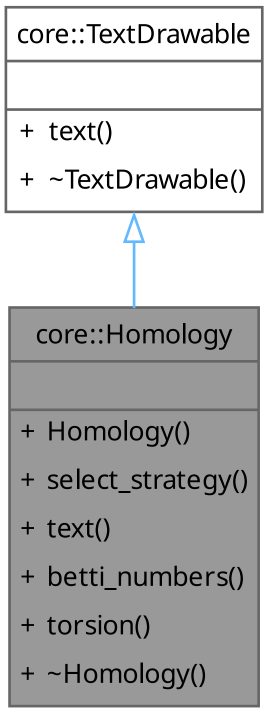
A homology interface.

```
#include <homology.h>
```

Inheritance diagram for core::Homology:



Collaboration diagram for core::Homology:



Public Member Functions

- Homology (std::unique_ptr< HomologyPrintingStrategy > printing_strategy=std::make_unique< HomologyRawPrint >())

  Constructs a new homology instance.
- void select_strategy (std::unique_ptr< HomologyPrintingStrategy > printing_strategy)

  Select a new printing_strategy.
- std::string text () const override

  Outputs the test form of the homology using given strategy.
- virtual std::vector< std::size_t > betti_numbers () const =0

  Returns text representations of stored betti numbers.
- virtual std::vector< std::vector< std::string > > torsion () const =0

  Returns text representations of stored torsion as the non-trivial part of the Smith's form diagonal.

Public Member Functions inherited from core::TextDrawable

- virtual ∼**TextDrawable** ()

  Virtual destructor.

6.19.1 Detailed Description

A homology interface.

Creates a common interface for various homology implementations. Additionally handles printing the result.

6.19.2 Constructor & Destructor Documentation

6.19.2.1 Homology()

```
core::Homology::Homology (
            std::unique_ptr< HomologyPrintingStrategy > printing_strategy = std::make_↩
unique<HomologyRawPrint>())
```

Constructs a new homology instance.

Constructs a new homology object with a specified printing strategy

Parameters

printing_strategy	A strategy used with printing

6.19.3 Member Function Documentation

6.19.3.1 betti_numbers()

```
virtual std::vector< std::size_t > core::Homology::betti_numbers () const  [pure virtual]
```

Returns text representations of stored betti numbers.

Implemented in core::AlgebraHomology< T >.

6.19.3.2 select_strategy()

```
void core::Homology::select_strategy (
            std::unique_ptr< HomologyPrintingStrategy > printing_strategy)
```

Select a new printing_strategy.

Parameters

printing_strategy	A strategy used with printing

6.19.3.3 text()

```
std::string core::Homology::text () const  [override], [virtual]
```

Outputs the test form of the homology using given strategy.

Implements core::TextDrawable.

6.19.3.4 torsion()

```
virtual std::vector< std::vector< std::string > > core::Homology::torsion () const  [pure
virtual]
```

Returns text representations of stored torsion as the non-trivial part of the Smith's form diagonal.

Implemented in core::AlgebraHomology< T >.

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

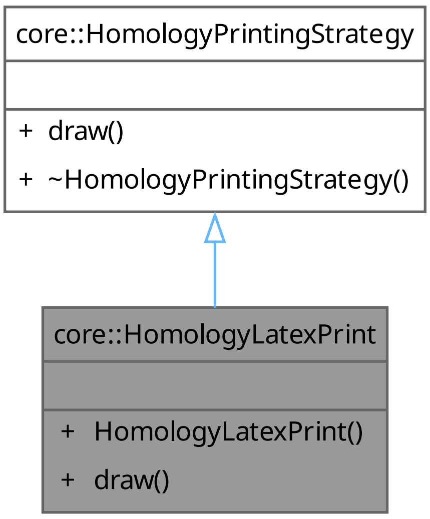
- mc-homology/core/include/core/homology.h
- mc-homology/core/src/homology.cpp

6.20 core::HomologyLatexPrint Class Reference

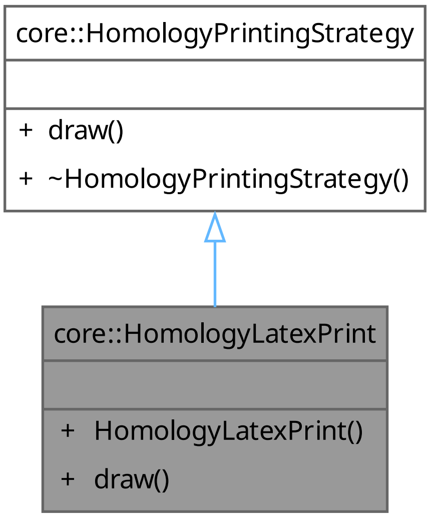
Prints a homology group in Latex syntax.

`#include <homology_printing_strategy.h>`

Inheritance diagram for core::HomologyLatexPrint:



Collaboration diagram for core::HomologyLatexPrint:



Public Member Functions

- HomologyLatexPrint (std::string ring_name, std::string homology_name="H")

  Constructor for the strategy.
- std::string draw (Homology const &homology) const override

  Returns the homology description in latex syntax.

Public Member Functions inherited from core::HomologyPrintingStrategy

- virtual ∼**HomologyPrintingStrategy** ()

  A virtual destructor.

6.20.1 Detailed Description

Prints a homology group in Latex syntax.

Prints a homology group in Latex syntax, making it easily embeddable into a Latex file.

6.20.2 Constructor & Destructor Documentation

6.20.2.1 HomologyLatexPrint()

```
core::HomologyLatexPrint::HomologyLatexPrint (
            std::string ring_name,
            std::string homology_name = "H")
```

Constructor for the strategy.

Constructor for the strategy, allows to select names for the homology and torsion groups

Parameters

ring_name	Name for the coefficient ring

homology_name	Name for the homology group

6.20.3 Member Function Documentation

6.20.3.1 draw()

```
std::string core::HomologyLatexPrint::draw (
            Homology const & homology) const  [override], [virtual]
```

Returns the homology description in latex syntax.

Implements core::HomologyPrintingStrategy.

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

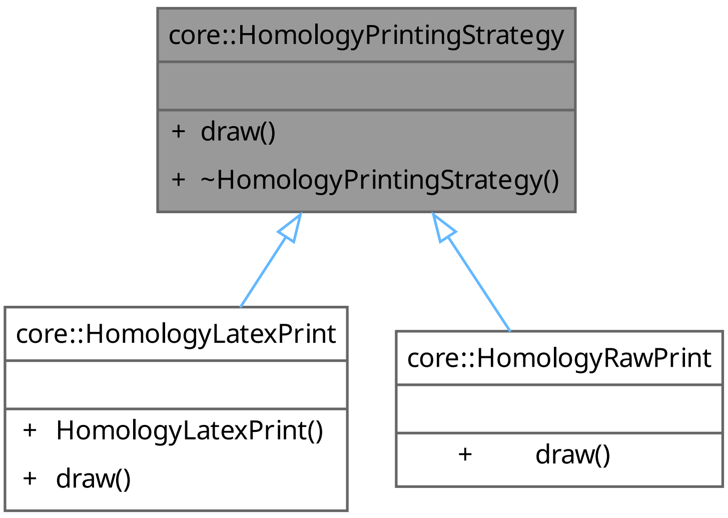
- mc-homology/core/include/core/homology_printing_strategy.h
- mc-homology/core/src/homology_printing_strategy.cpp

6.21 core::HomologyPrintingStrategy Class Reference

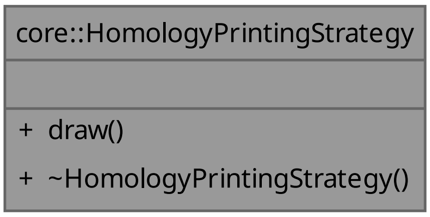
A strategy for printing a homology group.

```
#include <homology_printing_strategy.h>
```

Inheritance diagram for core::HomologyPrintingStrategy:



Collaboration diagram for core::HomologyPrintingStrategy:



Public Member Functions

- virtual std::string draw (Homology const &homology) const =0

  Returns the text representation of the homology group.
- virtual ∼**HomologyPrintingStrategy** ()

  A virtual destructor.

6.21.1 Detailed Description

A strategy for printing a homology group.

6.21.2 Member Function Documentation

6.21.2.1 draw()

```
virtual std::string core::HomologyPrintingStrategy::draw (
            Homology const & homology) const  [pure virtual]
```

Returns the text representation of the homology group.

Implemented in core::HomologyLatexPrint, and core::HomologyRawPrint.

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

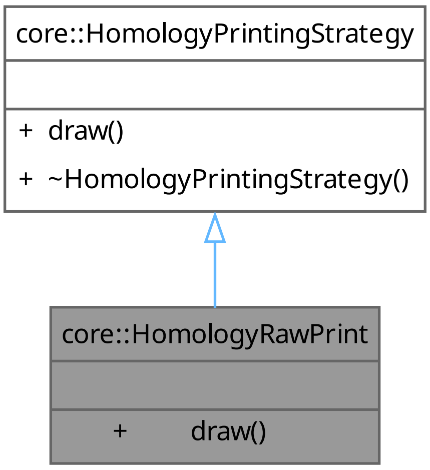
- mc-homology/core/include/core/homology_printing_strategy.h
- mc-homology/core/src/homology_printing_strategy.cpp

6.22 core::HomologyRawPrint Class Reference

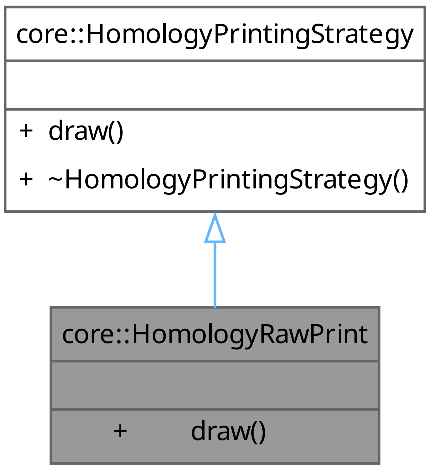
A strategy for basic string representation of homology.

```
#include <homology_printing_strategy.h>
```

Inheritance diagram for core::HomologyRawPrint:



Collaboration diagram for core::HomologyRawPrint:



Public Member Functions

- std::string draw (Homology const &homology) const override

    Prints betti numbers and torsion group.

Public Member Functions inherited from core::HomologyPrintingStrategy

- virtual ∼**HomologyPrintingStrategy** ()

    A virtual destructor.

6.22.1 Detailed Description

A strategy for basic string representation of homology.

6.22.2 Member Function Documentation

6.22.2.1 draw()

```
std::string core::HomologyRawPrint::draw (
            Homology const & homology) const  [override], [virtual]
```

Prints betti numbers and torsion group.

Implements core::HomologyPrintingStrategy.

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

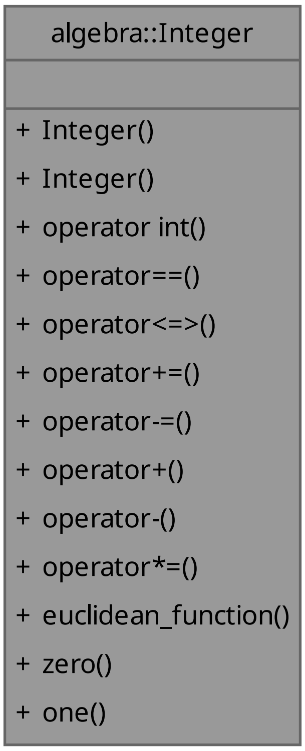
- mc-homology/core/include/core/homology_printing_strategy.h
- mc-homology/core/src/homology_printing_strategy.cpp

6.23 algebra::Integer Class Reference

Class of Integers.

```
#include <integer.h>
```

Collaboration diagram for algebra::Integer:



Public Member Functions

- constexpr **Integer** ()=default

  Returns 0.
- constexpr **Integer** (int k) noexcept

  Returns k.
- constexpr **operator int** () noexcept

  Returns the underlying integer.
- constexpr bool **operator==** (Integer const &) const =default

  Compares two integers for equality.
- constexpr std::strong_ordering **operator<=>** (Integer const &) const =default

  Compares two integers and returns their ordering.
- constexpr Integer & **operator+=** (Integer rhs) noexcept

  Adds rhs to itself.
- constexpr Integer & **operator-=** (Integer rhs) noexcept

  Subtracts rhs from itself.
- constexpr Integer **operator+** () const noexcept

  Returns a copy of self.
- constexpr Integer **operator-** () const noexcept

  Returns a negation of self.

- constexpr Integer & **operator*=** (Integer rhs) noexcept

  Multiplies itself by rhs.
- constexpr int euclidean_function () const noexcept

  Euclidean function for integers.

Static Public Member Functions

- static constexpr Integer **zero** () noexcept

  Returns 0.
- static constexpr Integer **one** () noexcept

  Returns 1.

6.23.1 Detailed Description

Class of Integers.

A class implementing integers, while additionally satisfying EuclideanDomain constraint

6.23.2 Member Function Documentation

6.23.2.1 euclidean_function()

```
int algebra::Integer::euclidean_function () const  [inline], [constexpr], [noexcept]
```

Euclidean function for integers.

Euclidean function for integers, in this case the absolute value.

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

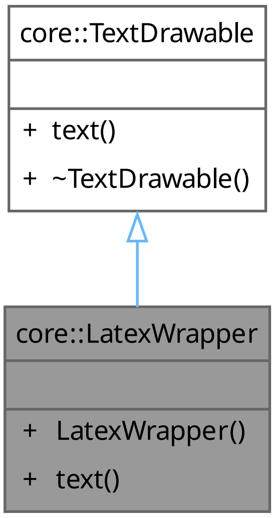
- mc-homology/algebra/include/algebra/integer.h

6.24 core::LatexWrapper Class Reference

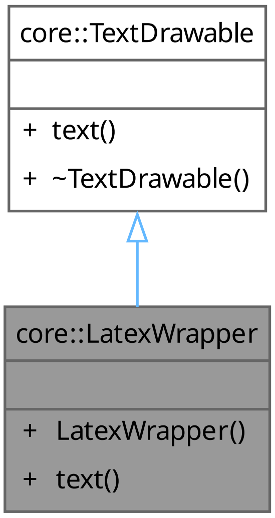
A decorator that wraps a text drawable object in a latex document.

```
#include <latex_wrapper.h>
```

Inheritance diagram for core::LatexWrapper:



Collaboration diagram for core::LatexWrapper:



Public Member Functions

- **LatexWrapper** (std::unique_ptr< TextDrawable > inner, std::string documentclass="article")

Constructs the decorator with the object to wrap.
- std::string text () const override
  Returns the text representation of the object.

Public Member Functions inherited from core::TextDrawable

- virtual ∼**TextDrawable** ()
  Virtual destructor.

6.24.1 Detailed Description

A decorator that wraps a text drawable object in a latex document.

6.24.2 Member Function Documentation

6.24.2.1 text()

```
std::string core::LatexWrapper::text () const  [override], [virtual]
```

Returns the text representation of the object.

Implements core::TextDrawable.

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

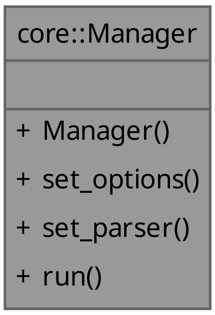
- mc-homology/core/include/core/latex_wrapper.h
- mc-homology/core/src/latex_wrapper.cpp

6.25 core::Manager Class Reference

Main manager for the program.

```
#include <manager.h>
```

Collaboration diagram for core::Manager:



Public Member Functions

- **Manager** (std::unique_ptr< Options > options, std::unique_ptr< MinecraftSavefileParser > parser)

  Construct a new manager with selected parser and options.
- void **set_options** (std::unique_ptr< Options > options)

  Select options.
- void **set_parser** (std::unique_ptr< MinecraftSavefileParser > parser)

  Select parser.
- int run ()

  Start the program.

6.25.1 Detailed Description

Main manager for the program.

6.25.2 Member Function Documentation

6.25.2.1 run()

```
int core::Manager::run ()
```

Start the program.

Returns

program return status

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

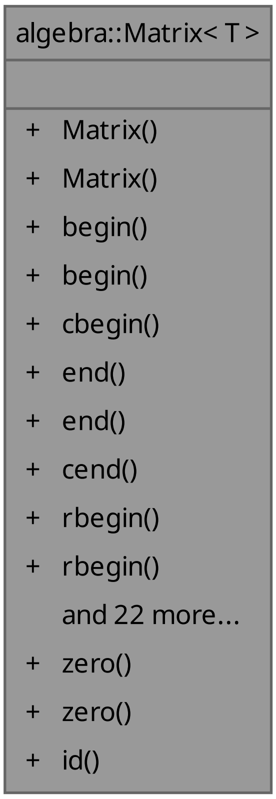
- mc-homology/core/include/core/manager.h
- mc-homology/core/src/manager.cpp

6.26 algebra::Matrix< T > Class Template Reference

A Matrix class.

```
#include <matrix.h>
```

Collaboration diagram for algebra::Matrix< T >:



Public Types

- using **storage_type** = std::vector<T>

  Underlying storage type.
- using **value_type** = std::vector<T>::value_type

  Type of the stored values.
- using **allocator_type** = std::vector<T>::allocator_type

  Type of the used allocator.
- using **size_type** = std::vector<T>::size_type

  Size type used by the underlying storage.
- using **difference_type** = std::vector<T>::difference_type

  Difference type used by the underlying storage.
- using **reference** = std::vector<T>::reference

  Reference type to the stored values.
- using **const_reference** = std::vector<T>::const_reference

  Const reference type to the stored values.
- using **pointer** = std::vector<T>::pointer

  Pointer type to the stored values.
- using **const_pointer** = std::vector<T>::const_pointer

Const pointer type to the stored values.
- using **iterator** = std::vector< T >::iterator

  Iterator over the underlying storage.
- using **const_iterator** = std::vector< T >::const_iterator

  Const iterator over the underlying storage.
- using **reverse_iterator** = std::vector< T >::reverse_iterator

  Reverse iterator over the underlying storage.
- using **const_reverse_iterator** = std::vector< T >::const_reverse_iterator

  Const reverse iterator over the underlying storage.

Public Member Functions

- template< std::ranges::sized_range R >
  constexpr Matrix (R &&data, size_type nrows, size_type ncols)

  Construct a matrix from a range.
- constexpr iterator **begin** () noexcept

  Iterator over the coefficients.
- constexpr const_iterator **begin** () const noexcept

  Iterator over the coefficients.
- constexpr const_iterator **cbegin** () const noexcept

  Const iterator over the coefficients.
- constexpr iterator **end** () noexcept

  Sentinel for the coefficients iterator.
- constexpr const_iterator **end** () const noexcept

  Sentinel for the coefficients iterator.
- constexpr const_iterator **cend** () const noexcept

  Sentinel for the coefficients const iterator.
- constexpr reverse_iterator **rbegin** () noexcept

  Reverse iterator over the coefficients.
- constexpr const_reverse_iterator **rbegin** () const noexcept

  Reverse iterator over the coefficients.
- constexpr const_reverse_iterator **crbegin** () const noexcept

  Const reverse iterator over the coefficients.
- constexpr reverse_iterator **rend** () noexcept

  Sentinel for the coefficients reverse iterator.
- constexpr const_reverse_iterator **rend** () const noexcept

  Sentinel for the coefficients reverse iterator.
- constexpr const_reverse_iterator **crend** () const noexcept

  Sentinel for the coefficients const reverse iterator.
- constexpr bool **empty** () const noexcept

  Test, if the matrix is empty.
- constexpr size_type **size** () const noexcept

  Number of elements in the matrix.
- constexpr void **swap** (Matrix &other) noexcept(m_data.swap(std::declval< std::vector< value_type > >()))

  Specialized swap algorithm for the matrix.
- constexpr size_type **nrows** () const noexcept

  Number of rows.
- constexpr size_type **ncols** () const noexcept

  Number of columns.
- constexpr bool **operator==** (Matrix const &) const =default

Equality comparison for matrices.
- constexpr reference operator[ ] (size_type row, size_type col)

  Access element at row `row` and columns `col`.
- constexpr const_reference operator[ ] (size_type row, size_type col) const

  Access element at row `row` and columns `col`.
- constexpr reference at (size_type row, size_type col)

  Access element at row `row` and columns `col`.
- constexpr const_reference at (size_type row, size_type col) const

  Access element at row `row` and columns `col`.
- constexpr storage_type const & **data** () const noexcept

  Direct to underlying storage.
- constexpr Matrix **transpose** () const

  The transpose of the matrix.
- constexpr Matrix & **operator+=** (Matrix const &rhs)

  Adds rhs to itself.
- constexpr Matrix & **operator-=** (Matrix const &rhs)

  Subtracts rhs from itself.
- constexpr Matrix **operator+** () const

  Returns a copy of itself.
- constexpr Matrix **operator-** () const

  Returns a negation of itself.
- constexpr Matrix operator∗= (Matrix const &rhs)

  Multiplies itself by rhs.
- constexpr bool **is_zero** () const noexcept

  Returns true if matrix is zero, false otherwise.

Static Public Member Functions

- static constexpr Matrix **zero** (size_type n)

  Return a square zero matrix.
- static constexpr Matrix **zero** (size_type n, size_type m)

  Return a rectangle zero matrix.
- static constexpr Matrix **id** (size_type n)

  Return an identity matrix.

6.26.1 Detailed Description

template<class T>
class algebra::Matrix< T >

A Matrix class.

A two-dimensional array representing a mathematical matrix.

6.26.2 Constructor & Destructor Documentation

6.26.2.1 Matrix()

```
template<class T>
template<std::ranges::sized_range R>
algebra::Matrix< T >::Matrix (
             R && data,
             size_type nrows,
             size_type ncols)  [inline], [explicit], [constexpr]
```

Construct a matrix from a range.

Create a matrix with coefficients taken from the range and with the specified number of rows and columns. Size of the range has be equal to the product of nrows and ncols.

Parameters

data	Range with the coefficients
nrows	Number of rows
ncols	Number of colums

6.26.3 Member Function Documentation

6.26.3.1 at() [1/2]

```
template<class T>
reference algebra::Matrix< T >::at (
             size_type row,
             size_type col)  [inline], [constexpr]
```

Access element at row `row` and columns `col`.

Parameters

row	Accessed row
col	Accessed column

6.26.3.2 at() [2/2]

```
template<class T>
const_reference algebra::Matrix< T >::at (
             size_type row,
             size_type col) const  [inline], [constexpr]
```

Access element at row `row` and columns `col`.

Parameters

row	Accessed row

col	Accessed column

6.26.3.3 operator*=()

```
template<class T>
Matrix algebra::Matrix< T >::operator*= (
              Matrix< T > const & rhs)  [inline], [constexpr]
```

Multiplies itself by rhs.

Matrix multiplies itself from the right-hand side by rhs.

6.26.3.4 operator[]() [1/2]

```
template<class T>
reference algebra::Matrix< T >::operator[] (
              size_type row,
              size_type col)  [inline], [constexpr]
```

Access element at row `row` and columns `col`.

Parameters

row	Accessed row
col	Accessed column

6.26.3.5 operator[]() [2/2]

```
template<class T>
const_reference algebra::Matrix< T >::operator[] (
              size_type row,
              size_type col) const  [inline], [constexpr]
```

Access element at row `row` and columns `col`.

Parameters

row	Accessed row
col	Accessed column

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

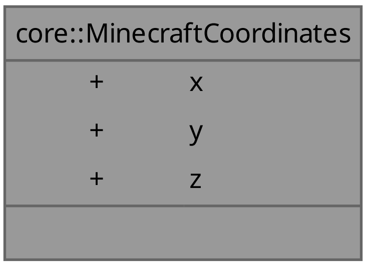
- mc-homology/algebra/include/algebra/matrix.h

6.27 core::MinecraftCoordinates Struct Reference

A struct containing coordinates in a Minecraft world.

```
#include <parser.h>
```

Collaboration diagram for core::MinecraftCoordinates:



Public Attributes

- int **x** = 0

  x coordinate
- int **y** = 0

  y coordinate
- int **z** = 0

  z coordinate

6.27.1 Detailed Description

A struct containing coordinates in a Minecraft world.

Minecraft world coordinates. Note, that y defines height.

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

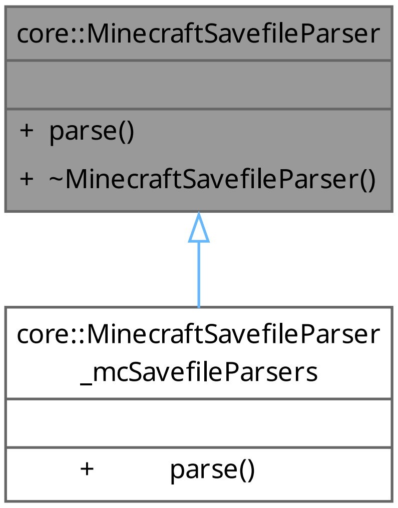
- mc-homology/core/include/core/parser.h

6.28 core::MinecraftSavefileParser Class Reference

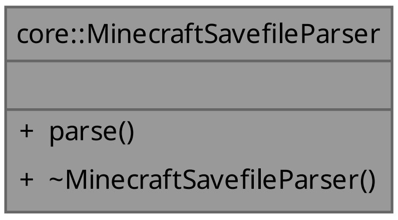
An interface for Minecraft savefile parser.

```
#include <parser.h>
```

Inheritance diagram for core::MinecraftSavefileParser:


core::MinecraftSavefileParser
+ parse()
+ ~MinecraftSavefileParser()
core::MinecraftSavefileParser
_mcSavefileParsers
+ parse()

Collaboration diagram for core::MinecraftSavefileParser:



Public Member Functions

- virtual std::unique_ptr< Complex > parse (std::filesystem::path const &path, MinecraftCoordinates lower_↩ corner, MinecraftCoordinates upper_corner)=0

  Parses a Minecraft savefile.

6.28.1 Detailed Description

An interface for Minecraft savefile parser.

6.28.2 Member Function Documentation

6.28.2.1 parse()

```
virtual std::unique_ptr< Complex > core::MinecraftSavefileParser::parse (
            std::filesystem::path const & path,
            MinecraftCoordinates lower_corner,
            MinecraftCoordinates upper_corner)  [pure virtual]
```

Parses a Minecraft savefile.

Parses a Minecraft savefile within given bounds. lower_corner.x <= upper_corner.x lower_corner.y <= upper_↩
corner.y lower_corner.z <= upper_corner.z

Parameters

path	Path to the save file region directory
lower_corner	Lower bounds on the studied cube
upper_corner	Upper bounds on the studied cube

Implemented in core::MinecraftSavefileParser_mcSavefileParsers.

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

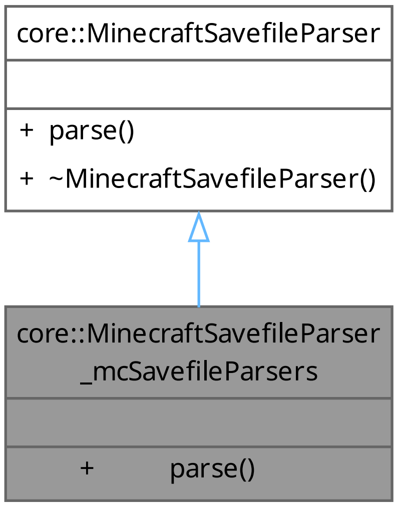
- mc-homology/core/include/core/parser.h
- mc-homology/core/src/parser.cpp

6.29 core::MinecraftSavefileParser_mcSavefileParsers Class Reference

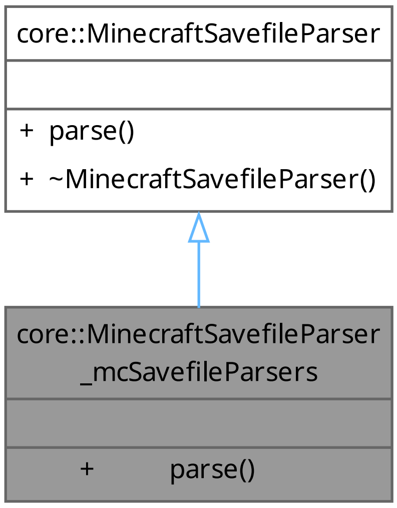
A Minecraft savefile parser based on `https://github.com/TCA166/mcSavefileParsers.git`.

```
#include <parser.h>
```

Inheritance diagram for core::MinecraftSavefileParser_mcSavefileParsers:


core::MinecraftSavefileParser
+ parse()
+ ~MinecraftSavefileParser()
core::MinecraftSavefileParser
_mcSavefileParsers
+ parse()

Collaboration diagram for core::MinecraftSavefileParser_mcSavefileParsers:


core::MinecraftSavefileParser
+ parse()
+ ~MinecraftSavefileParser()
core::MinecraftSavefileParser
_mcSavefileParsers
+ parse()

Public Member Functions

- std::unique_ptr< Complex > parse (std::filesystem::path const &path, MinecraftCoordinates lower_corner, MinecraftCoordinates upper_corner) override

    Parses a Minecraft savefile.

6.29.1 Detailed Description

A Minecraft savefile parser based on `https://github.com/TCA166/mcSavefileParsers.git`.

6.29.2 Member Function Documentation

6.29.2.1 parse()

```
std::unique_ptr< Complex > core::MinecraftSavefileParser_mcSavefileParsers::parse (
            std::filesystem::path const & path,
            MinecraftCoordinates lower_corner,
            MinecraftCoordinates upper_corner)  [override], [virtual]
```

Parses a Minecraft savefile.

Parses a Minecraft savefile within given bounds. lower_corner.x <= upper_corner.x lower_corner.y <= upper_↩ corner.y lower_corner.z <= upper_corner.z

Parameters

path	Path to the save file region directory
lower_corner	Lower bounds on the studied cube
upper_corner	Upper bounds on the studied cube

Implements core::MinecraftSavefileParser.

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

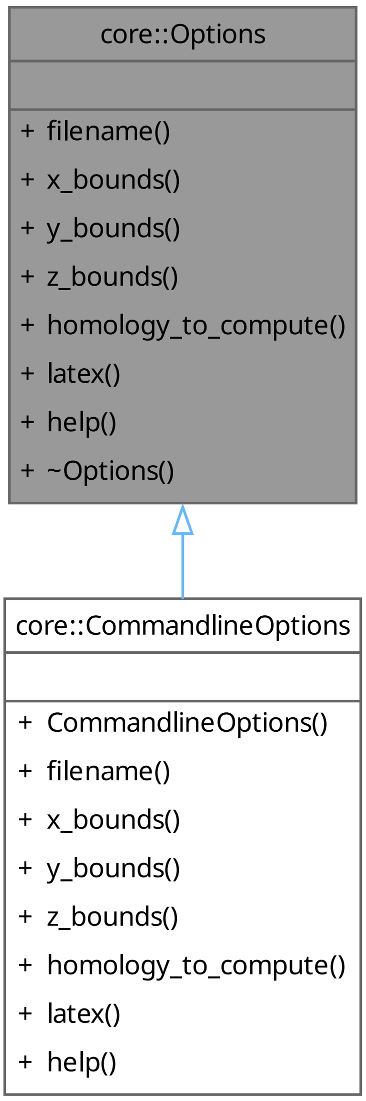
- mc-homology/core/include/core/parser.h
- mc-homology/core/src/parser.cpp

6.30 core::Options Class Reference

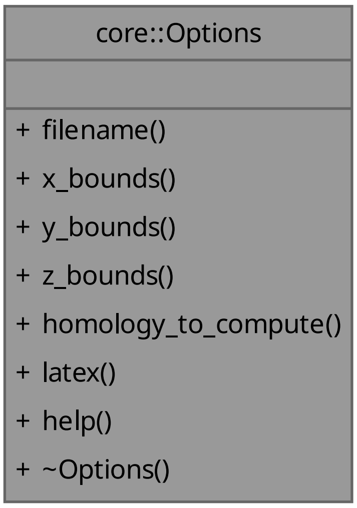
A class for storing user options.

```
#include <options.h>
```

Inheritance diagram for core::Options:



Collaboration diagram for core::Options:



Public Member Functions

- virtual std::filesystem::path filename () const =0

  Path to the region directory.
- virtual std::pair< int, int > x_bounds () const =0

  Bounds on x axis.
- virtual std::pair< int, int > y_bounds () const =0

  Bounds on y axis.
- virtual std::pair< int, int > z_bounds () const =0

  Bounds on y axis.
- virtual HomologyChoice homology_to_compute () const =0

  Type of homology to compute.
- virtual bool latex () const =0

  Whether to print latex output.
- virtual bool help () const =0

  Whether the user requested help.
- virtual ∼**Options** ()

  virtual destructor

6.30.1 Detailed Description

A class for storing user options.

6.30.2 Member Function Documentation

6.30.2.1 filename()

```
virtual std::filesystem::path core::Options::filename () const  [pure virtual]
```

Path to the region directory.

Implemented in core::CommandlineOptions.

6.30.2.2 help()

```
virtual bool core::Options::help () const  [pure virtual]
```

Whether the user requested help.

Implemented in core::CommandlineOptions.

6.30.2.3 homology_to_compute()

```
virtual HomologyChoice core::Options::homology_to_compute () const  [pure virtual]
```

Type of homology to compute.

Implemented in core::CommandlineOptions.

6.30.2.4 latex()

```
virtual bool core::Options::latex () const  [pure virtual]
```

Whether to print latex output.

Implemented in core::CommandlineOptions.

6.30.2.5 x_bounds()

```
virtual std::pair< int, int > core::Options::x_bounds () const  [pure virtual]
```

Bounds on x axis.

Implemented in core::CommandlineOptions.

6.30.2.6 y_bounds()

```
virtual std::pair< int, int > core::Options::y_bounds () const  [pure virtual]
```

Bounds on y axis.

Implemented in core::CommandlineOptions.

6.30.2.7 z_bounds()

```
virtual std::pair< int, int > core::Options::z_bounds () const  [pure virtual]
```

Bounds on y axis.

Implemented in core::CommandlineOptions.

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

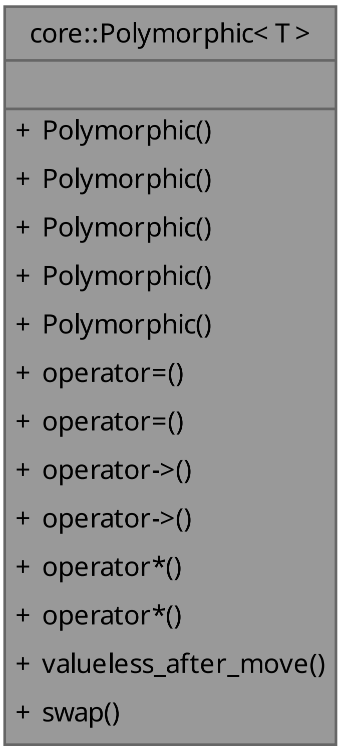
- mc-homology/core/include/core/options.h
- mc-homology/core/src/options.cpp

6.31 core::Polymorphic< T > Class Template Reference

A polymorphic class wrapper with value semantics.

```
#include <polymorphic.h>
```

Collaboration diagram for core::Polymorphic< T >:



Public Member Functions

- constexpr **Polymorphic** ()

  Constructs a default initialized object.
- template<class U = T>
  constexpr **Polymorphic** (U &&u)

  Constructs the object from the object u.
- template<class U, class... Args>
  constexpr **Polymorphic** (std::in_place_type_t< U >, Args &&... args)

  Constructs the object in place.
- constexpr **Polymorphic** (Polymorphic const &other)

  Copy constructor.
- constexpr **Polymorphic** (Polymorphic &&other) noexcept

  Move constructor.
- constexpr Polymorphic & **operator=** (Polymorphic const &other)

  Copy assignment.
- constexpr Polymorphic & **operator=** (Polymorphic &&other) noexcept

  Move assignment.
- constexpr T const ∗ **operator->** () const noexcept

  Arrow operator.
- constexpr T ∗ **operator->** () noexcept

  Arrow operator.
- constexpr T const & **operator**∗ () const noexcept

  Dereference operator.
- constexpr T & **operator**∗ () noexcept

  Dereference operator.
- constexpr bool **valueless_after_move** () const noexcept

  Returns true, if the object has been moved out of.
- constexpr void **swap** (Polymorphic &other) noexcept

  Swaps two objects together.

6.31.1 Detailed Description

template<class T>
class core::Polymorphic< T >

A polymorphic class wrapper with value semantics.

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

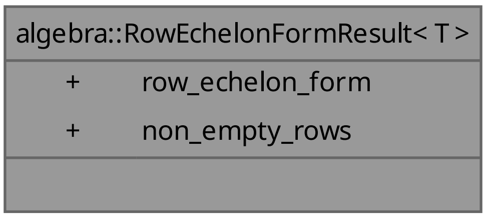
- mc-homology/core/include/core/polymorphic.h

6.32 algebra::RowEchelonFormResult< T > Struct Template Reference

Result struct for the row echelon algorithm.

```
#include <matrix_algorithms.h>
```

Collaboration diagram for algebra::RowEchelonFormResult< T >:



Public Attributes

- Matrix< T > **row_echelon_form** = {}

  Row echelon form of a matrix.
- std::size_t **non_empty_rows** = 0

  Number of non-empty rows of the matrix in row echelon form.

6.32.1 Detailed Description

template<class T>
struct algebra::RowEchelonFormResult< T >

Result struct for the row echelon algorithm.

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

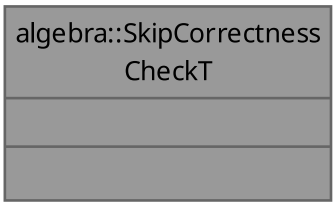
- mc-homology/algebra/include/algebra/matrix_algorithms.h

6.33 algebra::SkipCorrectnessCheckT Struct Reference

Type to mark an overload of a function, that doesn't perform optional correctness checks. Such overloads should only be used after manually ensuring, that other parameters guarantee correct output.

`#include <chain_complex.h>`

Collaboration diagram for algebra::SkipCorrectnessCheckT:



6.33.1 Detailed Description

Type to mark an overload of a function, that doesn't perform optional correctness checks. Such overloads should only be used after manually ensuring, that other parameters guarantee correct output.

Use with caution!

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

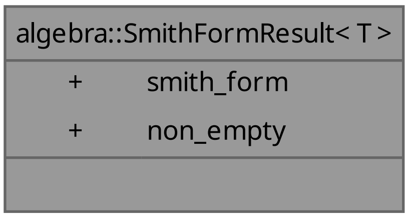
- mc-homology/algebra/include/algebra/chain_complex.h

6.34 algebra::SmithFormResult< T > Struct Template Reference

Result struct for the smith algorithm.

`#include <matrix_algorithms.h>`

Collaboration diagram for algebra::SmithFormResult< T >:



Public Attributes

- Matrix< T > **smith_form** = {}

  Smith form of a matrix.
- std::size_t **non_empty** = 0

  Number of non zero rows or columns.

6.34.1 Detailed Description

template<class T>
struct algebra::SmithFormResult< T >

Result struct for the smith algorithm.

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

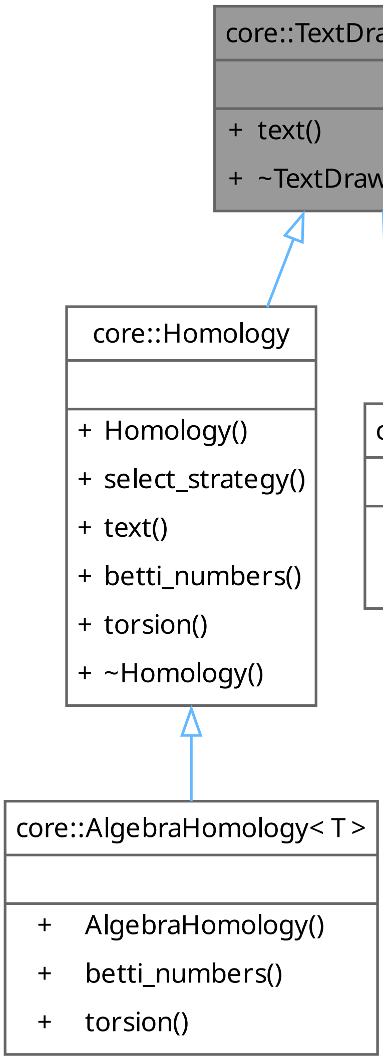
- mc-homology/algebra/include/algebra/matrix_algorithms.h

6.35 core::TextDrawable Class Reference

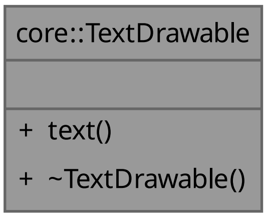
Marks a class text drawable - it can be represented as a string.

```
#include <text_drawable.h>
```

Inheritance diagram for core::TextDrawable:



Collaboration diagram for core::TextDrawable:



Public Member Functions

- virtual std::string text () const =0

    Returns the text representation of the object.
- virtual ∼**TextDrawable** ()

    Virtual destructor.

6.35.1 Detailed Description

Marks a class text drawable - it can be represented as a string.

6.35.2 Member Function Documentation

6.35.2.1 text()

```
virtual std::string core::TextDrawable::text () const  [pure virtual]
```

Returns the text representation of the object.

Implemented in core::Homology, and core::LatexWrapper.

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

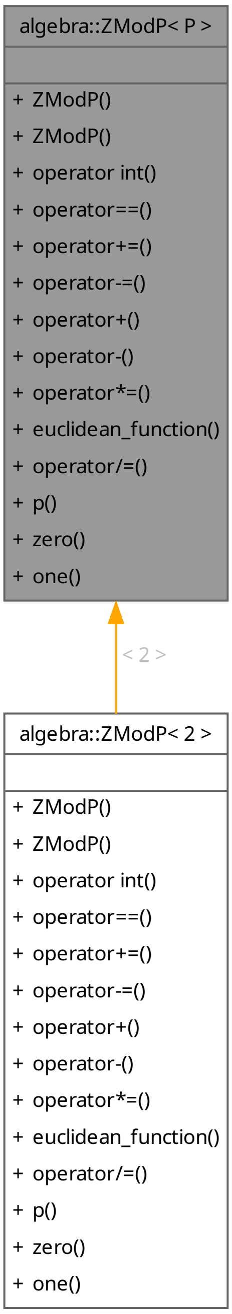
- mc-homology/core/include/core/text_drawable.h
- mc-homology/core/src/text_drawable.cpp

6.36 algebra::ZModP< P > Class Template Reference

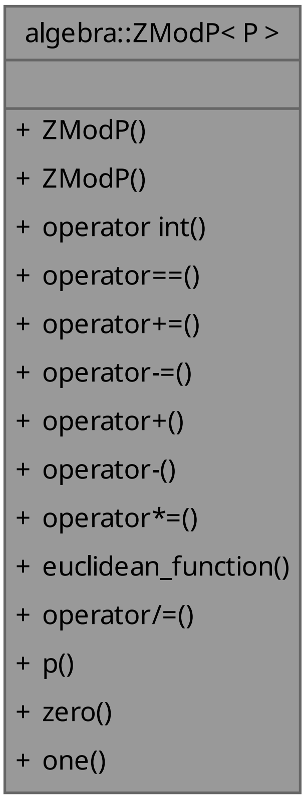
Field of integers modulo P.

```
#include <modulo_fields.h>
```

Inheritance diagram for algebra::ZModP< P >:



Collaboration diagram for algebra::ZModP< P >:



Public Member Functions

- constexpr **ZModP** ()=default

  Returns 0.
- constexpr **ZModP** (int n) noexcept

  Returns n mod P.
- constexpr **operator int** () const noexcept

  Returns the underlying integer.
- constexpr bool **operator==** (ZModP const &) const =default

  Equality comparison.
- constexpr ZModP & **operator+=** (ZModP rhs) noexcept

  Adds rhs to itself.
- constexpr ZModP & **operator-=** (ZModP rhs) noexcept

  Subtracts rhs from itself.
- constexpr ZModP **operator+** () const noexcept

  Returns a copy of itself.
- constexpr ZModP **operator-** () const noexcept

  Returns a negation of itself.
- constexpr ZModP & **operator∗=** (ZModP rhs) noexcept

Multiplies itself by rhs.
- constexpr int euclidean_function () const noexcept

  Euclidean function for a field.
- constexpr ZModP & **operator/=** (ZModP rhs)

  Divides itself by rhs.

Static Public Member Functions

- static constexpr int **p** () noexcept

  Returns the modulus P.
- static constexpr ZModP **zero** () noexcept

  Returns 0.
- static constexpr ZModP **one** () noexcept

  Returns 1.

6.36.1 Detailed Description

template<int P>
class algebra::ZModP< P >

Field of integers modulo P.

A class modeling fields of integers modulo P, where P is a prime number.

Parameters

P	A prime number

6.36.2 Member Function Documentation

6.36.2.1 euclidean_function()

```
template<int P>
int algebra::ZModP< P >::euclidean_function () const  [inline], [constexpr], [noexcept]
```

Euclidean function for a field.

Euclidean function for fields is constantly equal to 1.

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

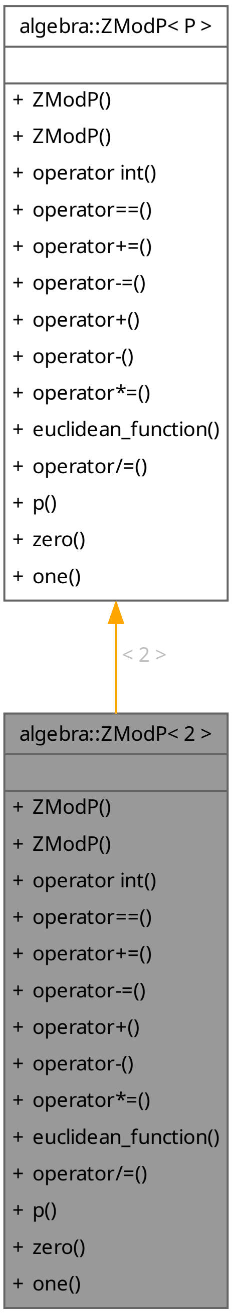
- mc-homology/algebra/include/algebra/modulo_fields.h

6.37 algebra::ZModP< 2 > Class Reference

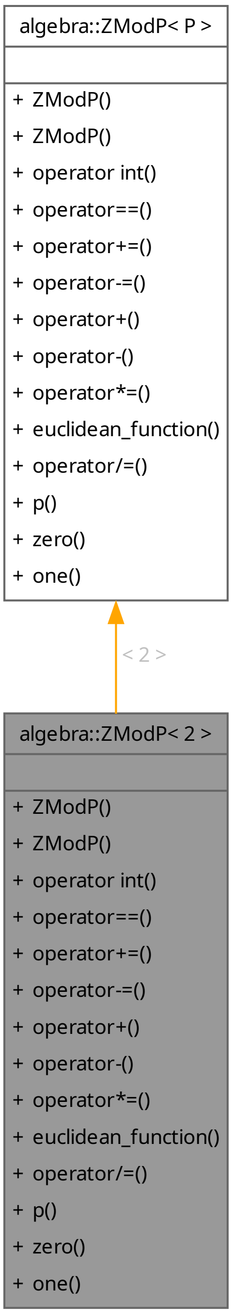
Template specialization for P == 2.

```
#include <z2_field.h>
```

Inheritance diagram for algebra::ZModP< 2 >:



Collaboration diagram for algebra::ZModP< 2 >:



Public Member Functions

- constexpr **ZModP** ()=default

    Returns 0.
- constexpr **ZModP** (int n) noexcept

    Retur>ns n mod P.
- constexpr **operator int** () const noexcept

Returns the underlying integer.
- constexpr bool **operator==** (ZModP const &) const =default

  Equality comparison.
- constexpr ZModP & **operator+=** (ZModP rhs) noexcept

  Adds rhs to itself.
- constexpr ZModP & **operator-=** (ZModP rhs) noexcept

  Subtracts rhs from itself.
- constexpr ZModP **operator+** () const noexcept

  Returns a copy of itself.
- constexpr ZModP **operator-** () const noexcept

  Returns a negation of itself.
- constexpr ZModP & **operator*=** (ZModP rhs) noexcept

  Multiplies itself by rhs.
- constexpr int euclidean_function () const noexcept

  Euclidean function for fields.
- constexpr ZModP & **operator/=** (ZModP rhs)

  Divides itself by rhs.

Static Public Member Functions

- static constexpr int **p** () noexcept

  Returns the modulus P.
- static constexpr ZModP **zero** () noexcept

  Returns 0.
- static constexpr ZModP **one** () noexcept

  Returns 1.

6.37.1 Detailed Description

Template specialization for P == 2.

6.37.2 Member Function Documentation

6.37.2.1 euclidean_function()

```
int algebra::ZModP< 2 >::euclidean_function () const  [inline], [constexpr], [noexcept]
```

Euclidean function for fields.

Euclidean function for fields is constantly equal to 1.

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

- mc-homology/algebra/include/algebra/z2_field.h

Chapter 7

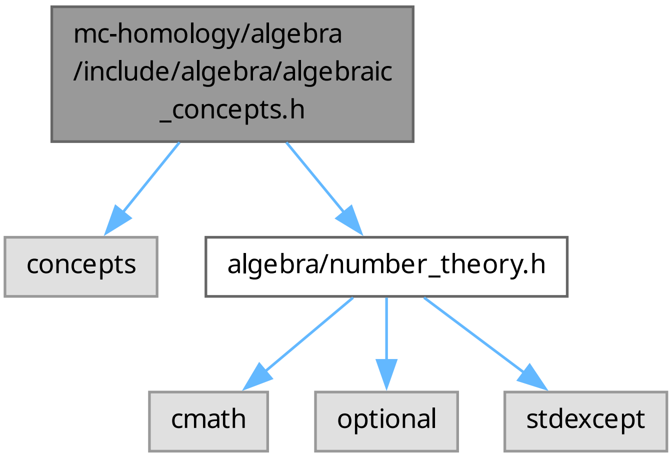
File Documentation

7.1 mc-homology/algebra/include/algebra/algebraic_concepts.h File Reference

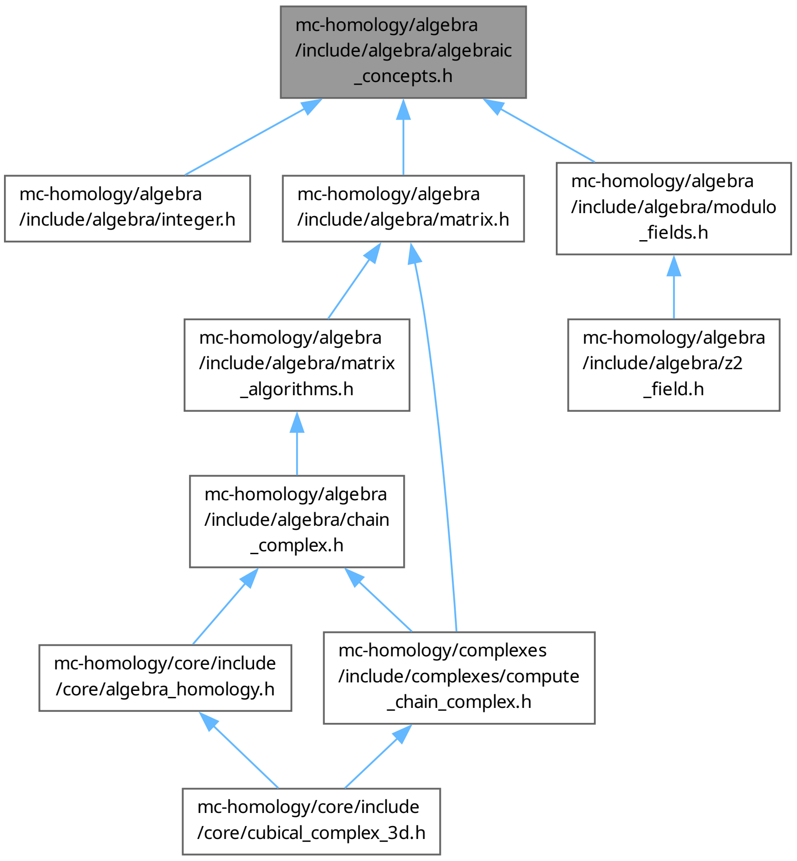
A file defining multiple concepts for algebraic structures.

```
#include <concepts>
#include "algebra/number_theory.h"
```

Include dependency graph for algebraic_concepts.h:



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


mc-homology/algebra /include/algebra/algebraic _concepts.h
mc-homology/algebra /include/algebra/integer.h
mc-homology/algebra /include/algebra/matrix.h
mc-homology/algebra /include/algebra/modulo _fields.h
mc-homology/algebra /include/algebra/matrix _algorithms.h
mc-homology/algebra /include/algebra/z2 _field.h
mc-homology/algebra /include/algebra/chain _complex.h
mc-homology/core/include /core/algebra_homology.h
mc-homology/complexes /include/complexes/compute _chain_complex.h
mc-homology/core/include /core/cubical_complex_3d.h

Concepts

- concept algebra::Commutative

  An abelian structure concept.
- concept algebra::Group

  A group concept.
- concept algebra::AdditiveGroup

  An additive group concept.
- concept algebra::Ring

  A ring concept.
- concept algebra::CommutativeRing

  An commutative ring concept.
- concept algebra::EuclideanDomain

  An euclidean domain concept.
- concept algebra::Field

  A field concept.

Variables

- template<class T>
  constexpr bool **algebra::is_commutative_v** = false

  Helper variable template that can be used as a mark that class' main operation (addition or multiplication) is abelian.

7.1.1 Detailed Description

A file defining multiple concepts for algebraic structures.

7.2 algebraic_concepts.h

Go to the documentation of this file.

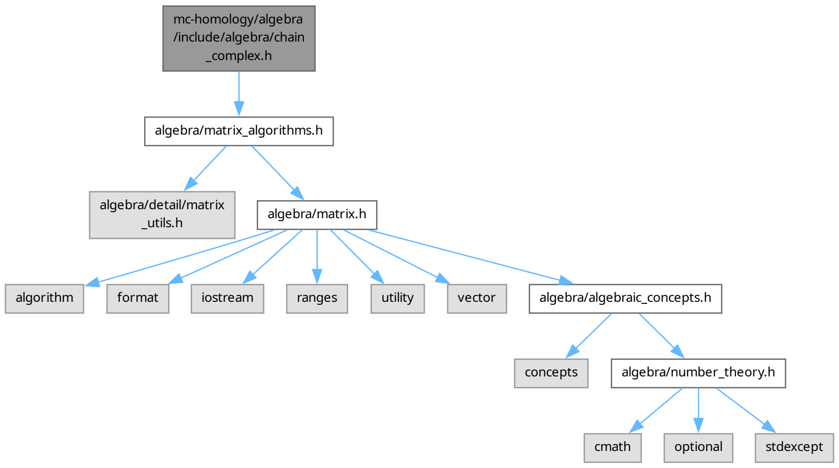
```
#pragma once

#include <concepts>

#include "algebra/number_theory.h"

namespace algebra {

template<class T>
constexpr inline bool is_commutative_v = false;

template<class T>
concept Commutative = is_commutative_v<T>;

template<class T>
concept Group =
    std::regular<T> && std::default_initializable<T> && requires(T x, T y) {
        { x * y } -> std::convertible_to<T>;
        { x / y } -> std::convertible_to<T>;
        { x *= y } -> std::same_as<T&>;
        { x /= y } -> std::same_as<T&>;
        { T::one() } -> std::convertible_to<T>;
    };

template<class T>
concept AdditiveGroup = std::regular<T> && std::default_initializable<T>
    && Commutative<T> && requires(T x, T y) {
           { x + y } -> std::convertible_to<T>;
           { x - y } -> std::convertible_to<T>;
           { x += y } -> std::same_as<T&>;
           { x -= y } -> std::same_as<T&>;
           { +x } -> std::convertible_to<T>;
           { -x } -> std::convertible_to<T>;
           { T::zero() } -> std::convertible_to<T>;
       };

template<class T>
concept Ring = AdditiveGroup<T> && requires(T x, T y) {
    { x * y } -> std::convertible_to<T>;
    { x *= y } -> std::same_as<T&>;
    { T::one() } -> std::convertible_to<T>;
};

template<class T>
concept CommutativeRing = Ring<T> && Commutative<T>;

template<class T>
concept EuclideanDomain = CommutativeRing<T> && requires(T a, T b, T x, T y) {
    { divide(x, y) } -> std::same_as<DivResult<T»;
    { x.euclidean_function() } -> std::same_as<int>;
};

template<class T>
concept Field = CommutativeRing<T> && requires(T x, T y) {
    { x / y } -> std::convertible_to<T>;
    { x /= y } -> std::same_as<T&>;
};

} // namespace algebra
```

7.3 mc-homology/algebra/include/algebra/chain_complex.h File Reference

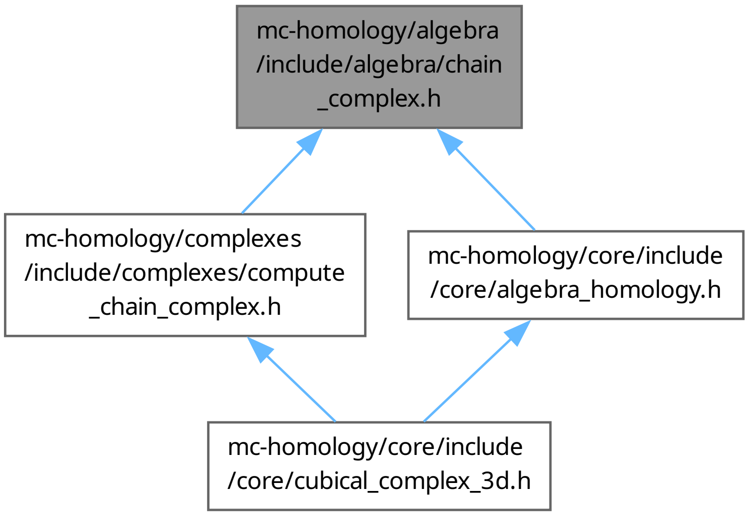
A file containing chain complex and homology implementations.

`#include "algebra/matrix_algorithms.h"`

Include dependency graph for chain_complex.h:


mc-homology/algebra/include/algebra/chain_complex.h
algebra/matrix_algorithms.h
algebra/detail/matrix_utils.h
algebra/matrix.h
algorithm
format
iostream
ranges
utility
vector
algebra/algebraic_concepts.h
concepts
algebra/number_theory.h
cmath
optional
stdexcept

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


mc-homology/algebra/include/algebra/chain_complex.h
mc-homology/complexes/include/complexes/compute_chain_complex.h
mc-homology/core/include/core/algebra_homology.h
mc-homology/core/include/core/cubical_complex_3d.h

Classes

- struct algebra::SkipCorrectnessCheckT

  Type to mark an overload of a function, that doesn't perform optional correctness checks. Such overloads should only be used after manually ensuring, that other parameters guarantee correct output.
- class algebra::ChainComplex< T >

  Class representing a chain complex with coefficients `T`.
- struct algebra::Homology< T >

  Homology of a chain complex.

Functions

- template<EuclideanDomain T>
  Homology< T > **algebra::homology** (ChainComplex< T > const &chain_complex)

  Computes homology of a chain complex with coefficients from an euclidean domain.
- template<Field T>
  Homology< T > **algebra::homology** (ChainComplex< T > const &chain_complex)

  Computes homology of a chain complex with coefficients from a field.
- template<std::ranges::sized_range R>
  algebra::ChainComplex (SkipCorrectnessCheckT, R &&) -> ChainComplex< typename std::ranges::↩
  range_value_t< R >::value_type >

  Deduction guide for the ChainComplex.
- template<std::ranges::sized_range R>
  algebra::ChainComplex (R &&) -> ChainComplex< typename std::ranges::range_value_t< R >::value_↩
  type >

  Deduction guide for the ChainComplex.

Variables

- constexpr SkipCorrectnessCheckT **algebra::skip_correctness_check** {}

  Helper value of that SkipCorrectnessCheckT.

7.3.1 Detailed Description

A file containing chain complex and homology implementations.

7.4 chain_complex.h

Go to the documentation of this file.

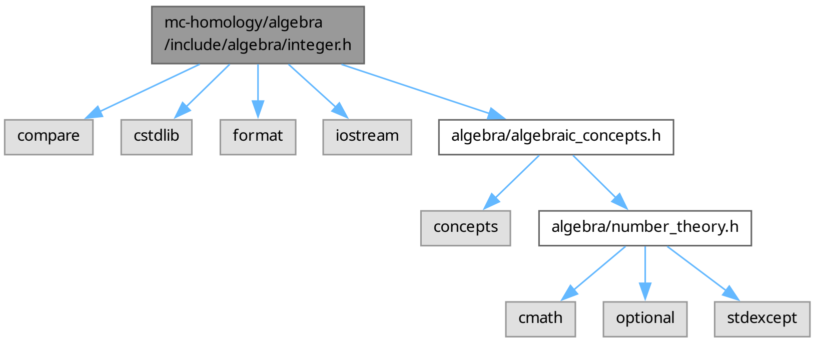
```


#pragma once

#include "algebra/matrix_algorithms.h"

namespace algebra {

struct SkipCorrectnessCheckT {};

constexpr inline SkipCorrectnessCheckT skip_correctness_check {};

template<class T>
class ChainComplex {
public:
    constexpr ChainComplex() = default;

    template<std::ranges::sized_range R>
        requires std::convertible_to<std::ranges::range_value_t<R>, Matrix<T>>
    constexpr ChainComplex(R&& boundaries) :
        ChainComplex(skip_correctness_check, std::forward<R>(boundaries)) {
        if (!check_boundary_correctness()) {
            throw std::domain_error(
                "The boundary matrices do not satisfy chain complex condition"
            );
        }
    }

    template<std::ranges::sized_range R>
        requires std::convertible_to<std::ranges::range_value_t<R>, Matrix<T>>
    constexpr ChainComplex(SkipCorrectnessCheckT, R&& boundaries) :
        m_boundaries(
            std::ranges::to<std::vector<Matrix<T»>(std::forward<R>(boundaries))
        ) {}

    constexpr bool check_boundary_correctness() const {
        if (m_boundaries.size() < 2) {
            return true;
        }
        namespace rs = std::ranges;
        namespace vs = std::views;
        auto b_n_plus_1_view = m_boundaries | vs::drop(1);
        auto b_n_view = m_boundaries | vs::take(m_boundaries.size() - 1);
        return rs::all_of(
            vs::zip(b_n_plus_1_view, b_n_view),
            [](auto const& boundaries) {
                try {
                    auto const& [b_n_plus_1, b_n] = boundaries;
                    return (b_n * b_n_plus_1).is_zero();
                } catch (std::domain_error&) {
                    return false;
                }
            }
        );
    }

    constexpr std::size_t dimension() const noexcept {
        return m_boundaries.size();
    }

    constexpr Matrix<T> const& boundary(std::size_t dim) const {
        return m_boundaries.at(dim);
    }

    constexpr std::vector<Matrix<T>» const& boundaries() const noexcept {
        return m_boundaries;
    }

private:
    std::vector<Matrix<T>» m_boundaries;
};

template<class T>
struct Homology {
    std::vector<std::size_t> betti_numbers {};

    std::vector<std::vector<T>» torsion {};
};

template<EuclideanDomain T>
Homology<T> homology(ChainComplex<T> const& chain_complex) {
    namespace rs = std::ranges;
    namespace vs = std::views;
    auto const& boundaries = chain_complex.boundaries();
    Homology<T> homology;
    homology.betti_numbers.resize(boundaries.size());
    homology.torsion.resize(boundaries.size());
    std::size_t prev_smith_units_count = 0;
    std::vector<T> prev_smith_diagonal_without_units {};
    for (std::size_t k = boundaries.size(); k > 0; --k) {
        auto const n = k - 1;
        auto const& boundary = boundaries[n];
        auto [smith, rank] = smith_form(boundary);
        auto nullity = boundary.ncols() - rank;
        auto smith_diagonal_without_units = rs::to<std::vector>(
            vs::iota(0u, rank)
            | vs::transform([&smith](std::size_t n) { return smith[n, n]; })
            | vs::drop_while([](T const& x) {
                  return x.euclidean_function() == 1;
              })
        );
        auto smith_units = rank - smith_diagonal_without_units.size();
        homology.betti_numbers[n] = nullity - prev_smith_units_count
            - prev_smith_diagonal_without_units.size();
        homology.torsion[n] = std::move(prev_smith_diagonal_without_units);
        prev_smith_units_count = smith_units;
        prev_smith_diagonal_without_units =
            std::move(smith_diagonal_without_units);
    }
    return homology;
}

template<Field T>
Homology<T> homology(ChainComplex<T> const& chain_complex) {
    auto const& boundaries = chain_complex.boundaries();
    Homology<T> homology;
    homology.betti_numbers.resize(boundaries.size());
    homology.torsion.resize(boundaries.size());
    std::size_t prev_rank = 0;
    for (std::size_t k = boundaries.size(); k > 0; --k) {
        auto const n = k - 1;
        auto const& boundary = boundaries[n];
        auto [_, rank] = row_echelon_form(boundary);
        auto nullity = boundary.ncols() - rank;
        homology.betti_numbers[n] = nullity - prev_rank;
        prev_rank = rank;
    }
    return homology;
}

template<std::ranges::sized_range R>
ChainComplex(SkipCorrectnessCheckT, R&&)
    -> ChainComplex<typename std::ranges::range_value_t<R>::value_type>;

template<std::ranges::sized_range R>
ChainComplex(R&&)
    -> ChainComplex<typename std::ranges::range_value_t<R>::value_type>;

} // namespace algebra
```

7.5 mc-homology/algebra/include/algebra/integer.h File Reference

A file containing Integer implementation satisfying algebraic concepts.

```
#include <compare>
#include <cstdlib>
#include <format>
#include <iostream>
#include "algebra/algebraic_concepts.h"
```

Include dependency graph for integer.h:



Classes

- class algebra::Integer

    Class of Integers.

- struct std::formatter< algebra::Integer >

    Formatter for Integer type.

Functions

- constexpr Integer **algebra::operator+** (Integer lhs, Integer rhs) noexcept

  Adds two integers.
- constexpr Integer **algebra::operator-** (Integer lhs, Integer rhs) noexcept

  Subtracts two integers.
- constexpr Integer **algebra::operator*** (Integer lhs, Integer rhs) noexcept

  Multiplies two integers.
- constexpr Integer **algebra::abs** (Integer k) noexcept

  Returns absolute value.
- constexpr DivResult< Integer > algebra::divide (Integer a, Integer b)

  Returns the result of integer division for Integers a and b.
- constexpr Integer algebra::modulo (Integer a, Integer n)

  Returns the remainder of division of a by n.
- std::ostream & **algebra::operator<<** (std::ostream &output, Integer k)

  Outputs an integer to a stream.

Variables

- template<> constexpr bool **algebra::is_commutative_v< Integer >** = true

  A marker that multiplying the integers is commutative.

7.5.1 Detailed Description

A file containing Integer implementation satisfying algebraic concepts.

7.5.2 Function Documentation

7.5.2.1 divide()

```
DivResult< Integer > algebra::divide (
            Integer a,
            Integer b) [constexpr]
```

Returns the result of integer division for Integers a and b.

Returns unique `q` and `r` satisfying

1. `a == q * b + r`
2. `0 <= r < b.euclidean_function()`

7.5.2.2 modulo()

```
Integer algebra::modulo (
            Integer a,
            Integer n) [constexpr]
```

Returns the remainder of division of a by n.

Returns an integer r, 0 <= r < n satisfying `a == q * n + r`

7.6 integer.h

Go to the documentation of this file.

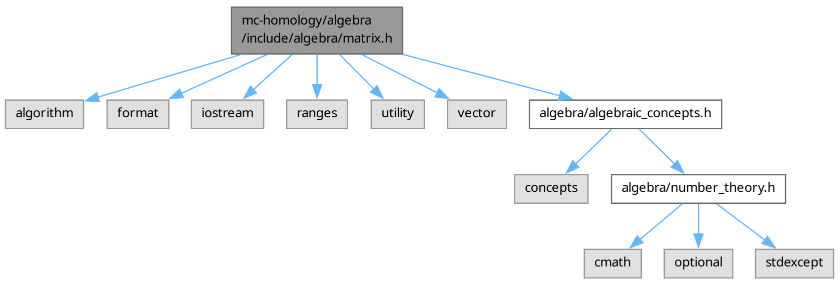
```
#pragma once

#include <compare>
#include <cstdlib>
#include <format>
#include <iostream>

#include "algebra/algebraic_concepts.h"

namespace algebra {

class Integer {
public:
    constexpr Integer() = default;

    constexpr Integer(int k) noexcept : m_inner_representation(k) {}

    constexpr static Integer zero() noexcept {
        return Integer(0);
    }

    constexpr static Integer one() noexcept {
        return Integer(1);
    }

    constexpr explicit operator int() noexcept {
        return m_inner_representation;
    }

    constexpr bool operator==(Integer const&) const = default;

    constexpr std::strong_ordering operator<=>(Integer const&) const = default;

    constexpr Integer& operator+=(Integer rhs) noexcept {
        m_inner_representation += rhs.m_inner_representation;
        return *this;
    }

    constexpr Integer& operator-=(Integer rhs) noexcept {
        m_inner_representation -= rhs.m_inner_representation;
        return *this;
    }

    constexpr Integer operator+() const noexcept {
        return *this;
    }

    constexpr Integer operator-() const noexcept {
        return Integer(-m_inner_representation);
    }

    constexpr Integer& operator*=(Integer rhs) noexcept {
        m_inner_representation *= rhs.m_inner_representation;
        return *this;
    }

    constexpr int euclidean_function() const noexcept {
        using std::abs;
        return abs(m_inner_representation);
    }

private:
    int m_inner_representation = 0;
};

constexpr Integer operator+(Integer lhs, Integer rhs) noexcept {
    return lhs += rhs;
}

constexpr Integer operator-(Integer lhs, Integer rhs) noexcept {
    return lhs -= rhs;
}

constexpr Integer operator*(Integer lhs, Integer rhs) noexcept {
    return lhs *= rhs;
}

constexpr Integer abs(Integer k) noexcept {
    return std::abs(static_cast<int>(k));
}

constexpr DivResult<Integer> divide(Integer a, Integer b) {
    auto div_result = divide(static_cast<int>(a), static_cast<int>(b));
    return DivResult<Integer> {
        .quotient = div_result.quotient,
        .remainder = div_result.remainder
    };
}

constexpr Integer modulo(Integer a, Integer n) {
    return divide(a, n).remainder;
}

template<>
constexpr inline bool is_commutative_v<Integer> = true;

inline std::ostream& operator«(std::ostream& output, Integer k) {
    return output « static_cast<int>(k);
}

} // namespace algebra

template<>
struct std::formatter<algebra::Integer>: public std::formatter<int> {
    template<class FmtContext>
    FmtContext::iterator format(algebra::Integer k, FmtContext& ctx) const {
        return std::formatter<int>::format(static_cast<int>(k), ctx);
    }
};
```

7.7 mc-homology/algebra/include/algebra/matrix.h File Reference

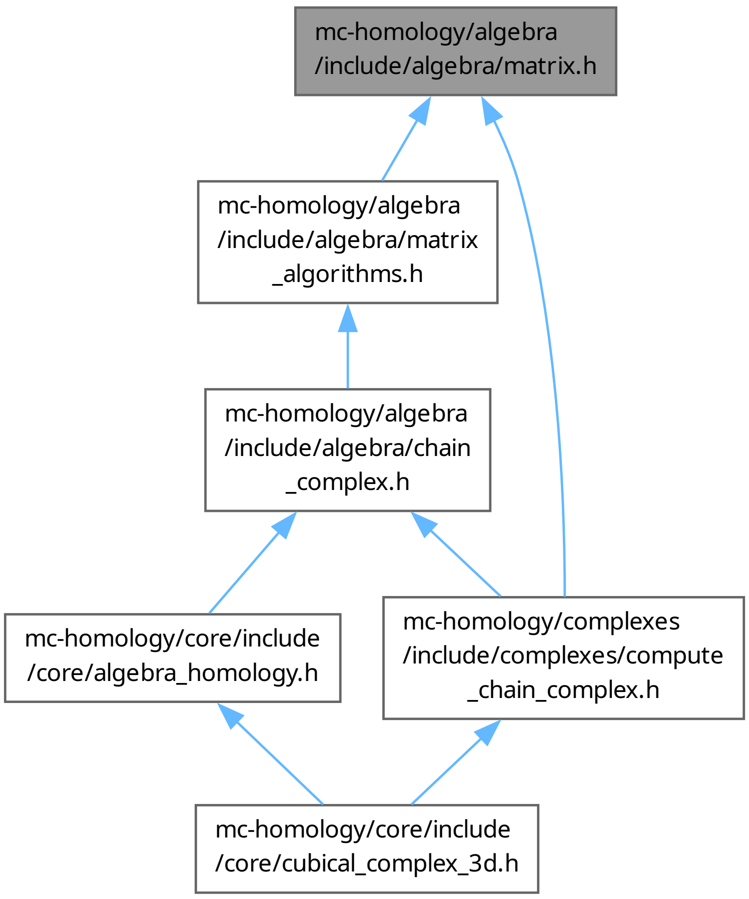
A file containing matrix implementation.

```
#include <algorithm>
#include <format>
#include <iostream>
#include <ranges>
#include <utility>
#include <vector>
#include "algebra/algebraic_concepts.h"
```

Include dependency graph for matrix.h:



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


mc-homology/algebra /include/algebra/matrix.h
mc-homology/algebra /include/algebra/matrix _algorithms.h
mc-homology/algebra /include/algebra/chain _complex.h
mc-homology/core/include /core/algebra_homology.h
mc-homology/complexes /include/complexes/compute _chain_complex.h
mc-homology/core/include /core/cubical_complex_3d.h

Classes

- class algebra::Matrix< T >

    A Matrix class.
- struct std::formatter< algebra::Matrix< T > >

    Formatter for the `Matrix` type.

Functions

- template<AdditiveGroup T>
  constexpr Matrix< T > **algebra::operator+** (Matrix< T > lhs, Matrix< T > const &rhs)

    Adds two matrices.
- template<AdditiveGroup T>
  constexpr Matrix< T > **algebra::operator-** (Matrix< T > lhs, Matrix< T > const &rhs)

    Subtracts two matrices.
- template<Ring T>
  constexpr Matrix< T > algebra::operator∗ (Matrix< T > const &lhs, Matrix< T > const &rhs)

Multiplies two matrices.

- template<std::ranges::sized_range R>
  algebra::Matrix (R &&, std::size_t, std::size_t) -> Matrix< std::ranges::range_value_t< R > >

  Deduce matrix value type from the submitted range.
- template<class T>
  std::ostream & **algebra::operator**<< (std::ostream &output, Matrix< T > const &matrix)

  Outputs a matrix to a stream.

7.7.1 Detailed Description

A file containing matrix implementation.

7.7.2 Function Documentation

7.7.2.1 operator*()

```
template<Ring T>
Matrix< T > algebra::operator* (
            Matrix< T > const & lhs,
            Matrix< T > const & rhs)  [constexpr]
```

Multiplies two matrices.

Multiplies two matrices using the usual matrix multiplication.

7.8 matrix.h

Go to the documentation of this file.

```


#pragma once

#include <algorithm>
#include <format>
#include <iostream>
#include <ranges>
#include <utility>
#include <vector>

#include "algebra/algebraic_concepts.h"

namespace algebra {

template<class T>
class Matrix {
public:
    using storage_type = std::vector<T>;
    using value_type = std::vector<T>::value_type;
    using allocator_type = std::vector<T>::allocator_type;
    using size_type = std::vector<T>::size_type;
    using difference_type = std::vector<T>::difference_type;
    using reference = std::vector<T>::reference;
    using const_reference = std::vector<T>::const_reference;
    using pointer = std::vector<T>::pointer;
    using const_pointer = std::vector<T>::const_pointer;
    using iterator = std::vector<T>::iterator;
    using const_iterator = std::vector<T>::const_iterator;
    using reverse_iterator = std::vector<T>::reverse_iterator;
    using const_reverse_iterator = std::vector<T>::const_reverse_iterator;

    constexpr Matrix() = default;

    template<std::ranges::sized_range R>
        requires std::convertible_to<std::ranges::range_value_t<R>, T>
    constexpr explicit Matrix(R&& data, size_type nrows, size_type ncols) :
        m_data {},
        m_nrows {nrows},
        m_ncols {ncols} {
        if (std::ranges::size(data) != nrows * ncols) [[unlikely]] {
            throw std::domain_error(
                "Size of the array is not equal to the number"
                " of rows times the number of columns"
            );
        }
        m_data = std::ranges::to<std::vector<T»(std::forward<R>(data));
    }

    constexpr iterator begin() noexcept {
        return m_data.begin();
    }

    constexpr const_iterator begin() const noexcept {
        return m_data.begin();
    }

    constexpr const_iterator cbegin() const noexcept {
        return m_data.cbegin();
    }

    constexpr iterator end() noexcept {
        return m_data.end();
    }

    constexpr const_iterator end() const noexcept {
        return m_data.end();
    }

    constexpr const_iterator cend() const noexcept {
        return m_data.cend();
    }

    constexpr reverse_iterator rbegin() noexcept {
        return m_data.rbegin();
    }

    constexpr const_reverse_iterator rbegin() const noexcept {
        return m_data.rbegin();
    }

    constexpr const_reverse_iterator crbegin() const noexcept {
        return m_data.crbegin();
    }

    constexpr reverse_iterator rend() noexcept {
        return m_data.rend();
    }

    constexpr const_reverse_iterator rend() const noexcept {
        return m_data.rend();
    }

    constexpr const_reverse_iterator crend() const noexcept {
        return m_data.crend();
    }

    constexpr bool empty() const noexcept {
        return m_data.empty();
    }

    constexpr size_type size() const noexcept {
        return m_data.size();
    }

    constexpr void swap(Matrix& other) noexcept(
        m_data.swap(std::declval<std::vector<value_type»())
    ) {
        namespace rs = std::ranges;
        m_data.swap(other.m_data);
        rs::swap(m_nrows, other.m_nrows);
        rs::swap(m_ncols, other.m_ncols);
    }

    constexpr size_type nrows() const noexcept {
        return m_nrows;
    }

    constexpr size_type ncols() const noexcept {
        return m_ncols;
    }
    constexpr bool operator==(Matrix const&) const = default;

    constexpr reference operator[](size_type row, size_type col) {
        if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
            throw std::out_of_range("Indices out of matrix range");
        }
        return m_data[to_underlying_index(row, col)];
    }

    constexpr const_reference operator[](size_type row, size_type col) const {
        if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
            throw std::out_of_range("Indices out of matrix range");
        }
        return m_data[to_underlying_index(row, col)];
    }

    constexpr reference at(size_type row, size_type col) {
        if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
            throw std::out_of_range("Indices out of matrix range");
        }
        return m_data.at(to_underlying_index(row, col));
    }

    constexpr const_reference at(size_type row, size_type col) const {
        if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
            throw std::out_of_range("Indices out of matrix range");
        }
        return m_data.at(to_underlying_index(row, col));
    }

    constexpr storage_type const& data() const noexcept {
        return m_data;
    }

    constexpr Matrix transpose() const {
        auto transposed = zero(m_ncols, m_nrows);
        for (size_type i = 0; i < m_nrows; ++i) {
            for (size_type j = 0; j < m_ncols; ++j) {
                transposed[j, i] = at(i, j);
            }
        }
        return transposed;
    }

    constexpr Matrix& operator+=(Matrix const& rhs)
        requires AdditiveGroup<T>
    {
        if (m_nrows != rhs.m_nrows || m_ncols != rhs.m_ncols) {
            throw std::domain_error("Adding matrices of different dimensions");
        }
        namespace rs = std::ranges;
        rs::transform(m_data, rhs.m_data, rs::begin(m_data), std::plus {});
        return *this;
    }

    constexpr Matrix& operator-=(Matrix const& rhs)
        requires AdditiveGroup<T>
    {
        if (m_nrows != rhs.m_nrows || m_ncols != rhs.m_ncols) {
            throw std::domain_error(
                "Subtracting matrices of different dimensions"
            );
        }
        namespace rs = std::ranges;
        rs::transform(m_data, rhs.m_data, rs::begin(m_data), std::minus {});
    }

    constexpr Matrix operator+() const
        requires AdditiveGroup<T>
    {
        return *this;
    }

    constexpr Matrix operator-() const
        requires AdditiveGroup<T>
    {
        namespace rs = std::ranges;
        std::vector<T> negated;
        negated.reserve(m_data.capacity());
        rs::transform(m_data, rs::begin(negated), std::negate {});
        return Matrix(negated, m_nrows, m_ncols);
    }

    constexpr Matrix operator*=(Matrix const& rhs)
        requires Ring<T>
    {
        *this = *this * rhs;
    }

    constexpr static Matrix zero(size_type n)
        requires AdditiveGroup<T>
    {
        std::vector data(n * n, value_type::zero());
        return Matrix(std::move(data), n, n);
    }

    constexpr static Matrix zero(size_type n, size_type m)
        requires AdditiveGroup<T>
    {
        std::vector data(n * m, value_type::zero());
        return Matrix(std::move(data), n, m);
    }

    constexpr bool is_zero() const noexcept {
        return std::ranges::all_of(m_data, [](T const& x) {
            return x == T::zero();
        });
    }

    constexpr static Matrix id(size_type n)
        requires CommutativeRing<T>
    {
        auto identity = zero(n);
        for (size_type i = 0; i < n; ++i) {
            identity[i, i] = value_type::one();
        }
        return identity;
    }

private:
    constexpr size_type
    to_underlying_index(size_type row, size_type col) const noexcept {
        return row * m_ncols + col;
    }

    storage_type m_data;
    size_type m_nrows;
    size_type m_ncols;
};

template<AdditiveGroup T>
constexpr Matrix<T> operator+(Matrix<T> lhs, Matrix<T> const& rhs) {
    return lhs += rhs;
}

template<AdditiveGroup T>
constexpr Matrix<T> operator-(Matrix<T> lhs, Matrix<T> const& rhs) {
    return lhs -= rhs;
}

template<Ring T>
constexpr Matrix<T> operator*(Matrix<T> const& lhs, Matrix<T> const& rhs) {
    using Matrix = Matrix<T>;
    using size_type = Matrix::size_type;
    if (lhs.ncols() != rhs.nrows()) {
        throw std::domain_error(
            "The number of columns of lhs is different "
            "than the number of rows of rhs"
        );
    }
    auto const inner_dim = lhs.ncols();
    Matrix product = Matrix::zero(lhs.nrows(), rhs.ncols());
    for (size_type i = 0; i < product.nrows(); ++i) {
        for (size_type j = 0; j < product.ncols(); ++j) {
            for (size_type k = 0; k < inner_dim; ++k) {
                product[i, j] += lhs[i, k] * rhs[k, j];
            }
        }
    }
    return product;
}

template<std::ranges::sized_range R>
Matrix(R&&, std::size_t, std::size_t) -> Matrix<std::ranges::range_value_t<R>>;

template<class T>
std::ostream& operator«(std::ostream& output, Matrix<T> const& matrix) {
    return output « std::format("{}", matrix);
}

} // namespace algebra

template<class T>
```

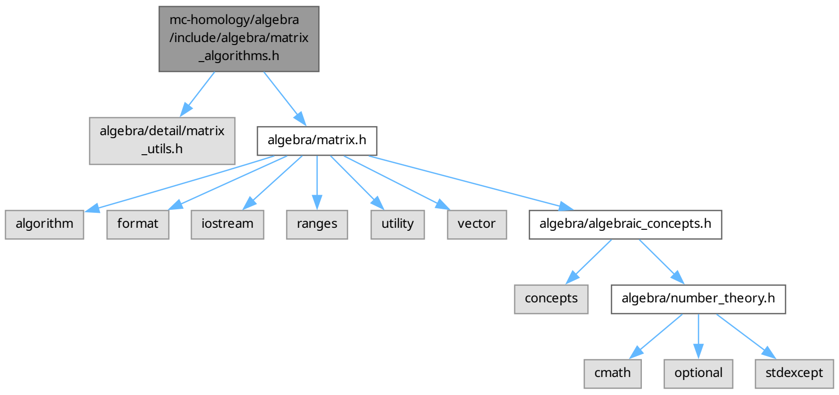
```
    requires std::formattable<T, char>
struct std::formatter<algebra::Matrix<T» {
    template<class ParseContext>
    constexpr ParseContext::iterator parse(ParseContext& ctx) {
        auto it = ctx.begin();
        if (it == ctx.end() || *it == '}') {
            return it;
        } else if (*it == '-') {
            multi_line = false;
            ++it;
        } else if (*it == '#') {
            multi_line = true;
            ++it;
        }

        if (it == ctx.end() || *it == '}') {
            return it;
        } else if (*it == ':') {
            ctx.advance_to(++it);
            return coefficient_formatter.parse(ctx);
        } else {
            throw std::format_error("Invalid formatting option for Matrix");
        }
    }

    template<class FmtContext>
    FmtContext::iterator
    format(algebra::Matrix<T> const& matrix, FmtContext& ctx) const {
        auto const maybe_new_line = multi_line ? "\n" : "";
        if (matrix.empty()) {
            std::format_to(ctx.out(), "[]");
        } else {
            std::format_to(ctx.out(), "[");
            for (std::size_t i = 0; i < matrix.nrows(); ++i) {
                std::format_to(ctx.out(), "[");
                for (std::size_t j = 0; j < matrix.ncols(); ++j) {
                    coefficient_formatter.format(matrix[i, j], ctx);
                    if (j + 1 < matrix.ncols()) {
                        std::format_to(ctx.out(), ", ");
                    }
                }
                std::format_to(ctx.out(), "]");
                if (i + 1 < matrix.nrows()) {
                    std::format_to(ctx.out(), ",{} ", maybe_new_line);
                }
            }
            std::format_to(ctx.out(), "]");
        }
        if (multi_line) {
            std::format_to(
                ctx.out(),
                "\nMatrix {} x {}\n",
                matrix.nrows(),
                matrix.ncols()
            );
        }
        return ctx.out();
    }

private:
    bool multi_line = false;
    std::formatter<T> coefficient_formatter;
};
```

7.9 mc-homology/algebra/include/algebra/matrix_algorithms.h File Reference

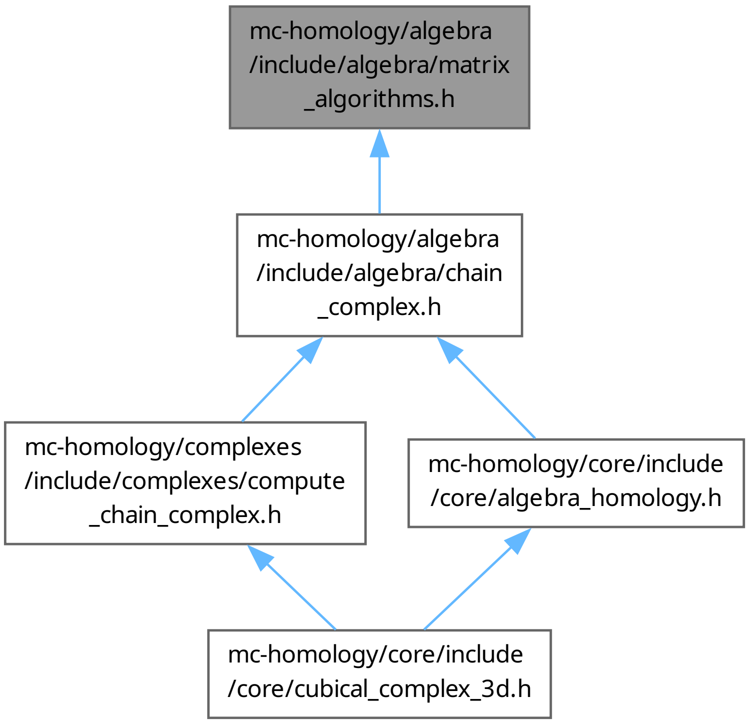
A file containing selected matrix algorithms.

```
#include "algebra/detail/matrix_utils.h"
#include "algebra/matrix.h"
```

Include dependency graph for matrix_algorithms.h:


mc-homology/algebra/include/algebra/matrix_algorithms.h
algebra/detail/matrix_utils.h
algebra/matrix.h
algorithm
format
iostream
ranges
utility
vector
algebra/algebraic_concepts.h
concepts
algebra/number_theory.h
cmath
optional
stdexcept

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


mc-homology/algebra/include/algebra/matrix_algorithms.h
mc-homology/algebra/include/algebra/chain_complex.h
mc-homology/complexes/include/complexes/compute_chain_complex.h
mc-homology/core/include/core/algebra_homology.h
mc-homology/core/include/core/cubical_complex_3d.h

Classes

- struct algebra::RowEchelonFormResult< T >

    Result struct for the row echelon algorithm.
- struct algebra::SmithFormResult< T >

    Result struct for the smith algorithm.

Functions

- template<Field T>
  constexpr std::size_t algebra::row_echelon_form (std::in_place_t, Matrix< T > &matrix)

    Transforms a matrix into a row echolon form in place.
- template<Field T>
  constexpr RowEchelonFormResult< T > algebra::row_echelon_form (Matrix< T > matrix)

    Transforms a matrix into a row echelon form.
- template<EuclideanDomain T>
  constexpr std::size_t algebra::smith_form (std::in_place_t, Matrix< T > &matrix)

    Transforms a matrix into a Smith form in place.
- template<EuclideanDomain T>
  constexpr SmithFormResult< T > algebra::smith_form (Matrix< T > matrix)

    Transforms a matrix into a Smith form.

7.9.1 Detailed Description

A file containing selected matrix algorithms.

7.9.2 Function Documentation

7.9.2.1 row_echelon_form() [1/2]

```
template<Field T>
RowEchelonFormResult< T > algebra::row_echelon_form (
            Matrix< T > matrix)  [constexpr]
```

Transforms a matrix into a row echelon form.

Transforms a matrix into a row echelon form and returns the number of non-zero rows.

Parameters

matrix	Matrix to be transformed

Returns

A struct containing two fields

1. row_echelon_form The transformed matrix
2. non_empty_rows the number of non-zero rows

7.9.2.2 row_echelon_form() [2/2]

```
template<Field T>
std::size_t algebra::row_echelon_form (
            std::in_place_t ,
            Matrix< T > & matrix)  [constexpr]
```

Transforms a matrix into a row echolon form in place.

Transforms a matrix into a row echelon form in place and returns the number of non-zero rows.

Parameters

in,out	*matrix*	Matrix to be transformed

Returns

The number of non-zero rows

7.9.2.3 smith_form() [1/2]

```
template<EuclideanDomain T>
SmithFormResult< T > algebra::smith_form (
            Matrix< T > matrix)  [constexpr]
```

Transforms a matrix into a Smith form.

Transforms a matrix into a Smith form and returns the smaller of non-zero rows or columns

Parameters

matrix	Matrix to be transformed

Returns

A struct containing two fields

1. smith_form The transformed matrix
2. non_empty The number of non-zero rows or columns

7.9.2.4 smith_form() [2/2]

```
template<EuclideanDomain T>
std::size_t algebra::smith_form (
            std::in_place_t ,
            Matrix< T > & matrix)  [constexpr]
```

Transforms a matrix into a Smith form in place.

Transforms a matrix into a Smith form in place and returns the smaller of non-zero rows or columns

Parameters

in,out	*matrix*	Matrix to be transformed

Returns

The number of non-zero rows or columns

7.10 matrix_algorithms.h

Go to the documentation of this file.

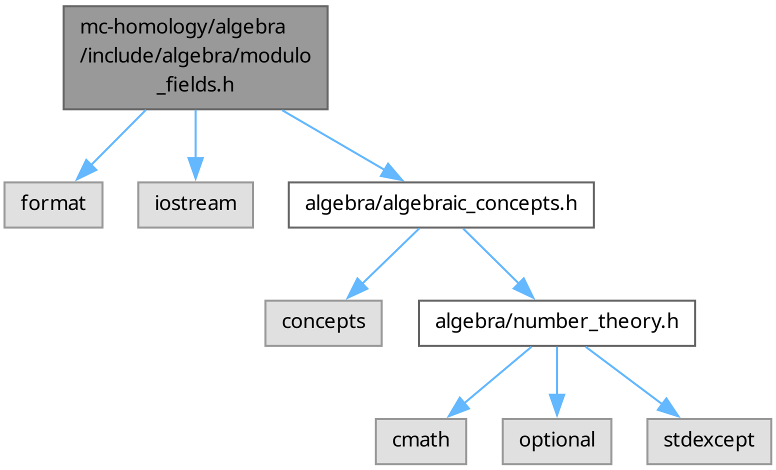
```
#pragma once

#include "algebra/detail/matrix_utils.h"
#include "algebra/matrix.h"

namespace algebra {

template<Field T>
constexpr std::size_t row_echelon_form(std::in_place_t, Matrix<T>& matrix) {
    std::size_t i = 0;
    for (std::size_t j = 0; j < matrix.ncols(); ++j) {
        auto maybe_i =
            detail::first_nonzero_submatrix_column_coefficient(matrix, i, j);
        if (!maybe_i) {
            continue;
        }
        if (*maybe_i != i) {
            detail::submatrix_swap_rows(matrix, i, *maybe_i, j);
        }
        for (auto k = i + 1; k < matrix.nrows(); ++k) {
            auto mult = -matrix[k, j] / matrix[i, j];
            detail::submatrix_add_row(matrix, mult, i, k, j);
        }
        ++i;
    }
    return i;
}

template<class T>
struct RowEchelonFormResult {
    Matrix<T> row_echelon_form = {};

    std::size_t non_empty_rows = 0;
};

template<Field T>
constexpr RowEchelonFormResult<T> row_echelon_form(Matrix<T> matrix) {
    auto i = row_echelon_form(std::in_place, matrix);
    return RowEchelonFormResult {
        .row_echelon_form = matrix,
        .non_empty_rows = i
    };
}

template<EuclideanDomain T>
constexpr std::size_t smith_form(std::in_place_t, Matrix<T>& matrix) {
    std::size_t k = 0;
    auto move_min_to_corner = [&] {
        auto min_element =
            detail::minimal_nonzero_submatrix_element(matrix, k, k);
        if (matrix[min_element.first, min_element.second] == T::zero()) {
            return;
        }
        if (min_element.first != k) {
            detail::submatrix_swap_rows(matrix, k, min_element.first, k);
        }
        if (min_element.second != k) {
            detail::submatrix_swap_cols(matrix, k, min_element.second, k);
        }
    };
    for (; k < std::min(matrix.nrows(), matrix.ncols()); ++k) {
        bool col_all_zeros = false;
        while (!col_all_zeros) {
            move_min_to_corner();
            if (matrix[k, k] == T::zero()) {
                return k;
            }
            col_all_zeros = true;
            for (std::size_t i = k + 1; i < matrix.nrows(); ++i) {
                auto [q, r] = divide(matrix[i, k], matrix[k, k]);
                detail::submatrix_add_row(matrix, -q, k, i, k);
                if (r != T::zero()) {
                    col_all_zeros = false;
                }
            }
        }
        bool row_all_zeros = false;
        while (!row_all_zeros) {
            move_min_to_corner();
            if (matrix[k, k] == T::zero()) {
                return k;
            }
            row_all_zeros = true;
            for (std::size_t j = k + 1; j < matrix.ncols(); ++j) {
                auto [q, r] = divide(matrix[k, j], matrix[k, k]);
                detail::submatrix_add_col(matrix, -q, k, j, k);
                if (r != T::zero()) {
                    row_all_zeros = false;
                }
            }
        }
    }
    if constexpr (std::totally_ordered<T>) {
        for (std::size_t i = 0; i < k; ++i) {
            if (matrix[i, i] < T::zero()) {
                matrix[i, i] = -matrix[i, i];
            }
        }
    }
    return k;
}

template<class T>
struct SmithFormResult {
    Matrix<T> smith_form = {};

    std::size_t non_empty = 0;
};

template<EuclideanDomain T>
constexpr SmithFormResult<T> smith_form(Matrix<T> matrix) {
    auto k = smith_form(std::in_place, matrix);
    return SmithFormResult {.smith_form = matrix, .non_empty = k};
}

} // namespace algebra
```

7.11 mc-homology/algebra/include/algebra/modulo_fields.h File Reference

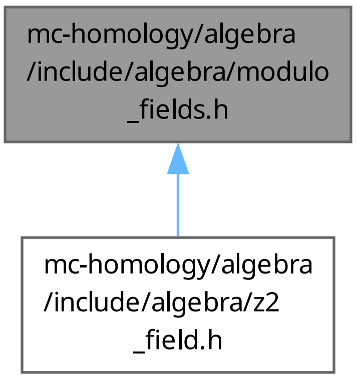
A file containing implementation of fields of integers mod P.

```
#include <format>
#include <iostream>
#include "algebra/algebraic_concepts.h"
```

Include dependency graph for modulo_fields.h:


mc-homology/algebra/include/algebra/modulo_fields.h
format
iostream
algebra/algebraic_concepts.h
concepts
algebra/number_theory.h
cmath
optional
stdexcept

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


mc-homology/algebra /include/algebra/modulo _fields.h
mc-homology/algebra /include/algebra/z2 _field.h

Classes

- class algebra::ZModP< P >

  Field of integers modulo P.
- struct std::formatter< algebra::ZModP< P > >

  Formatter for ZModP type.

Functions

- template<int P>
  constexpr ZModP< P > **algebra::operator+** (ZModP< P > lhs, ZModP< P > rhs) noexcept

  Adds two integers mod P.
- template<int P>
  constexpr ZModP< P > **algebra::operator-** (ZModP< P > lhs, ZModP< P > rhs) noexcept

  Subtracts two integers mod P.
- template<int P>
  constexpr ZModP< P > **algebra::operator*** (ZModP< P > lhs, ZModP< P > rhs) noexcept

  Multiplies two integers mod P.
- template<int P>
  constexpr ZModP< P > **algebra::operator/** (ZModP< P > lhs, ZModP< P > rhs)

  Divides two integers mod P.
- template<int P>
  std::ostream & **algebra::operator<<** (std::ostream &output, ZModP< P > x)

  Outputs an integer mod P to a stream.

Variables

- template<int P>
  constexpr bool **algebra::is_commutative_v< ZModP< P > >** = true

  A marker that multiplying integers mod P is commutative.

7.11.1 Detailed Description

A file containing implementation of fields of integers mod P.

7.12 modulo_fields.h

Go to the documentation of this file.

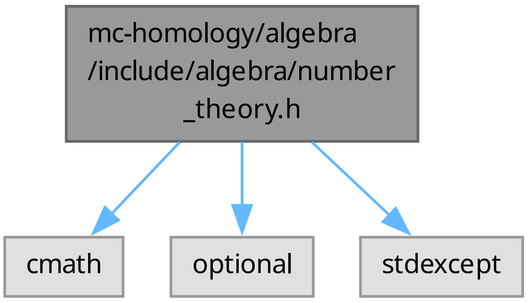
```
#pragma once

#include <format>
#include <iostream>

#include "algebra/algebraic_concepts.h"

namespace algebra {

template<int P>
class ZModP {
    static_assert(is_prime(P));

public:
    constexpr ZModP() = default;

    constexpr ZModP(int n) noexcept : m_inner_representation(modulo(n, P)) {}

    constexpr static int p() noexcept {
        return P;
    }

    constexpr static ZModP zero() noexcept {
        return ZModP(0);
    }

    constexpr static ZModP one() noexcept {
        return ZModP(1);
    }

    constexpr explicit operator int() const noexcept {
        return m_inner_representation;
    }

    constexpr bool operator==(ZModP const&) const = default;

    constexpr ZModP& operator+=(ZModP rhs) noexcept {
        m_inner_representation =
            modulo(m_inner_representation + rhs.m_inner_representation, P);
        return *this;
    }

    constexpr ZModP& operator-=(ZModP rhs) noexcept {
        m_inner_representation =
            modulo(m_inner_representation - rhs.m_inner_representation, P);
        return *this;
    }

    constexpr ZModP operator+() const noexcept {
        return *this;
    }

    constexpr ZModP operator-() const noexcept {
        return ZModP(modulo(-m_inner_representation, P));
    }

    constexpr ZModP& operator*=(ZModP rhs) noexcept {
        m_inner_representation =
            modulo(m_inner_representation * rhs.m_inner_representation, P);
        return *this;
    }

    constexpr int euclidean_function() const noexcept {
        return 1;
    }

    constexpr ZModP& operator/=(ZModP rhs) {
        auto inverse = inverse_mod(rhs.m_inner_representation, P);
        if (!inverse) [[unlikely]] {
            throw std::domain_error("Division by 0");
        }
        return *this *= *inverse;
    }

private:
    int m_inner_representation = 0;
};

template<int P>
constexpr ZModP<P> operator+(ZModP<P> lhs, ZModP<P> rhs) noexcept {
    return lhs += rhs;
}

template<int P>
constexpr ZModP<P> operator-(ZModP<P> lhs, ZModP<P> rhs) noexcept {
    return lhs -= rhs;
}

template<int P>
constexpr ZModP<P> operator*(ZModP<P> lhs, ZModP<P> rhs) noexcept {
    return lhs *= rhs;
}

template<int P>
constexpr ZModP<P> operator/(ZModP<P> lhs, ZModP<P> rhs) {
    return lhs /= rhs;
}

template<int P>
std::ostream& operator«(std::ostream& output, ZModP<P> x) {
    return output « static_cast<int>(x);
}

template<int P>
constexpr inline bool is_commutative_v<ZModP<P>> = true;

} // namespace algebra

template<int P>
struct std::formatter<algebra::ZModP<P»: public std::formatter<int> {
    template<class FmtContext>
    FmtContext::iterator format(algebra::ZModP<P> x, FmtContext& ctx) const {
        return std::formatter<int>::format(static_cast<int>(x), ctx);
    }
};
```

7.13 mc-homology/algebra/include/algebra/number_theory.h File Reference

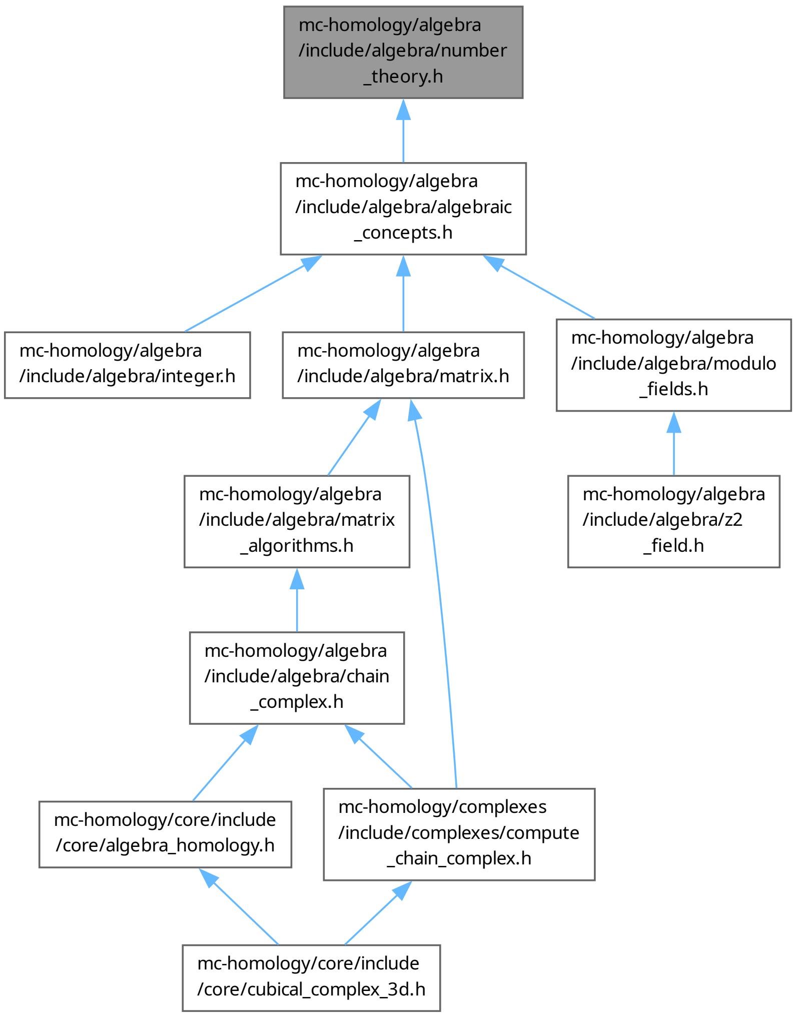
A file containing implementations of selected number theoretic algorithms.

```
#include <cmath>
#include <optional>
#include <stdexcept>
```

Include dependency graph for number_theory.h:



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


mc-homology/algebra /include/algebra/number _theory.h
mc-homology/algebra /include/algebra/algebraic _concepts.h
mc-homology/algebra /include/algebra/integer.h
mc-homology/algebra /include/algebra/matrix.h
mc-homology/algebra /include/algebra/modulo _fields.h
mc-homology/algebra /include/algebra/matrix _algorithms.h
mc-homology/algebra /include/algebra/z2 _field.h
mc-homology/algebra /include/algebra/chain _complex.h
mc-homology/core/include /core/algebra_homology.h
mc-homology/complexes /include/complexes/compute _chain_complex.h
mc-homology/core/include /core/cubical_complex_3d.h

Classes

- struct algebra::DivResult< T >

  Result struct for the division operation.
- struct algebra::ExtendedGCDResult

  Result struct for the extended gcd algorithm.

Functions

- constexpr bool algebra::is_prime (int n) noexcept

  Primality test.
- constexpr DivResult< int > algebra::divide (int a, int b)

  Calculates quotient and remainder of two numbers.
- constexpr int algebra::modulo (int a, int n)

Calculates the remainder of a by n.

- constexpr ExtendedGCDResult algebra::extended_gcd (int a, int b) noexcept

  Extended Euclidean algorithm.

- constexpr std::optional< int > algebra::inverse_mod (int a, int n) noexcept

  Inverse modulo n.

7.13.1 Detailed Description

A file containing implementations of selected number theoretic algorithms.

7.13.2 Function Documentation

7.13.2.1 divide()

```
DivResult< int > algebra::divide (
              int a,
              int b)  [constexpr]
```

Calculates quotient and remainder of two numbers.

A function calculating numbers `quotient` and `remainder` satisfying `a == quotient * n + remainder` and 0 <= remainder < abs(n)

Parameters

a	An integer
b	An integer

Returns

A struct holding three numbers: `quotient`: the quotient `remainder`: the remainder

7.13.2.2 extended_gcd()

```
ExtendedGCDResult algebra::extended_gcd (
              int a,
              int b)  [constexpr], [noexcept]
```

Extended Euclidean algorithm.

Calculates the greatest common divisor `g` and integers `x`, `y` such that `a*x + b*y == g`.

Parameters

a	An integer
b	An integer

Returns

A struct holding three numbers: `g`: the gcd `x`: first coefficient `y`: second coefficient

7.13.2.3 inverse_mod()

```
std::optional< int > algebra::inverse_mod (
            int a,
            int n)  [constexpr], [noexcept]
```

Inverse modulo n.

Calculates inverse of `a` mod `n`.

Parameters

a	Inverted number
n	The modulus

Returns

If `a` is invertible mod `n`, returns the inverse, `nullopt` otherwize

7.13.2.4 is_prime()

```
bool algebra::is_prime (
            int n)  [constexpr], [noexcept]
```

Primality test.

Tests, if a given number is prime, that is divisible only by 1 and by itself

Parameters

n	Number to test

Returns

`true`, if the number is prime, otherwise `false`

7.13.2.5 modulo()

```
int algebra::modulo (
            int a,
            int n)  [constexpr]
```

Calculates the remainder of a by n.

Parameters

a	An integer

n	an integer

Returns

The remainder

7.14 number_theory.h

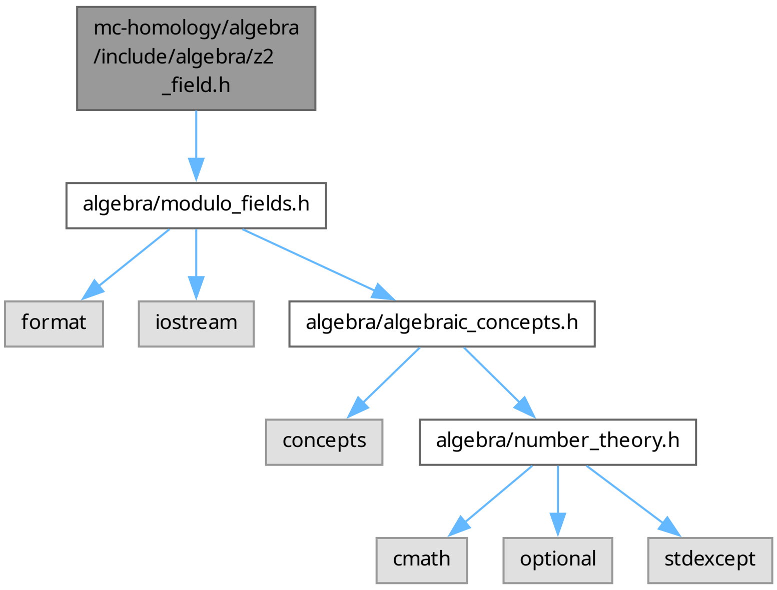
Go to the documentation of this file.

```

#pragma once

#include <cmath>
#include <optional>
#include <stdexcept>

namespace algebra {

constexpr bool is_prime(int n) noexcept {
    if (n < 2) {
        return false;
    }
    for (int i = 2; i * i <= n; ++i) {
        if (n % i == 0) {
            return false;
        }
    }
    return true;
}

template<class T>
struct DivResult {
    T quotient = {};
    T remainder = {};
};

constexpr DivResult<int> divide(int a, int b) {
    if (b != 0) [[likely]] {
        auto [q, r] = std::div(a, b);
        if (r < 0) {
            q -= b > 0 ? 1 : -1;
            r += b > 0 ? b : -b;
        }
        return {.quotient = q, .remainder = r};
    } else [[unlikely]] {
        throw std::domain_error("Division by 0");
    }
}

constexpr int modulo(int a, int n) {
    return divide(a, n).remainder;
}

struct ExtendedGCDResult {
    int g = 0;
    int x = 0;
    int y = 0;
};

constexpr ExtendedGCDResult extended_gcd(int a, int b) noexcept {
    a = std::abs(a);
    b = std::abs(b);
    int x1 = 1;
    int x2 = 0;
    int y1 = 0;
    int y2 = 1;
    while (b > 0) {
        auto [q, r] = divide(a, b);
        std::tie(a, b) = std::pair {b, r};
        std::tie(x1, x2) = std::pair {x2, x1 - q * x2};
        std::tie(y1, y2) = std::pair {y2, y1 - q * y2};
    }
    return {.g = a, .x = x1, .y = y1};
}

constexpr std::optional<int> inverse_mod(int a, int n) noexcept {
    auto [g, x, _] = extended_gcd(a, n);
    if (g == 1) [[likely]] {
        return x;
    } else [[unlikely]] {
        return std::nullopt;
    }
}

} // namespace algebra
```

7.15 mc-homology/algebra/include/algebra/z2_field.h File Reference

A file containing optimized implementation of Z2 field.

`#include "algebra/modulo_fields.h"`
Include dependency graph for z2_field.h:



Classes

- class algebra::ZModP< 2 >

  Template specialization for P == 2.

Typedefs

- using **algebra::Z2** = ZModP<2>

  Alias for ZModP<2>.

7.15.1 Detailed Description

A file containing optimized implementation of Z2 field.

7.16 z2_field.h

Go to the documentation of this file.

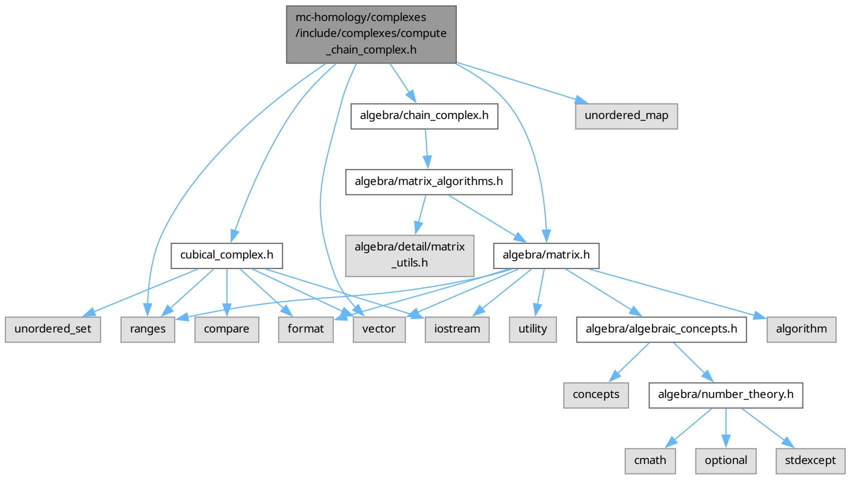
```
#pragma once

#include "algebra/modulo_fields.h"

namespace algebra {

template<>
class ZModP<2> {
public:
    constexpr ZModP() = default;

    constexpr ZModP(int n) noexcept : m_inner_representation(modulo(n, 2)) {}

    constexpr static int p() noexcept {
        return 2;
    }

    constexpr static ZModP zero() noexcept {
        return ZModP(0);
    }

    constexpr static ZModP one() noexcept {
        return ZModP(1);
    }

    constexpr explicit operator int() const noexcept {
        return m_inner_representation;
    }

    constexpr bool operator==(ZModP const&) const = default;

    constexpr ZModP& operator+=(ZModP rhs) noexcept {
        m_inner_representation ^= rhs.m_inner_representation;
        return *this;
    }

    constexpr ZModP& operator-=(ZModP rhs) noexcept {
        m_inner_representation ^= rhs.m_inner_representation;
        return *this;
    }

    constexpr ZModP operator+() const noexcept {
        return *this;
    }

    constexpr ZModP operator-() const noexcept {
        return *this;
    }

    constexpr ZModP& operator*=(ZModP rhs) noexcept {
        m_inner_representation &= rhs.m_inner_representation;
        return *this;
    }

    constexpr int euclidean_function() const noexcept {
        return 1;
    }

    constexpr ZModP& operator/=(ZModP rhs) {
        if (!rhs.m_inner_representation) [[unlikely]] {
            throw std::domain_error("Division by 0");
        }
        return *this;
    }

private:
    bool m_inner_representation = false;
};

using Z2 = ZModP<2>;

} // namespace algebra
```

7.17 mc-homology/complexes/include/complexes/compute_chain_↩ complex.h File Reference

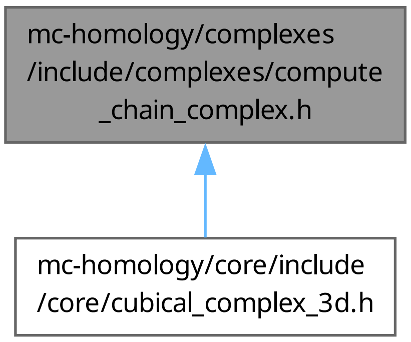
A file containing algorithms for transforming complexes into chain complexes.

```
#include <ranges>
#include <unordered_map>
#include <vector>
#include "algebra/chain_complex.h"
#include "algebra/matrix.h"
#include "cubical_complex.h"
```

Include dependency graph for compute_chain_complex.h:



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



Functions

- template<class T>
  algebra::ChainComplex< T > complexes::compute_chain_complex (CubicalComplex const &cubical_↩
  complex)

  Computes a chain complex from a cubical complex.

7.17.1 Detailed Description

A file containing algorithms for transforming complexes into chain complexes.

7.17.2 Function Documentation

7.17.2.1 compute_chain_complex()

```
template<class T>
algebra::ChainComplex< T > complexes::compute_chain_complex (
            CubicalComplex const & cubical_complex)
```

Computes a chain complex from a cubical complex.

Transforms relationships between faces into boundary operators

Parameters

cubical_complex	Complex to transorm

Returns

A chain complex

7.18 compute_chain_complex.h

Go to the documentation of this file.

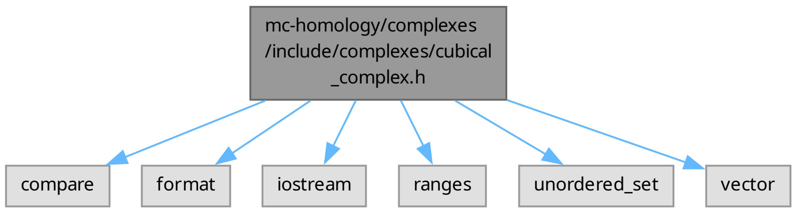
```

#pragma once

#include <ranges>
#include <unordered_map>
#include <vector>

#include "algebra/chain_complex.h"
#include "algebra/matrix.h"
#include "cubical_complex.h"

namespace complexes {

template<class T>
algebra::ChainComplex<T>
compute_chain_complex(CubicalComplex const& cubical_complex) {
    namespace vs = std::views;
    auto const& simplices = cubical_complex.simplices();
    if (simplices.empty()) {
        return algebra::ChainComplex<T> {};
    }
    std::vector<algebra::Matrix<T» boundaries(simplices.size());
    // 0'th dimensional matrix is empty, we are computing non-reduced
    // homology
    boundaries[0] = algebra::Matrix<T>::zero(0, simplices[0].size());
    for (std::size_t dim = 1; dim < simplices.size(); ++dim) {
        boundaries[dim] = algebra::Matrix<T>::zero(
            simplices[dim - 1].size(),
            simplices[dim].size()
        );
        std::unordered_map<CubicalSimplex, std::size_t> assigned_rows;
        for (auto const& [s, i] : vs::zip(simplices[dim - 1], vs::iota(0))) {
            assigned_rows.emplace(s, i);
        }
        for (auto const& [j, simplex] : simplices[dim] | vs::enumerate) {
            std::array sgn = {1, -1, -1, 1};
            for (auto const& [k, bd] : simplex.boundary() | vs::enumerate) {
                boundaries[dim][assigned_rows[bd], j] = sgn[k % sgn.size()];
            }
        }
    }
    return algebra::ChainComplex<T> {std::move(boundaries)};
}

} // namespace complexes
```

7.19 mc-homology/complexes/include/complexes/cubical_complex.h File Reference

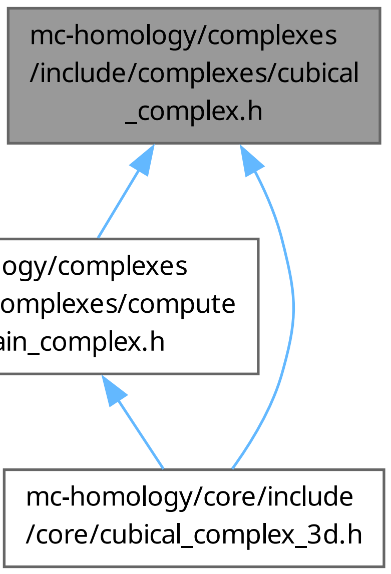
A file containing the CubicalComplex class.

```
#include <compare>
#include <format>
#include <iostream>
#include <ranges>
#include <unordered_set>
#include <vector>
```

Include dependency graph for cubical_complex.h:



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


mc-homology/complexes /include/complexes/cubical _complex.h
mc-homology/complexes /include/complexes/compute _chain_complex.h
mc-homology/core/include /core/cubical_complex_3d.h

Classes

- class complexes::BasicInterval

  A struct representing an interval of length 0 or 1.
- class complexes::CubicalSimplex

  A class representing a single cubical simplex in a complex.
- class complexes::CubicalComplex

  Class representing a cubical complex.
- struct std::hash< complexes::BasicInterval >

  std::hash specialization for Interval
- struct std::hash< complexes::CubicalSimplex >

  std::hash specialization for CubicalSimplex
- struct std::formatter< complexes::BasicInterval >

  Formatter for BasicInterval type.
- struct std::formatter< complexes::CubicalSimplex >

  Formatter for CubicalSimplex type.

Functions

- std::ostream & **complexes::operator**<< (std::ostream &output, BasicInterval i)

  Prints an interval to output.
- std::ostream & **complexes::operator**<< (std::ostream &output, CubicalSimplex const &s)

  Prints a cubical simplex to output.

7.19.1 Detailed Description

A file containing the CubicalComplex class.

7.20 cubical_complex.h

Go to the documentation of this file.

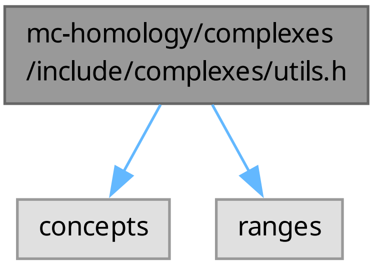
```

#pragma once

#include <compare>
#include <format>
#include <iostream>
#include <ranges>
#include <unordered_set>
#include <vector>

namespace complexes {

class BasicInterval {
public:
    BasicInterval();

    std::size_t hash() const;

    bool operator==(BasicInterval const&) const;

    int left() const;

    int right() const;

    bool is_trivial() const;

    static BasicInterval point(int p);

    static BasicInterval interval(int left);

private:
    int m_left = 0;
    bool m_full = false;
};

std::ostream& operator«(std::ostream& output, BasicInterval i);

class CubicalSimplex {
public:
    CubicalSimplex(std::vector<BasicInterval> intervals);

    std::size_t dimension() const;

    std::size_t ambient_dimension() const;

    std::size_t hash() const;

    std::vector<CubicalSimplex> boundary() const;

    bool operator==(CubicalSimplex const&) const;

    std::vector<BasicInterval> const& intervals() const;

    std::strong_ordering operator<=>(CubicalSimplex const& simplex) const;

    static CubicalSimplex point(int p);

    static CubicalSimplex interval(int left);

    friend CubicalSimplex
    product(CubicalSimplex const& s1, CubicalSimplex const& s2);

private:
    CubicalSimplex();

    std::vector<BasicInterval> m_intervals = {};
    std::size_t m_dimension = 0;
};

std::ostream& operator«(std::ostream& output, CubicalSimplex const& s);

class CubicalComplex {
public:
    CubicalComplex();

    bool add(CubicalSimplex simplex);

    void add_recursive(CubicalSimplex simplex);

    bool remove(CubicalSimplex const& simplex);

    bool contains(CubicalSimplex const& simplex) const;

    bool operator==(CubicalComplex const&) const;

    std::vector<std::unordered_set<CubicalSimplex» const& simplices() const;

    std::size_t dimension() const;

    std::size_t ambient_dimension() const;

private:
    void add_recursive_impl(CubicalSimplex simplex);

    std::vector<std::unordered_set<CubicalSimplex» m_simplices;
};

} // namespace complexes

template<>
struct std::hash<complexes::BasicInterval> {
    std::size_t operator()(complexes::BasicInterval const& i) const;
};

template<>
struct std::hash<complexes::CubicalSimplex> {
    std::size_t operator()(complexes::CubicalSimplex const& s) const;
};

template<>
struct std::formatter<complexes::BasicInterval>: public std::formatter<int> {
    template<class FmtContext>
    FmtContext::iterator
    format(complexes::BasicInterval i, FmtContext& ctx) const {
        std::format_to(ctx.out(), "[");
        std::formatter<int>::format(i.left(), ctx);
        if (!i.is_trivial()) {
            std::format_to(ctx.out(), ", ");
            std::formatter<int>::format(i.right(), ctx);
        }
        std::format_to(ctx.out(), "]");
        return ctx.out();
    }
};

template<>
struct std::formatter<complexes::CubicalSimplex>:
    public std::formatter<complexes::BasicInterval> {
    template<class FmtContext>
    FmtContext::iterator
    format(complexes::CubicalSimplex const& s, FmtContext& ctx) const {
        namespace vs = std::views;
        std::formatter<complexes::BasicInterval>::format(
            s.intervals().front(),
            ctx
        );
        for (auto i : s.intervals() | vs::drop(1)) {
            std::format_to(ctx.out(), "x");
            std::formatter<complexes::BasicInterval>::format(i, ctx);
        }
        return ctx.out();
    }
};
```

7.21 mc-homology/complexes/include/complexes/utils.h File Reference

Contains auxillary classes and functions.

```
#include <concepts>
#include <ranges>
```

Include dependency graph for utils.h:



Functions

- std::size_t **complexes::utils::combine_hashes** (std::size_t hash1, std::size_t hash2)

  Combines two hashes into a single hash.
- template<std::ranges::range R>
  std::size_t **complexes::utils::hash_range** (R &&r)

  Combines hashes of a range into a single hash.

7.21.1 Detailed Description

Contains auxillary classes and functions.

7.22 utils.h

Go to the documentation of this file.

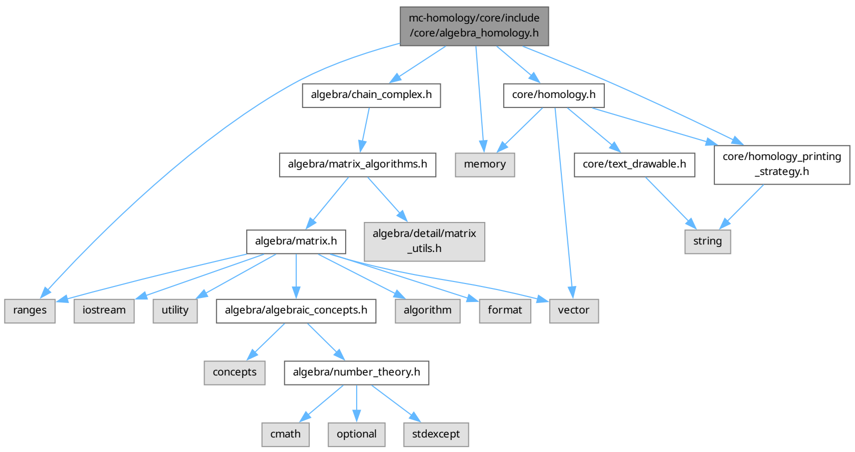
```

#include <concepts>
#include <ranges>

namespace complexes {
namespace utils {

std::size_t combine_hashes(std::size_t hash1, std::size_t hash2);

template<std::ranges::range R>
    requires requires(std::ranges::range_value_t<R> x) {
        {
            std::hash<std::ranges::range_value_t<R» {}(x)
        } -> std::convertible_to<std::size_t>;
    }
std::size_t hash_range(R&& r) {
    std::size_t hash = std::hash<std::size_t> {}(0);
    for (auto&& x : std::forward<R>(r)) {
        hash = combine_hashes(
            hash,
            std::hash<std::ranges::range_value_t<R» {}(
                std::forward<decltype(x)>(x)
            )
        );
    }
    return hash;
}

} // namespace utils
} // namespace complexes
```

7.23 mc-homology/core/include/core/algebra_homology.h File Reference

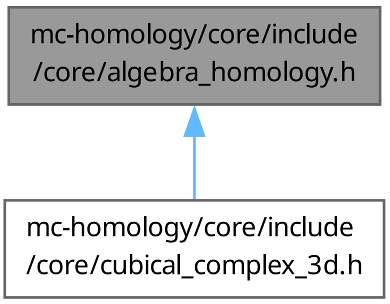
File containing concrete classes deriving from Homology.

```
#include <memory>
#include <ranges>
#include "algebra/chain_complex.h"
#include "core/homology.h"
#include "core/homology_printing_strategy.h"
```

Include dependency graph for algebra_homology.h:



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



Classes

- class core::AlgebraHomology< T >

  Concrete subclass of Homology, based on algebra library.

7.23.1 Detailed Description

File containing concrete classes deriving from Homology.

7.24 algebra_homology.h

Go to the documentation of this file.

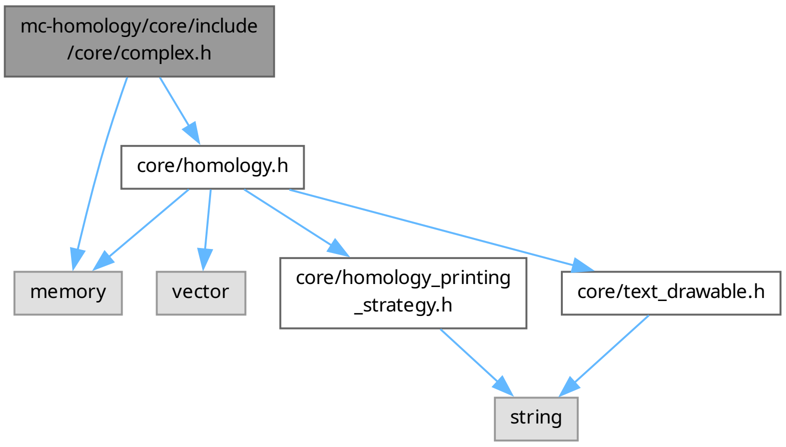
```
#pragma once

#include <memory>
#include <ranges>

#include "algebra/chain_complex.h"
#include "core/homology.h"
#include "core/homology_printing_strategy.h"

namespace core {

template<class T>
class AlgebraHomology: public Homology {
public:
    AlgebraHomology(
        algebra::Homology<T> homology,
        std::unique_ptr<HomologyPrintingStrategy> printing_strategy =
            std::make_unique<HomologyRawPrint>()
    ) :
        Homology(std::move(printing_strategy)),
        m_homology(std::move(homology)) {}

    std::vector<std::size_t> betti_numbers() const override {
        return m_homology.betti_numbers;
    }

    std::vector<std::vector<std::string>> torsion() const override {
        namespace rs = std::ranges;
        namespace vs = std::views;
        return rs::to<std::vector>(
            m_homology.torsion | vs::transform([](auto const& tor) {
                return rs::to<std::vector>(tor | vs::transform([](auto x) {
                                                  return std::format("{}", x);
                                              }));
            })
        );
    }

private:
    algebra::Homology<T> m_homology;
};

} // namespace core
```

7.25 mc-homology/core/include/core/complex.h File Reference

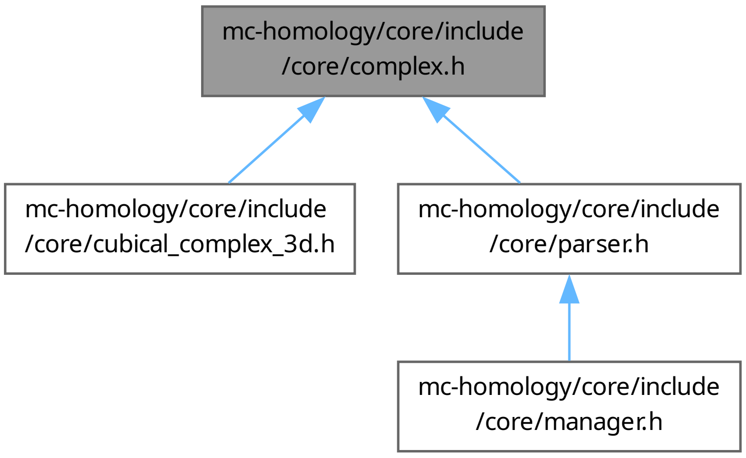
A file containing interface for the Complex class.

```
#include <memory>
#include "core/homology.h"
```

Include dependency graph for complex.h:



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



Classes

- class core::Complex

  Interface for complexes.

7.25.1 Detailed Description

A file containing interface for the Complex class.

7.26 complex.h

Go to the documentation of this file.

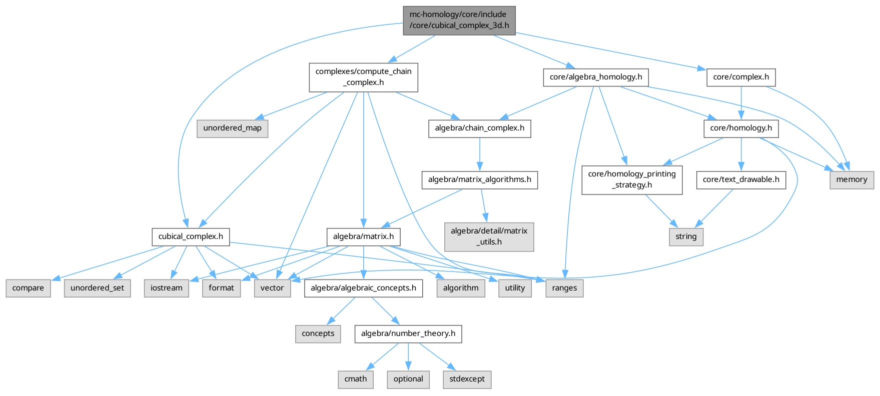
```

#pragma once

#include <memory>

#include "core/homology.h"

namespace core {

class Complex {
public:
    virtual std::unique_ptr<Homology> z2_homology() const = 0;

    virtual std::unique_ptr<Homology> z3_homology() const = 0;

    virtual std::unique_ptr<Homology> z_homology() const = 0;

    virtual void reduce() = 0;

    virtual ~Complex();
};

} // namespace core
```

7.27 mc-homology/core/include/core/cubical_complex_3d.h File Reference

A file containing a concrete class CubicalComplex3D.

```
#include "complexes/compute_chain_complex.h"
#include "complexes/cubical_complex.h"
#include "core/algebra_homology.h"
#include "core/complex.h"
```

Include dependency graph for cubical_complex_3d.h:



Classes

- class core::CubicalComplex3D

    A class representing a cubical complex in 3D space.

7.27.1 Detailed Description

A file containing a concrete class CubicalComplex3D.

7.28 cubical_complex_3d.h

Go to the documentation of this file.

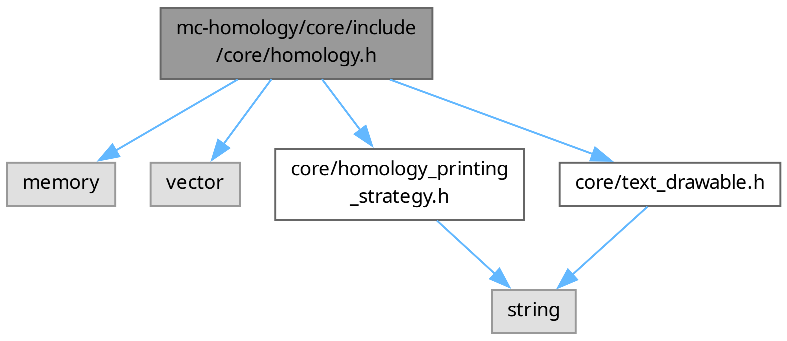
```
#pragma once

#include "complexes/compute_chain_complex.h"
#include "complexes/cubical_complex.h"
#include "core/algebra_homology.h"
#include "core/complex.h"

namespace core {

class CubicalComplex3D: public Complex {
public:
    void add_cube(int x, int y, int z);

    std::unique_ptr<Homology> z2_homology() const override;

    std::unique_ptr<Homology> z3_homology() const override;

    std::unique_ptr<Homology> z_homology() const override;

    template<class T>
    std::unique_ptr<AlgebraHomology<T>> homology() const {
        return std::make_unique<AlgebraHomology<T>>(
            algebra::homology(complexes::compute_chain_complex<T>(m_inner))
        );
    }

    void reduce() override;

private:
    complexes::CubicalComplex m_inner;
};

} // namespace core
```

7.29 mc-homology/core/include/core/homology.h File Reference

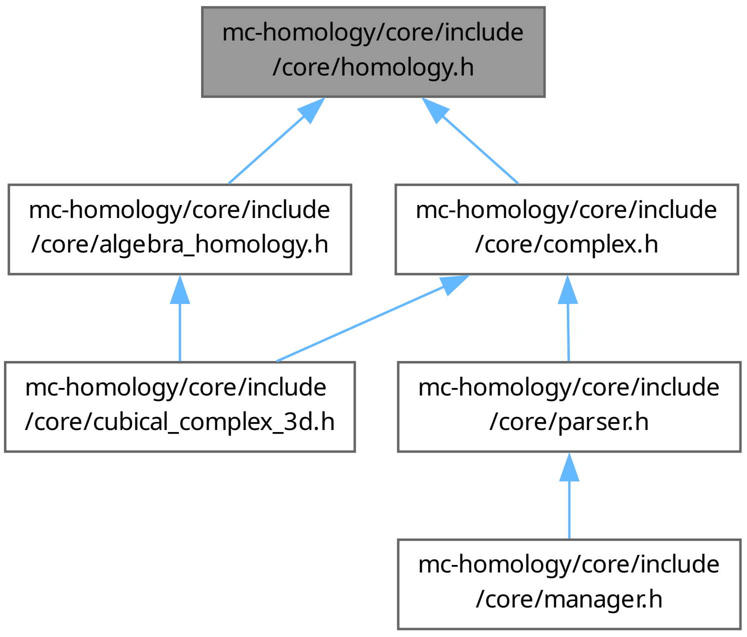
A file containing a Homology interface.

```
#include <memory>
#include <vector>
#include "core/homology_printing_strategy.h"
#include "core/text_drawable.h"
```

Include dependency graph for homology.h:



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



Classes

- class core::Homology

  A homology interface.

7.29.1 Detailed Description

A file containing a Homology interface.

7.30 homology.h

Go to the documentation of this file.

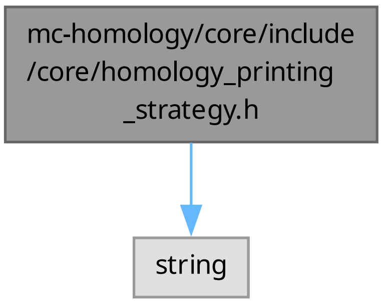
```
#pragma once

#include <memory>
#include <vector>

#include "core/homology_printing_strategy.h"
#include "core/text_drawable.h"

namespace core {

class Homology: public TextDrawable {
public:
    Homology(
        std::unique_ptr<HomologyPrintingStrategy> printing_strategy =
            std::make_unique<HomologyRawPrint>()
    );

    void select_strategy(
        std::unique_ptr<HomologyPrintingStrategy> printing_strategy
    );

    std::string text() const override;

    virtual std::vector<std::size_t> betti_numbers() const = 0;

    virtual std::vector<std::vector<std::string» torsion() const = 0;

    // \brief Virtual destructor
    virtual ~Homology();

private:
    std::unique_ptr<HomologyPrintingStrategy> m_printing_strategy;
};

} // namespace core
```

7.31 mc-homology/core/include/core/homology_printing_strategy.h File Reference

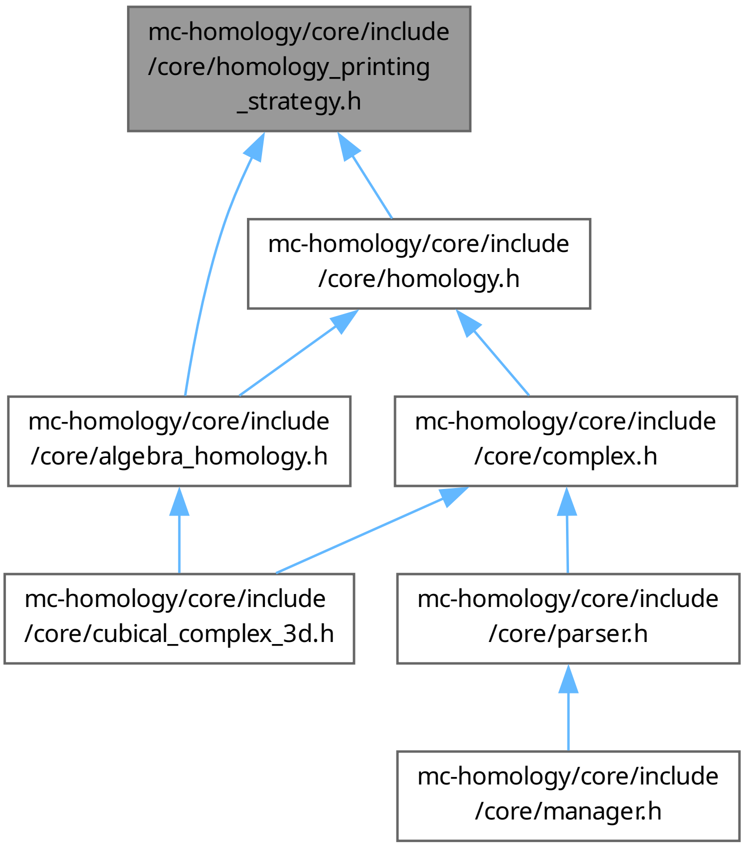
File containing strategies for printing homology.

`#include <string>`

Include dependency graph for homology_printing_strategy.h:



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


mc-homology/core/include /core/homology_printing _strategy.h
mc-homology/core/include /core/homology.h
mc-homology/core/include /core/algebra_homology.h
mc-homology/core/include /core/complex.h
mc-homology/core/include /core/cubical_complex_3d.h
mc-homology/core/include /core/parser.h
mc-homology/core/include /core/manager.h

Classes

- class core::HomologyPrintingStrategy

  A strategy for printing a homology group.
- class core::HomologyRawPrint

  A strategy for basic string representation of homology.
- class core::HomologyLatexPrint

  Prints a homology group in Latex syntax.

7.31.1 Detailed Description

File containing strategies for printing homology.

7.32 homology_printing_strategy.h

Go to the documentation of this file.

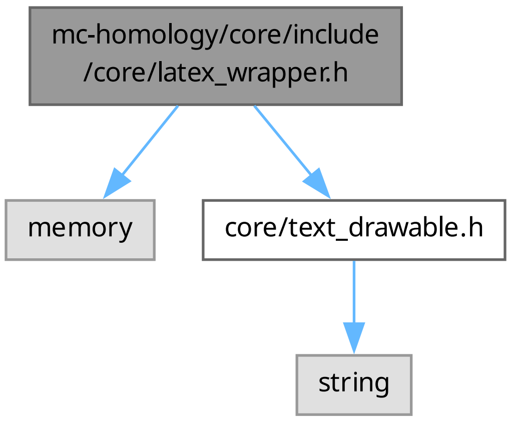
```
#pragma once

#include <string>

namespace core {

class Homology;

class HomologyPrintingStrategy {
public:
    virtual std::string draw(Homology const& homology) const = 0;

    virtual ~HomologyPrintingStrategy();
};

class HomologyRawPrint: public HomologyPrintingStrategy {
public:
    std::string draw(Homology const& homology) const override;
};

class HomologyLatexPrint: public HomologyPrintingStrategy {
public:
    HomologyLatexPrint(std::string ring_name, std::string homology_name = "H");

    std::string draw(Homology const& homology) const override;

private:
    std::string m_ring_name;
    std::string m_homology_name;
};

} // namespace core
```

7.33 mc-homology/core/include/core/latex_wrapper.h File Reference

Turns a TextDrawable into a simple latex document.

```
#include <memory>
#include "core/text_drawable.h"
```

Include dependency graph for latex_wrapper.h:



Classes

- class core::LatexWrapper

    A decorator that wraps a text drawable object in a latex document.

7.33.1 Detailed Description

Turns a TextDrawable into a simple latex document.

7.34 latex_wrapper.h

Go to the documentation of this file.

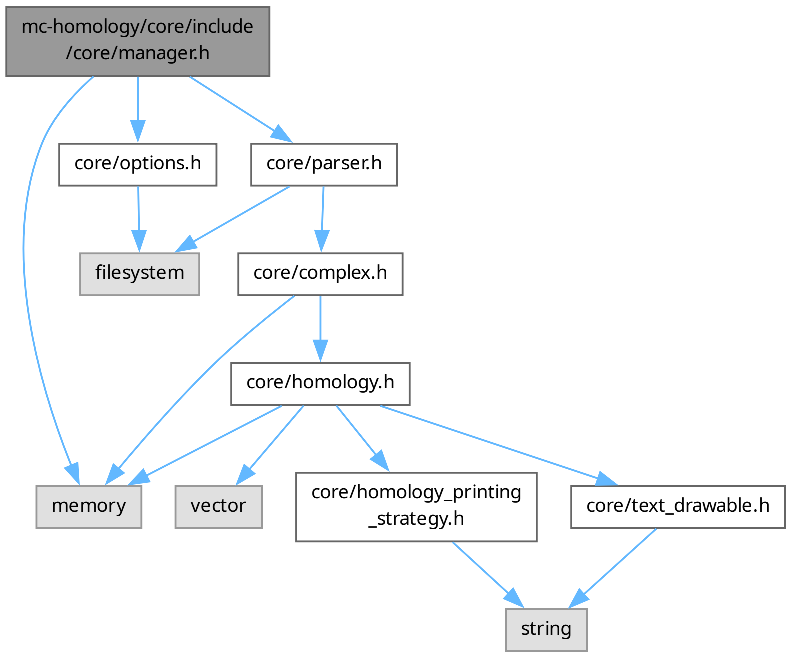
```
#pragma once

#include <memory>

#include "core/text_drawable.h"

namespace core {

class LatexWrapper: public TextDrawable {
public:
    LatexWrapper(
        std::unique_ptr<TextDrawable> inner,
        std::string documentclass = "article"
    );

    std::string text() const override;

private:
    std::unique_ptr<TextDrawable> m_inner;
    std::string m_documentclass;
};

} // namespace core
```

7.35 mc-homology/core/include/core/manager.h File Reference

A file containing main logic of the program.

```
#include <memory>
#include "core/options.h"
#include "core/parser.h"
```

Include dependency graph for manager.h:



Classes

- class core::Manager

  Main manager for the program.

7.35.1 Detailed Description

A file containing main logic of the program.

7.36 manager.h

Go to the documentation of this file.

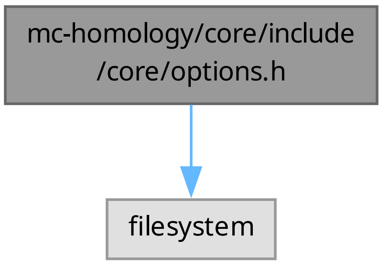
```
#pragma once

#include <memory>

#include "core/options.h"
#include "core/parser.h"

namespace core {

class Manager {
public:
    Manager(
        std::unique_ptr<Options> options,
        std::unique_ptr<MinecraftSavefileParser> parser
    );

    void set_options(std::unique_ptr<Options> options);

    void set_parser(std::unique_ptr<MinecraftSavefileParser> parser);

    int run();

private:
    std::unique_ptr<Options> m_options;
    std::unique_ptr<MinecraftSavefileParser> m_parser;
};

} // namespace core
```

7.37 mc-homology/core/include/core/options.h File Reference

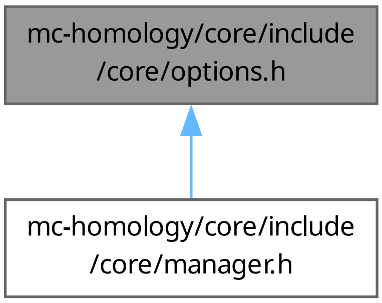
File containing an Options class for storing user options.

`#include <filesystem>`

Include dependency graph for options.h:



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


mc-homology/core/include/core/options.h
mc-homology/core/include/core/manager.h

Classes

- class core::Options

    A class for storing user options.
- class core::CommandlineOptions

    A class for storing user's options from the commandline.

Enumerations

- enum class core::HomologyChoice { **Z** , **Z2** , **Z3** }

    Enum for storing user's choice of homology to compute.

7.37.1 Detailed Description

File containing an Options class for storing user options.

7.38 options.h

Go to the documentation of this file.

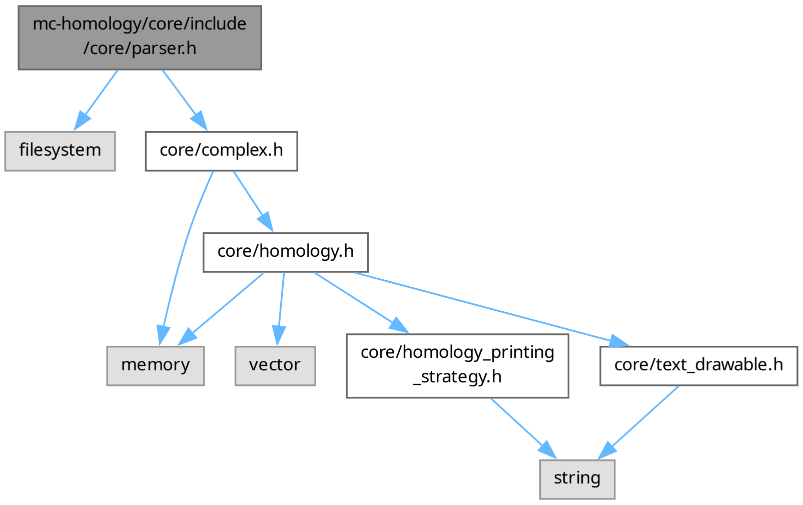
```

#pragma once

#include <filesystem>

namespace core {

enum class HomologyChoice {
    Z,
    Z2,
    Z3,
};

class Options {
public:
    virtual std::filesystem::path filename() const = 0;

    virtual std::pair<int, int> x_bounds() const = 0;

    virtual std::pair<int, int> y_bounds() const = 0;

    virtual std::pair<int, int> z_bounds() const = 0;

    virtual HomologyChoice homology_to_compute() const = 0;

    virtual bool latex() const = 0;

    virtual bool help() const = 0;

    virtual ~Options();
};

class CommandlineOptions: public Options {
public:
    CommandlineOptions(int argc, char** argv);

    std::filesystem::path filename() const override;

    std::pair<int, int> x_bounds() const override;

    std::pair<int, int> y_bounds() const override;

    std::pair<int, int> z_bounds() const override;

    HomologyChoice homology_to_compute() const override;

    bool latex() const override;

    bool help() const override;
private:
    std::filesystem::path m_filename = "";
    std::pair<int, int> m_x_bounds = {0, 0};
    std::pair<int, int> m_y_bounds = {0, 0};
    std::pair<int, int> m_z_bounds = {0, 0};
    HomologyChoice m_homology_to_compute = HomologyChoice::Z2;
    bool m_latex = false;
    bool m_help = false;
};

} // namespace core
```

7.39 mc-homology/core/include/core/parser.h File Reference

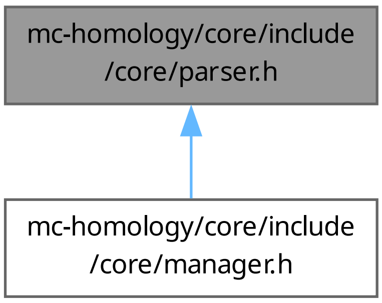
A file containing a minecraft savefile parser.

```
#include <filesystem>
#include "core/complex.h"
```

Include dependency graph for parser.h:



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


mc-homology/core/include /core/parser.h
mc-homology/core/include /core/manager.h

Classes

- struct core::MinecraftCoordinates

  A struct containing coordinates in a Minecraft world.
- class core::MinecraftSavefileParser

  An interface for Minecraft savefile parser.
- class core::MinecraftSavefileParser_mcSavefileParsers

  A Minecraft savefile parser based on `https://github.com/TCA166/mcSavefileParsers.git`.

7.39.1 Detailed Description

A file containing a minecraft savefile parser.

7.40 parser.h

Go to the documentation of this file.

```
#pragma once

#include <filesystem>

#include "core/complex.h"

namespace core {

struct MinecraftCoordinates {
    int x = 0;
    int y = 0;
    int z = 0;
};

class MinecraftSavefileParser {
public:
    virtual std::unique_ptr<Complex> parse(
        std::filesystem::path const& path,
        MinecraftCoordinates lower_corner,
        MinecraftCoordinates upper_corner
    ) = 0;
    virtual ~MinecraftSavefileParser();
};

class MinecraftSavefileParser_mcSavefileParsers:
    public MinecraftSavefileParser {
public:
    std::unique_ptr<Complex> parse(
        std::filesystem::path const& path,
        MinecraftCoordinates lower_corner,
        MinecraftCoordinates upper_corner
    ) override;
};

} // namespace core
```

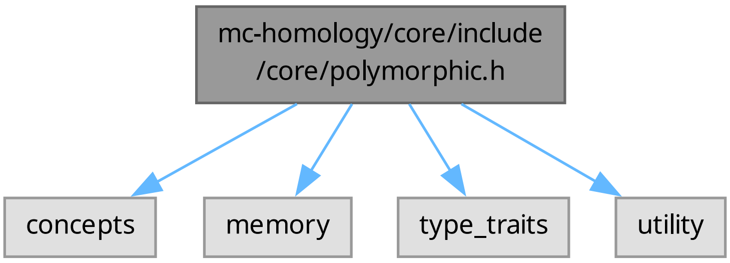
7.41 mc-homology/core/include/core/polymorphic.h File Reference

A polymorphic type wrapper with value sematincs.

```
#include <concepts>
#include <memory>
#include <type_traits>
```

```
#include <utility>
```

Include dependency graph for polymorphic.h:



Classes

- class core::Polymorphic< T >

  A polymorphic class wrapper with value semantics.

7.41.1 Detailed Description

A polymorphic type wrapper with value sematincs.

This file contaings a simplified implementation of std::polymorphic from c++26

7.42 polymorphic.h

Go to the documentation of this file.

```
#pragma once

#include <concepts>
#include <memory>
#include <type_traits>
#include <utility>

namespace core {

#ifdef __cpp_lib_polymorphic

template<class T>
using Polymorphic = std::polymorphic<T>
#else

template<class T>
class Polymorphic {
public:
    constexpr explicit Polymorphic()
        requires std::default_initializable<T> && std::copy_constructible<T>
        : Polymorphic(T {}) {}

    template<class U = T>
        requires std::derived_from<std::remove_cvref_t<U>, T>
        && std::copy_constructible<std::remove_cvref_t<U»
        && std::constructible_from<std::remove_cvref_t<U>, U>
    constexpr explicit Polymorphic(U&& u) :
        m_object(
            std::make_unique<Model<std::remove_cvref_t<U»>(std::forward<U>(u))
        ) {}

    template<class U, class... Args>
        requires std::derived_from<std::remove_cvref_t<U>, T>
        && std::copy_constructible<std::remove_cvref_t<U»
        && std::constructible_from<std::remove_cvref_t<U>, Args...>
        && std::same_as<std::remove_cvref_t<U>, U>
    constexpr explicit Polymorphic(std::in_place_type_t<U>, Args&&... args) :
        m_object(
            std::make_unique<Model<std::remove_cvref_t<U»>(
                std::forward<Args>(args)...
            )
        ) {}

    constexpr Polymorphic(Polymorphic const& other) :
        m_object(other.m_object->clone()) {}

    constexpr Polymorphic(Polymorphic&& other) noexcept :
        m_object(std::move(other.m_object)) {}

    constexpr Polymorphic& operator=(Polymorphic const& other) {
        m_object = other.m_object->clone();
        return *this;
    }

    constexpr Polymorphic& operator=(Polymorphic&& other) noexcept {
        m_object = std::move(other.m_object);
        return *this;
    }

    constexpr T const* operator->() const noexcept {
        return m_object->get_address();
    }

    constexpr T* operator->() noexcept {
        return m_object->get_address();
    }

    constexpr T const& operator*() const noexcept {
        return *m_object->get_address();
    }

    constexpr T& operator*() noexcept {
        return *m_object->get_address();
    }

    constexpr bool valueless_after_move() const noexcept {
        return m_object == nullptr;
    }

    constexpr void swap(Polymorphic& other) noexcept {
        using std::swap;
        swap(m_object, other.m_object);
    }

private:
    struct Concept {
        constexpr virtual T const* get_address() const noexcept = 0;
        constexpr virtual T* get_address() noexcept = 0;
        constexpr virtual std::unique_ptr<Concept> clone() const = 0;
        constexpr virtual ~Concept() = default;
    };

    template<class Object>
    struct Model: public Concept {
        template<class U = Object>
        constexpr Model(U&& value) : m_value(std::forward<U>(value)) {}

        constexpr T const* get_address() const noexcept override {
            return std::addressof(m_value);
        }

        constexpr T* get_address() noexcept override {
            return std::addressof(m_value);
        }

        constexpr std::unique_ptr<Concept> clone() const override {
            return std::make_unique<Model<Object»(m_value);
        }

    private:
        Object m_value;
    };

    std::unique_ptr<Concept> m_object;
```

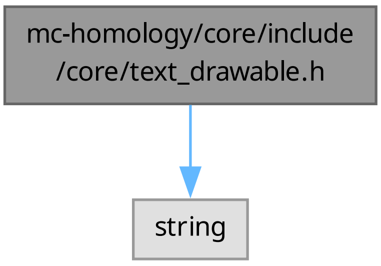
```
};

#endif

} // namespace core
```

7.43 mc-homology/core/include/core/text_drawable.h File Reference

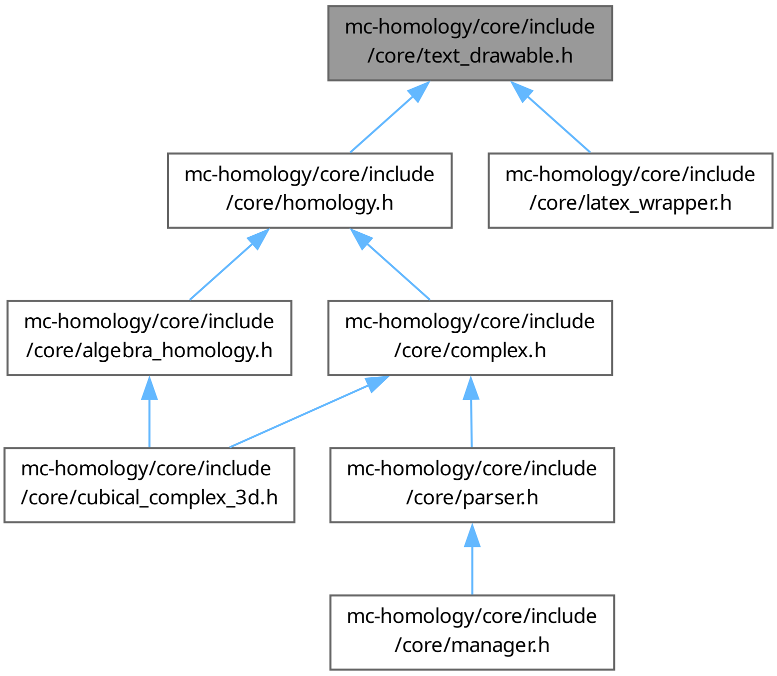
A file containing a TextDrawable interface, for classes that can be drawn on screen in the form of text.

`#include <string>`

Include dependency graph for text_drawable.h:



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



Classes

- class core::TextDrawable

  Marks a class text drawable - it can be represented as a string.

7.43.1 Detailed Description

A file containing a TextDrawable interface, for classes that can be drawn on screen in the form of text.

7.44 text_drawable.h

Go to the documentation of this file.

```

#pragma once

#include <string>

namespace core {

class TextDrawable {
public:
    virtual std::string text() const = 0;

    virtual ~TextDrawable();
};

} // namespace core
```

Index