

mc-homology

Generated by Doxygen 1.16.1



<b>1 Concept Index</b>	<b>1</b>
1.1 Concepts	1
<b>2 Hierarchical Index</b>	<b>3</b>
2.1 Class Hierarchy	3
<b>3 Class Index</b>	<b>5</b>
3.1 Class List	5
<b>4 File Index</b>	<b>7</b>
4.1 File List	7
<b>5 Concept Documentation</b>	<b>9</b>
5.1 algebra::AdditiveGroup Concept Reference	9
5.1.1 Concept definition	9
5.1.2 Detailed Description	9
5.1.3 Semantic requirements	9
5.2 algebra::Commutative Concept Reference	10
5.2.1 Concept definition	10
5.2.2 Detailed Description	10
5.2.3 Semantic requirements	10
5.3 algebra::CommutativeRing Concept Reference	10
5.3.1 Concept definition	10
5.3.2 Detailed Description	10
5.3.3 Sematic requirements	10
5.4 algebra::EuclideanDomain Concept Reference	11
5.4.1 Concept definition	11
5.4.2 Detailed Description	11
5.4.3 Semantic requirements	11
5.5 algebra::Field Concept Reference	11
5.5.1 Concept definition	11
5.5.2 Detailed Description	12
5.5.3 Semantic requirements	12
5.6 algebra::Group Concept Reference	12
5.6.1 Concept definition	12
5.6.2 Detailed Description	12
5.6.3 Semantic requirements	12
5.7 algebra::Ring Concept Reference	13
5.7.1 Concept definition	13
5.7.2 Detailed Description	13
5.7.3 Sematic requirements	13
<b>6 Class Documentation</b>	<b>15</b>
6.1 core::AlgebraHomology< T > Class Template Reference	15

6.1.1 Detailed Description . . . . .	18
6.1.2 Member Function Documentation . . . . .	18
6.1.2.1 betti_numbers() . . . . .	18
6.1.2.2 torsion() . . . . .	19
6.2 complexes::BasicInterval Class Reference . . . . .	19
6.2.1 Detailed Description . . . . .	20
6.2.2 Member Function Documentation . . . . .	20
6.2.2.1 interval() . . . . .	20
6.3 algebra::ChainComplex< T > Class Template Reference . . . . .	21
6.3.1 Detailed Description . . . . .	22
6.3.2 Constructor & Destructor Documentation . . . . .	22
6.3.2.1 ChainComplex() [1/2] . . . . .	22
6.3.2.2 ChainComplex() [2/2] . . . . .	23
6.3.3 Member Function Documentation . . . . .	23
6.3.3.1 boundary() . . . . .	23
6.4 core::CommandLineOptions Class Reference . . . . .	23
6.4.1 Detailed Description . . . . .	26
6.4.2 Member Function Documentation . . . . .	26
6.4.2.1 filename() . . . . .	26
6.4.2.2 help() . . . . .	27
6.4.2.3 homology_to_compute() . . . . .	27
6.4.2.4 latex() . . . . .	27
6.4.2.5 x_bounds() . . . . .	27
6.4.2.6 y_bounds() . . . . .	27
6.4.2.7 z_bounds() . . . . .	27
6.5 core::Complex Class Reference . . . . .	28
6.5.1 Detailed Description . . . . .	29
6.5.2 Member Function Documentation . . . . .	29
6.5.2.1 reduce() . . . . .	29
6.5.2.2 z2_homology() . . . . .	30
6.5.2.3 z3_homology() . . . . .	30
6.5.2.4 z_homology() . . . . .	30
6.6 core::Polymorphic< T >::Concept Struct Reference . . . . .	30
6.6.1 Detailed Description . . . . .	32
6.6.2 Member Function Documentation . . . . .	32
6.6.2.1 clone() . . . . .	32
6.6.2.2 get_address() [1/2] . . . . .	32
6.6.2.3 get_address() [2/2] . . . . .	32
6.7 complexes::CubicalComplex Class Reference . . . . .	33
6.7.1 Detailed Description . . . . .	34
6.7.2 Constructor & Destructor Documentation . . . . .	34
6.7.2.1 CubicalComplex() . . . . .	34

6.7.3 Member Function Documentation	35
6.7.3.1 add()	35
6.7.3.2 add_recursive()	35
6.7.3.3 contains()	35
6.7.3.4 remove()	36
6.7.3.5 simplices()	36
6.8 core::CubicalComplex3D Class Reference	36
6.8.1 Detailed Description	39
6.8.2 Member Function Documentation	39
6.8.2.1 reduce()	39
6.8.2.2 z2_homology()	39
6.8.2.3 z3_homology()	39
6.8.2.4 z_homology()	40
6.9 complexes::CubicalSimplex Class Reference	40
6.9.1 Detailed Description	42
6.9.2 Constructor & Destructor Documentation	43
6.9.2.1 CubicalSimplex()	43
6.9.3 Member Function Documentation	43
6.9.3.1 boundary()	43
6.9.3.2 interval()	43
6.9.3.3 operator<=>()	43
6.9.3.4 point()	44
6.9.4 Friends And Related Symbol Documentation	44
6.9.4.1 product	44
6.10 algebra::DivResult< T > Struct Template Reference	44
6.10.1 Detailed Description	45
6.11 algebra::ExtendedGCDResult Struct Reference	46
6.11.1 Detailed Description	46
6.12 std::formatter< algebra::Integer > Struct Reference	47
6.12.1 Detailed Description	48
6.12.2 Member Function Documentation	48
6.12.2.1 format()	48
6.13 std::formatter< algebra::Matrix< T > > Struct Template Reference	48
6.13.1 Detailed Description	49
6.13.2 Format syntax	49
6.13.3 Example	49
6.13.4 Member Function Documentation	50
6.13.4.1 format()	50
6.14 std::formatter< algebra::ZModP< P > > Struct Template Reference	50
6.14.1 Detailed Description	51
6.15 std::formatter< complexes::BasicInterval > Struct Reference	51
6.15.1 Detailed Description	53

6.15.2 Member Function Documentation	53
6.15.2.1 format()	53
6.16 std::formatter< complexes::CubicalSimplex > Struct Reference	53
6.16.1 Detailed Description	55
6.16.2 Member Function Documentation	56
6.16.2.1 format()	56
6.17 std::hash< complexes::BasicInterval > Struct Reference	56
6.17.1 Detailed Description	56
6.18 std::hash< complexes::CubicalSimplex > Struct Reference	57
6.18.1 Detailed Description	57
6.19 algebra::Homology< T > Struct Template Reference	57
6.19.1 Detailed Description	58
6.19.2 Member Data Documentation	59
6.19.2.1 torsion	59
6.20 core::Homology Class Reference	59
6.20.1 Detailed Description	62
6.20.2 Constructor & Destructor Documentation	62
6.20.2.1 Homology()	62
6.20.3 Member Function Documentation	62
6.20.3.1 betti_numbers()	62
6.20.3.2 select_strategy()	63
6.20.3.3 text()	63
6.20.3.4 torsion()	63
6.21 core::HomologyLatexPrint Class Reference	63
6.21.1 Detailed Description	66
6.21.2 Constructor & Destructor Documentation	66
6.21.2.1 HomologyLatexPrint()	66
6.21.3 Member Function Documentation	66
6.21.3.1 draw()	66
6.22 core::HomologyPrintingStrategy Class Reference	67
6.22.1 Detailed Description	68
6.22.2 Member Function Documentation	68
6.22.2.1 draw()	68
6.23 core::HomologyRawPrint Class Reference	68
6.23.1 Detailed Description	69
6.23.2 Member Function Documentation	69
6.23.2.1 draw()	69
6.24 algebra::Integer Class Reference	70
6.24.1 Detailed Description	71
6.24.2 Member Function Documentation	71
6.24.2.1 euclidean_function()	71
6.25 core::LatexWrapper Class Reference	72

6.25.1 Detailed Description	74
6.25.2 Member Function Documentation	74
6.25.2.1 text()	74
6.26 core::Manager Class Reference	74
6.26.1 Detailed Description	76
6.26.2 Member Function Documentation	76
6.26.2.1 run()	76
6.27 algebra::Matrix< T > Class Template Reference	76
6.27.1 Detailed Description	80
6.27.2 Constructor & Destructor Documentation	80
6.27.2.1 Matrix()	80
6.27.3 Member Function Documentation	80
6.27.3.1 at() [1/2]	80
6.27.3.2 at() [2/2]	81
6.27.3.3 operator*=( )	82
6.27.3.4 operator[]() [1/2]	82
6.27.3.5 operator[]() [2/2]	82
6.28 core::MinecraftCoordinates Struct Reference	83
6.28.1 Detailed Description	83
6.29 core::MinecraftSavefileParser Class Reference	84
6.29.1 Detailed Description	85
6.29.2 Member Function Documentation	85
6.29.2.1 parse()	85
6.30 core::MinecraftSavefileParser_mcSavefileParsers Class Reference	85
6.30.1 Detailed Description	87
6.30.2 Member Function Documentation	87
6.30.2.1 parse()	87
6.31 core::Polymorphic< T >::Model< Object > Struct Template Reference	87
6.31.1 Detailed Description	90
6.31.2 Member Function Documentation	90
6.31.2.1 clone()	90
6.31.2.2 get_address() [1/2]	90
6.31.2.3 get_address() [2/2]	90
6.32 core::Options Class Reference	91
6.32.1 Detailed Description	92
6.32.2 Member Function Documentation	93
6.32.2.1 filename()	93
6.32.2.2 help()	93
6.32.2.3 homology_to_compute()	93
6.32.2.4 latex()	93
6.32.2.5 x_bounds()	93
6.32.2.6 y_bounds()	93

6.32.2.7 <code>z_bounds()</code>	94
6.33 <code>core::Polymorphic&lt; T &gt;</code> Class Template Reference	94
6.33.1 Detailed Description	96
6.34 <code>algebra::RowEchelonFormResult&lt; T &gt;</code> Struct Template Reference	97
6.34.1 Detailed Description	97
6.35 <code>algebra::SkipCorrectnessCheckT</code> Struct Reference	98
6.35.1 Detailed Description	98
6.36 <code>algebra::SmithFormResult&lt; T &gt;</code> Struct Template Reference	98
6.36.1 Detailed Description	99
6.37 <code>core::TextDrawable</code> Class Reference	99
6.37.1 Detailed Description	101
6.37.2 Member Function Documentation	101
6.37.2.1 <code>text()</code>	101
6.38 <code>algebra::ZModP&lt; P &gt;</code> Class Template Reference	102
6.38.1 Detailed Description	104
6.38.2 Member Function Documentation	104
6.38.2.1 <code>euclidean_function()</code>	104
6.39 <code>algebra::ZModP&lt; 2 &gt;</code> Class Reference	105
6.39.1 Detailed Description	107
6.39.2 Member Function Documentation	107
6.39.2.1 <code>euclidean_function()</code>	107
<b>7 File Documentation</b>	<b>109</b>
7.1 <code>mc-homology/algebra/include/algebra/algebraic_concepts.h</code> File Reference	109
7.1.1 Detailed Description	111
7.2 <code>algebraic_concepts.h</code>	111
7.3 <code>mc-homology/algebra/include/algebra/chain_complex.h</code> File Reference	111
7.3.1 Detailed Description	113
7.4 <code>chain_complex.h</code>	113
7.5 <code>mc-homology/algebra/include/algebra/integer.h</code> File Reference	115
7.5.1 Detailed Description	116
7.5.2 Function Documentation	116
7.5.2.1 <code>divide()</code>	116
7.5.2.2 <code>modulo()</code>	116
7.6 <code>integer.h</code>	117
7.7 <code>mc-homology/algebra/include/algebra/matrix.h</code> File Reference	118
7.7.1 Detailed Description	120
7.7.2 Function Documentation	120
7.7.2.1 <code>operator*()</code>	120
7.8 <code>matrix.h</code>	120
7.9 <code>mc-homology/algebra/include/algebra/matrix_algorithms.h</code> File Reference	124
7.9.1 Detailed Description	126



7.9.2 Function Documentation	126
7.9.2.1 row_echelon_form() [1/2]	126
7.9.2.2 row_echelon_form() [2/2]	127
7.9.2.3 smith_form() [1/2]	127
7.9.2.4 smith_form() [2/2]	127
7.10 matrix_algorithms.h	128
7.11 mc-homology/algebra/include/algebra/modulo_fields.h File Reference	129
7.11.1 Detailed Description	131
7.12 modulo_fields.h	131
7.13 mc-homology/algebra/include/algebra/number_theory.h File Reference	132
7.13.1 Detailed Description	135
7.13.2 Function Documentation	135
7.13.2.1 divide()	135
7.13.2.2 extended_gcd()	135
7.13.2.3 inverse_mod()	136
7.13.2.4 is_prime()	136
7.13.2.5 modulo()	136
7.14 number_theory.h	137
7.15 mc-homology/algebra/include/algebra/z2_field.h File Reference	138
7.15.1 Detailed Description	138
7.16 z2_field.h	139
7.17 mc-homology/complexes/include/complexes/compute_chain_complex.h File Reference	140
7.17.1 Detailed Description	141
7.17.2 Function Documentation	141
7.17.2.1 compute_chain_complex()	141
7.18 compute_chain_complex.h	141
7.19 mc-homology/complexes/include/complexes/cubical_complex.h File Reference	142
7.19.1 Detailed Description	144
7.20 cubical_complex.h	144
7.21 mc-homology/complexes/include/complexes/utils.h File Reference	145
7.21.1 Detailed Description	146
7.22 utils.h	146
7.23 mc-homology/core/include/core/algebra_homology.h File Reference	147
7.23.1 Detailed Description	148
7.24 algebra_homology.h	148
7.25 mc-homology/core/include/core/complex.h File Reference	148
7.25.1 Detailed Description	149
7.26 complex.h	150
7.27 mc-homology/core/include/core/cubical_complex_3d.h File Reference	150
7.27.1 Detailed Description	151
7.28 cubical_complex_3d.h	151
7.29 mc-homology/core/include/core/homology.h File Reference	151

7.29.1 Detailed Description	152
7.30 homology.h	153
7.31 mc-homology/core/include/core/homology_printing_strategy.h File Reference	153
7.31.1 Detailed Description	154
7.32 homology_printing_strategy.h	155
7.33 mc-homology/core/include/core/latex_wrapper.h File Reference	155
7.33.1 Detailed Description	156
7.34 latex_wrapper.h	156
7.35 mc-homology/core/include/core/manager.h File Reference	156
7.35.1 Detailed Description	157
7.36 manager.h	157
7.37 mc-homology/core/include/core/options.h File Reference	158
7.37.1 Detailed Description	159
7.38 options.h	159
7.39 mc-homology/core/include/core/parser.h File Reference	160
7.39.1 Detailed Description	161
7.40 parser.h	161
7.41 mc-homology/core/include/core/polymorphic.h File Reference	161
7.41.1 Detailed Description	162
7.42 polymorphic.h	162
7.43 mc-homology/core/include/core/text_drawable.h File Reference	164
7.43.1 Detailed Description	165
7.44 text_drawable.h	165
<b>Index</b>	<b>167</b>

# Chapter 1

## Concept Index

### 1.1 Concepts

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

<a href="#">algebra::AdditiveGroup</a>	
An additive group concept . . . . .	9
<a href="#">algebra::Commutative</a>	
An abelian structure concept . . . . .	10
<a href="#">algebra::CommutativeRing</a>	
An commutative ring concept . . . . .	10
<a href="#">algebra::EuclideanDomain</a>	
An euclidean domain concept . . . . .	11
<a href="#">algebra::Field</a>	
A field concept . . . . .	11
<a href="#">algebra::Group</a>	
A group concept . . . . .	12
<a href="#">algebra::Ring</a>	
A ring concept . . . . .	13



## Chapter 2

# Hierarchical Index

### 2.1 Class Hierarchy

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

complexes::BasicInterval	19
algebra::ChainComplex< T >	21
core::Complex	28
core::CubicalComplex3D	36
core::Polymorphic< T >Concept	30
core::Polymorphic< T >Model< Object >	87
complexes::CubicalComplex	33
complexes::CubicalSimplex	40
algebra::DivResult< T >	44
algebra::ExtendedGCDResult	46
std::formatter	
std::formatter< algebra::Integer >	47
std::formatter< algebra::ZModP< P > >	50
std::formatter< complexes::BasicInterval >	51
std::formatter< complexes::CubicalSimplex >	53
std::formatter< algebra::Matrix< T > >	48
std::hash< complexes::BasicInterval >	56
std::hash< complexes::CubicalSimplex >	57
algebra::Homology< T >	57
core::HomologyPrintingStrategy	67
core::HomologyLatexPrint	63
core::HomologyRawPrint	68
algebra::Integer	70
core::Manager	74
algebra::Matrix< T >	76
core::MinecraftCoordinates	83
core::MinecraftSavefileParser	84
core::MinecraftSavefileParser_mcSavefileParsers	85
core::Options	91
core::CommandLineOptions	23
core::Polymorphic< T >	94
algebra::RowEchelonFormResult< T >	97
algebra::SkipCorrectnessCheckT	98
algebra::SmithFormResult< T >	98

core::TextDrawable . . . . .	99
core::Homology . . . . .	59
core::AlgebraHomology< T > . . . . .	15
core::LatexWrapper . . . . .	72
algebra::ZModP< P > . . . . .	102
algebra::ZModP< 2 > . . . . .	105

## Chapter 3

# Class Index

### 3.1 Class List

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

<a href="#">core::AlgebraHomology&lt; T &gt;</a>	Concrete subclass of <a href="#">Homology</a> , based on algebra library . . . . .	15
<a href="#">complexes::BasicInterval</a>	A struct representing an interval of length 0 or 1 . . . . .	19
<a href="#">algebra::ChainComplex&lt; T &gt;</a>	Class representing a chain complex with coefficients $\mathbb{T}$ . . . . .	21
<a href="#">core::CommandLineOptions</a>	A class for storing user's options from the commandline . . . . .	23
<a href="#">core::Complex</a>	Interface for complexes . . . . .	28
<a href="#">core::Polymorphic&lt; T &gt;::Concept</a>	Base class used for type erasure . . . . .	30
<a href="#">complexes::CubicalComplex</a>	Class representing a cubical complex . . . . .	33
<a href="#">core::CubicalComplex3D</a>	A class representing a cubical complex in 3D space . . . . .	36
<a href="#">complexes::CubicalSimplex</a>	A class representing a single cubical simplex in a complex . . . . .	40
<a href="#">algebra::DivResult&lt; T &gt;</a>	Result struct for the division operation . . . . .	44
<a href="#">algebra::ExtendedGCDResult</a>	Result struct for the extended gcd algorithm . . . . .	46
<a href="#">std::formatter&lt; algebra::Integer &gt;</a>	Formatter for Integer type . . . . .	47
<a href="#">std::formatter&lt; algebra::Matrix&lt; T &gt; &gt;</a>	Formatter for the <code>Matrix</code> type . . . . .	48
<a href="#">std::formatter&lt; algebra::ZModP&lt; P &gt; &gt;</a>	Formatter for ZModP type . . . . .	50
<a href="#">std::formatter&lt; complexes::BasicInterval &gt;</a>	Formatter for BasicInterval type . . . . .	51
<a href="#">std::formatter&lt; complexes::CubicalSimplex &gt;</a>	Formatter for CubicalSimplex type . . . . .	53
<a href="#">std::hash&lt; complexes::BasicInterval &gt;</a>	Std::hash specialization for Interval . . . . .	56
<a href="#">std::hash&lt; complexes::CubicalSimplex &gt;</a>	Std::hash specialization for CubicalSimplex . . . . .	57

<a href="#">algebra::Homology&lt; T &gt;</a>	
<a href="#">Homology</a> of a chain complex	57
<a href="#">core::Homology</a>	
A homology interface	59
<a href="#">core::HomologyLatexPrint</a>	
Prints a homology group in Latex syntax	63
<a href="#">core::HomologyPrintingStrategy</a>	
A strategy for printing a homology group	67
<a href="#">core::HomologyRawPrint</a>	
A strategy for basic string representation of homology	68
<a href="#">algebra::Integer</a>	
Class of Integers	70
<a href="#">core::LatexWrapper</a>	
A decorator that wraps a text drawable object in a latex document	72
<a href="#">core::Manager</a>	
Main manager for the program	74
<a href="#">algebra::Matrix&lt; T &gt;</a>	
A <a href="#">Matrix</a> class	76
<a href="#">core::MinecraftCoordinates</a>	
A struct containing coordinates in a Minecraft world	83
<a href="#">core::MinecraftSavefileParser</a>	
An interface for Minecraft savefile parser	84
<a href="#">core::MinecraftSavefileParser_mcSavefileParsers</a>	
A Minecraft savefile parser based on <a href="https://github.com/TCA166/mcSavefileParsers.git">https://github.com/TCA166/mcSavefileParsers.git</a>	85
<a href="#">core::Polymorphic&lt; T &gt;::Model&lt; Object &gt;</a>	
Concrete class used for type erasure	87
<a href="#">core::Options</a>	
A class for storing user options	91
<a href="#">core::Polymorphic&lt; T &gt;</a>	
A polymorphic class wrapper with value semantics	94
<a href="#">algebra::RowEchelonFormResult&lt; T &gt;</a>	
Result struct for the row echelon algorithm	97
<a href="#">algebra::SkipCorrectnessCheckT</a>	
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	98
<a href="#">algebra::SmithFormResult&lt; T &gt;</a>	
Result struct for the smith algorithm	98
<a href="#">core::TextDrawable</a>	
Marks a class text drawable - it can be represented as a string	99
<a href="#">algebra::ZModP&lt; P &gt;</a>	
Field of integers modulo P	102
<a href="#">algebra::ZModP&lt; 2 &gt;</a>	
Template specialization for P == 2	105



## Chapter 4

# File Index

### 4.1 File List

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

mc-homology/algebra/include/algebra/ <a href="#">algebraic_concepts.h</a>	
A file defining multiple concepts for algebraic structures . . . . .	109
mc-homology/algebra/include/algebra/ <a href="#">chain_complex.h</a>	
A file containing chain complex and homology implementations . . . . .	111
mc-homology/algebra/include/algebra/ <a href="#">integer.h</a>	
A file containing Integer implementation satisfying algebraic concepts . . . . .	115
mc-homology/algebra/include/algebra/ <a href="#">matrix.h</a>	
A file containing matrix implementation . . . . .	118
mc-homology/algebra/include/algebra/ <a href="#">matrix_algorithms.h</a>	
A file containing selected matrix algorithms . . . . .	124
mc-homology/algebra/include/algebra/ <a href="#">modulo_fields.h</a>	
A file containing implementation of fields of integers mod P . . . . .	129
mc-homology/algebra/include/algebra/ <a href="#">number_theory.h</a>	
A file containing implementations of selected number theoretic algorithms . . . . .	132
mc-homology/algebra/include/algebra/ <a href="#">z2_field.h</a>	
A file containing optimized implementation of Z2 field . . . . .	138
mc-homology/complexes/include/complexes/ <a href="#">compute_chain_complex.h</a>	
A file containing algorithms for transforming complexes into chain complexes . . . . .	140
mc-homology/complexes/include/complexes/ <a href="#">cubical_complex.h</a>	
A file containing the CubicalComplex class . . . . .	142
mc-homology/complexes/include/complexes/ <a href="#">utils.h</a>	
Contains auxillary classes and functions . . . . .	145
mc-homology/core/include/core/ <a href="#">algebra_homology.h</a>	
File containing concrete classes deriving from Homology . . . . .	147
mc-homology/core/include/core/ <a href="#">complex.h</a>	
A file containing interface for the Complex class . . . . .	148
mc-homology/core/include/core/ <a href="#">cubical_complex_3d.h</a>	
A file containing a concrete class CubicalComplex3D . . . . .	150
mc-homology/core/include/core/ <a href="#">homology.h</a>	
A file containing a Homology interface . . . . .	151
mc-homology/core/include/core/ <a href="#">homology_printing_strategy.h</a>	
File containing strategies for printing homology . . . . .	153
mc-homology/core/include/core/ <a href="#">latex_wrapper.h</a>	
Turns a TextDrawable into a simple latex document . . . . .	155
mc-homology/core/include/core/ <a href="#">manager.h</a>	
A file containing main logic of the program . . . . .	156

mc-homology/core/include/core/ <a href="#">options.h</a>	
File containing an Options class for storing user options . . . . .	158
mc-homology/core/include/core/ <a href="#">parser.h</a>	
A file containing a minecraft savefile parser . . . . .	160
mc-homology/core/include/core/ <a href="#">polymorphic.h</a>	
A polymorphic type wrapper with value sematincs . . . . .	161
mc-homology/core/include/core/ <a href="#">text_drawable.h</a>	
A file containing a TextDrawable interface, for classes that can be drawn on screen in the form of text . . . . .	164

## 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 Semantic 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  $f(a) \leq f(b)$

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  $y \neq T::\text{zero}()$

1.  $x = \text{divide}(x, y).\text{quotient} * y + \text{divide}(x, y).\text{remainder}$
2.  $x.\text{euclidean\_function}() \leq (x * y).\text{euclidean\_function}()$

## 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 mathematical definition of a field, where addition is denoted by `+` and multiplication is denoted by `*`. Additionally, there is a division 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

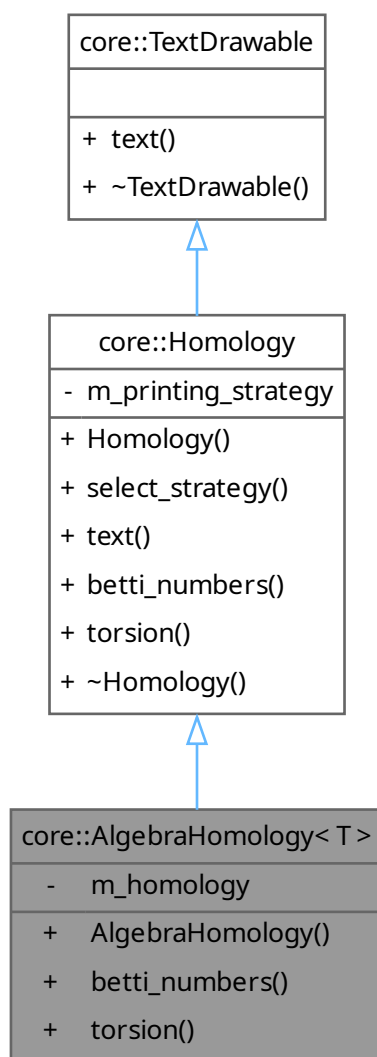
# Class Documentation

### 6.1 `core::AlgebraHomology< T >` Class Template Reference

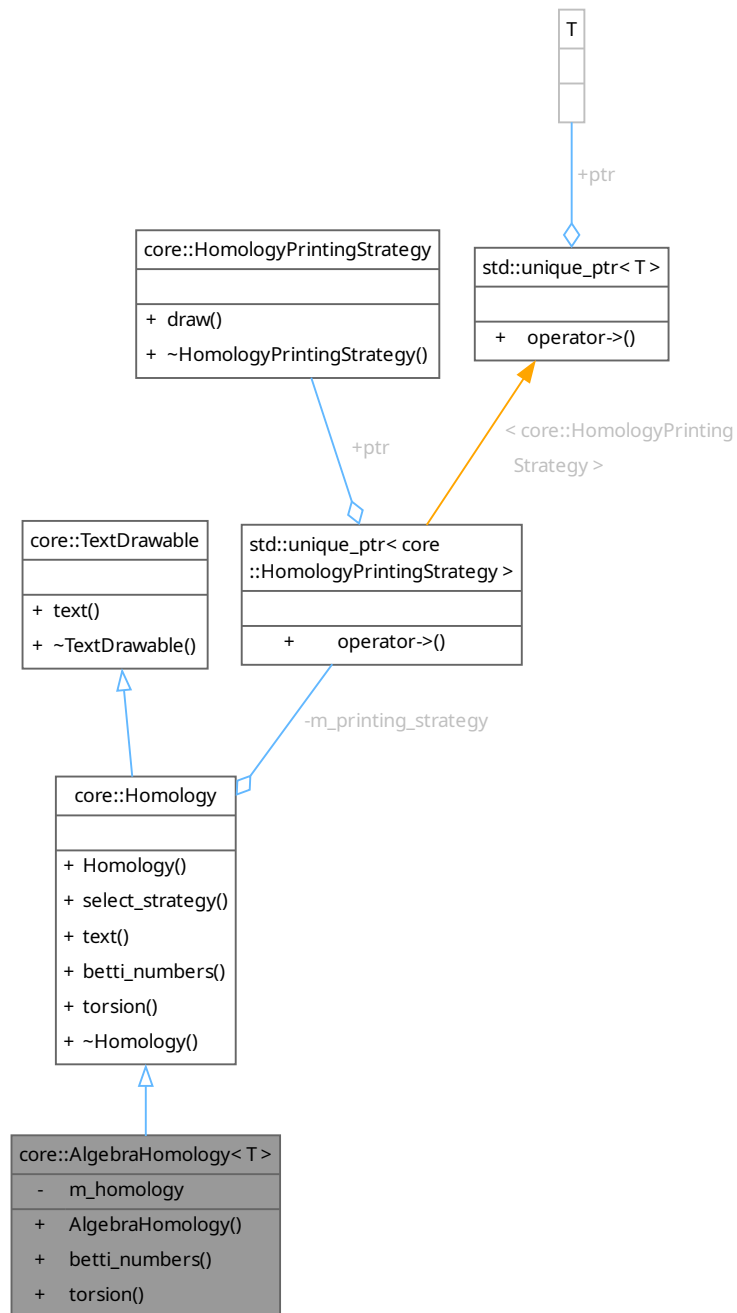
Concrete subclass of [Homology](#), based on algebra library.

```
#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 representation 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.*

## Private Attributes

- `algebra::Homology< T > m_homology`  
*The inner representation of the homology.*

## 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 representation 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:

complexes::BasicInterval
- m_left
- m_full
+ BasicInterval()
+ hash()
+ operator==()
+ left()
+ right()
+ is_trivial()
+ point()
+ interval()

## Public Member Functions

- **BasicInterval** ()  
*Returns an singleton of 0.*
- `std::size_t` **hash** () const  
*Computer a hash of the interval.*
- `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.*

## Private Attributes

- `int` **m\_left** = 0  
*Left bound of the interval.*
- `bool` **m\_full** = false  
*Marker, if the interval is full or trivial (degenerate).*

## 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  $\text{right} - \text{left} < 2$

## 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

<a href="#">left</a>	end of the interval
----------------------	---------------------

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

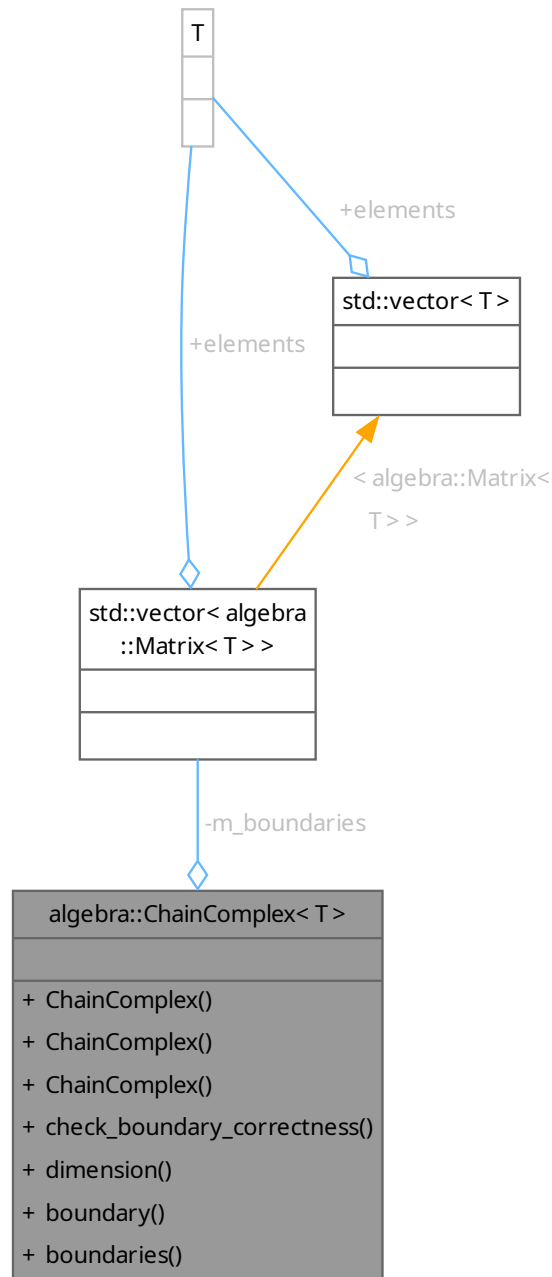
- mc-homology/complexes/include/complexes/[cubical\\_complex.h](#)
- mc-homology/complexes/src/cubical\_complex.cpp

## 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.*

## Private Attributes

- `std::vector< Matrix< T > > m_boundaries`  
*A 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  $(C_d, \dots, C_2, C_1, C_0, C_{(-1)})$  and linear operators  $B_n : C_n \rightarrow C_{(n-1)}$  satisfying  $B_n \circ B_{(n+1)} = 0$  (the chain complex condition).

### 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

<code>boundaries</code>	The range containing the boundary operators
-------------------------	---



### 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

<a href="#">boundaries</a>	The range containing the boundary operators
----------------------------	---

## 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

<i>dim</i>	Dimension of the boundary operator
------------	------------------------------------

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

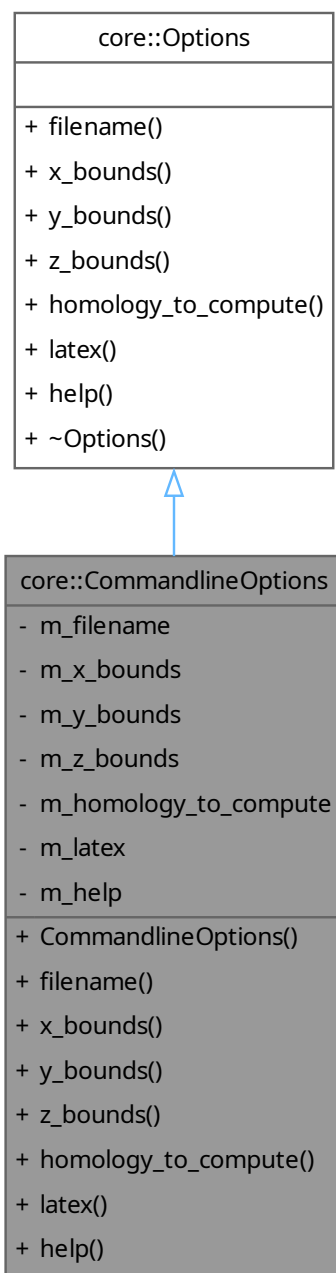
- `mc-homology/algebra/include/algebra/chain\_complex.h`

## 6.4 core::CommandLineOptions Class Reference

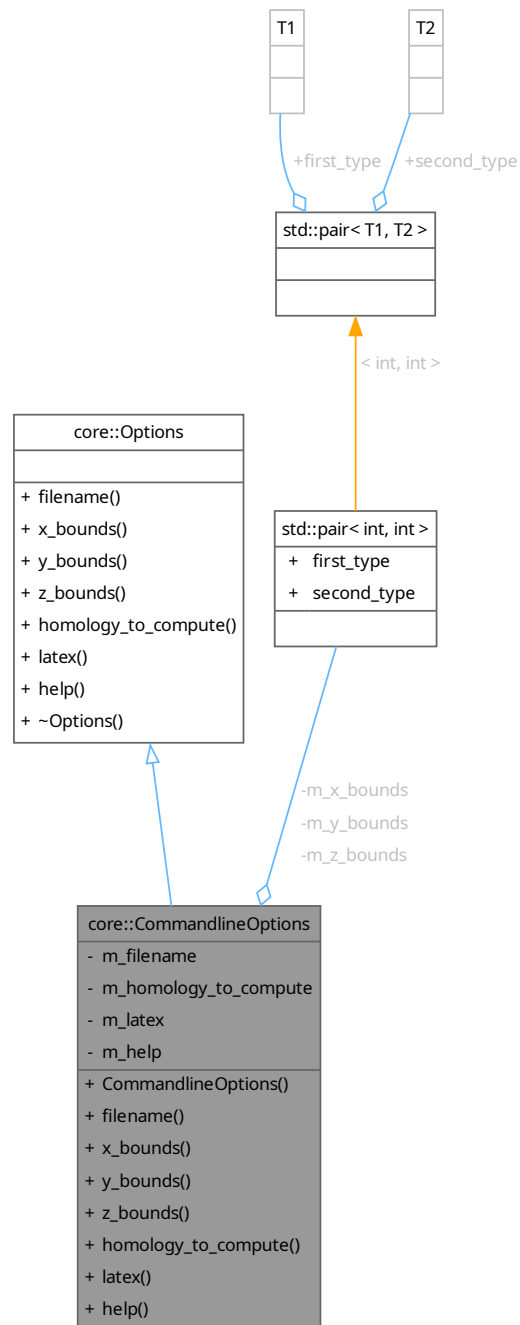
A class for storing user's options from the commandline.

```
#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 z 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*

## Private Attributes

- `std::filesystem::path m\_filename = ""`  
*Filename of the save file.*
- `std::pair< int, int > m\_x\_bounds = {0, 0}`  
*x bounds in the save file*
- `std::pair< int, int > m\_y\_bounds = {0, 0}`  
*y bounds in the save file*
- `std::pair< int, int > m\_z\_bounds = {0, 0}`  
*z bounds in the save file*
- `HomologyChoice m\_homology\_to\_compute = HomologyChoice::Z2`  
*Type of homology to compute.*
- `bool m\_latex = false`  
*Flag whether to print latex syntax.*
- `bool m\_help = false`  
*Flag whether to print help.*

## 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 z axis.

Implements [core::Options](#).

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

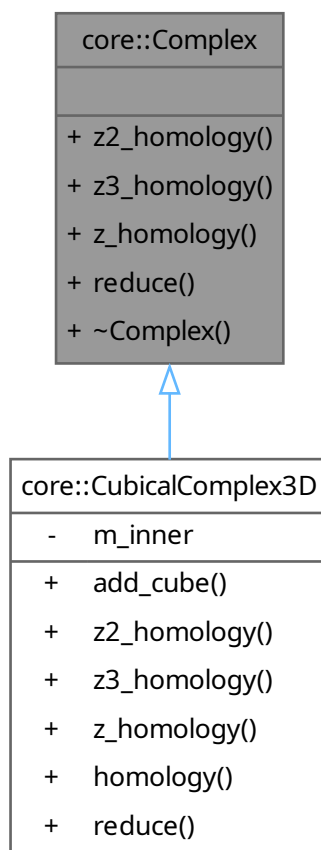
- mc-homology/core/include/core/[options.h](#)
- mc-homology/core/src/[options.cpp](#)

## 6.5 core::Complex Class Reference

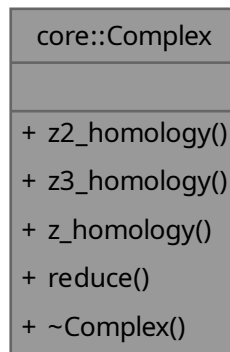
Interface for complexes.

```
#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:

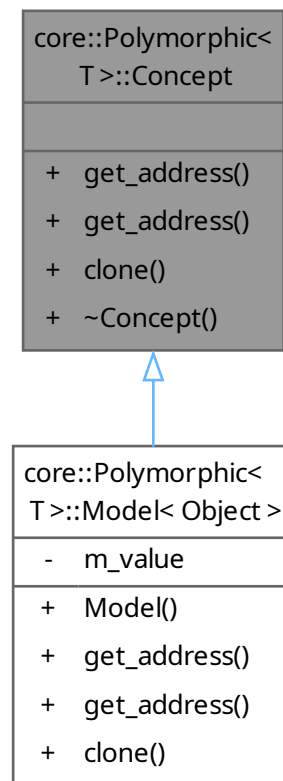
- [mc-homology/core/include/core/complex.h](#)
- [mc-homology/core/src/complex.cpp](#)

## 6.6 `core::Polymorphic< T >::Concept Struct Reference`

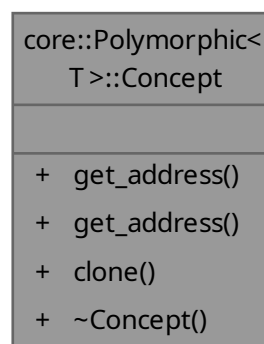
Base class used for type erasure.



Inheritance diagram for core::Polymorphic< T >::Concept:



Collaboration diagram for core::Polymorphic< T >::Concept:



## Public Member Functions

- virtual constexpr T const \* [get\\_address](#) () const noexcept=0  
*Returns address of the stored object.*
- virtual constexpr T \* [get\\_address](#) () noexcept=0  
*Returns address of the stored object.*
- virtual constexpr std::unique\_ptr< [Concept](#) > [clone](#) () const =0  
*Returns a copy of the stored object.*
- virtual constexpr ~**Concept** ()=default  
*Virtual destructor.*

### 6.6.1 Detailed Description

```
template<class T>
struct core::Polymorphic< T >::Concept
```

Base class used for type erasure.

### 6.6.2 Member Function Documentation

#### 6.6.2.1 clone()

```
template<class T>
virtual constexpr std::unique_ptr< Concept > core::Polymorphic< T >::Concept::clone () const
[constexpr], [pure virtual]
```

Returns a copy of the stored object.

Implemented in [core::Polymorphic< T >::Model< Object >](#).

#### 6.6.2.2 get\_address() [1/2]

```
template<class T>
virtual constexpr T const * core::Polymorphic< T >::Concept::get\_address () const [constexpr],
[pure virtual], [noexcept]
```

Returns address of the stored object.

Implemented in [core::Polymorphic< T >::Model< Object >](#).

#### 6.6.2.3 get\_address() [2/2]

```
template<class T>
virtual constexpr T * core::Polymorphic< T >::Concept::get\_address () [constexpr], [pure
virtual], [noexcept]
```

Returns address of the stored object.

Implemented in [core::Polymorphic< T >::Model< Object >](#).

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

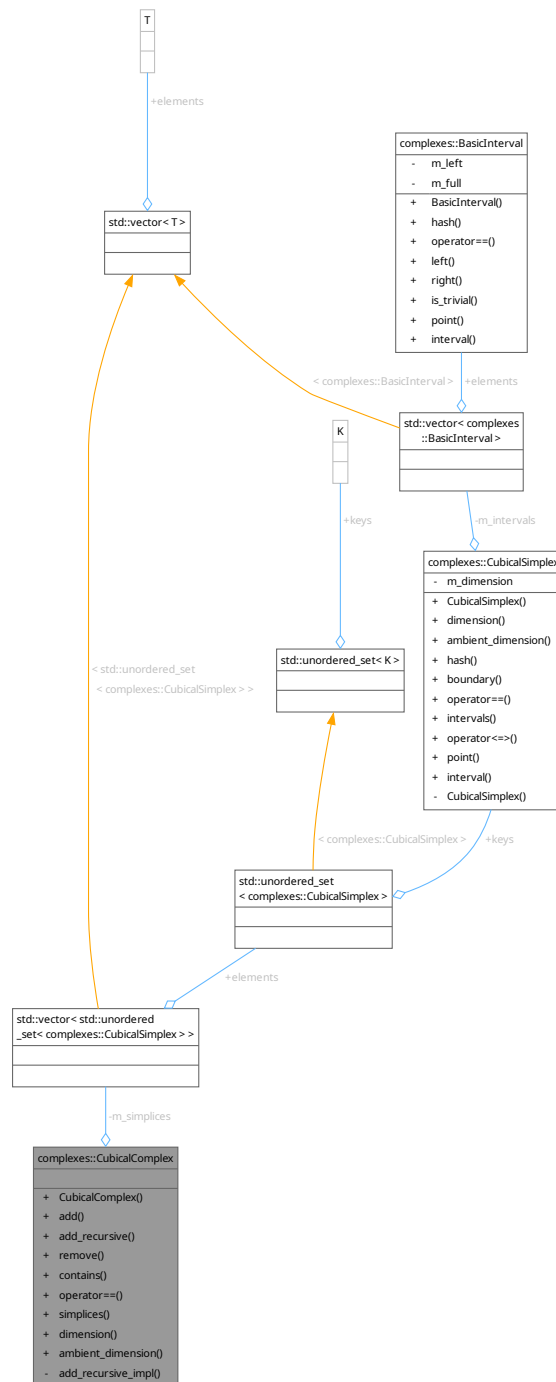
- mc-homology/core/include/core/[polymorphic.h](#)

## 6.7 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.*

## Private Member Functions

- void [add\\_recursive\\_impl](#) ([CubicalSimplex](#) simplex)  
*Implementation of the recursive add algorithm.*

## Private Attributes

- std::vector< std::unordered\_set< [CubicalSimplex](#) > > [m\\_simplices](#)  
*Simplexes in the complex stored by dimension.*

## 6.7.1 Detailed Description

Class representing a cubical complex.

A class representing a topological space constructed from cubical simplexes.

## 6.7.2 Constructor & Destructor Documentation

### 6.7.2.1 CubicalComplex()

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

Default constructor for the [CubicalComplex](#).

Creates an empty complex

## 6.7.3 Member Function Documentation

### 6.7.3.1 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

<i>simplex</i>	Simplex to add
----------------	----------------

#### Returns

true, if simplex has been successfully added, false otherwise

### 6.7.3.2 add\_recursive()

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

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

#### Parameters

<i>simplex</i>	Simplex to add
----------------	----------------

### 6.7.3.3 contains()

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

Checks, if a cubical complex contains simplex.

#### Parameters

<i>simplex</i>	simplex to test
----------------	-----------------

#### Returns

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

#### 6.7.3.4 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

<i>simplex</i>	Simplex to remove
----------------	-------------------

##### Returns

true, if removal was successfull, false otherwise

#### 6.7.3.5 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:

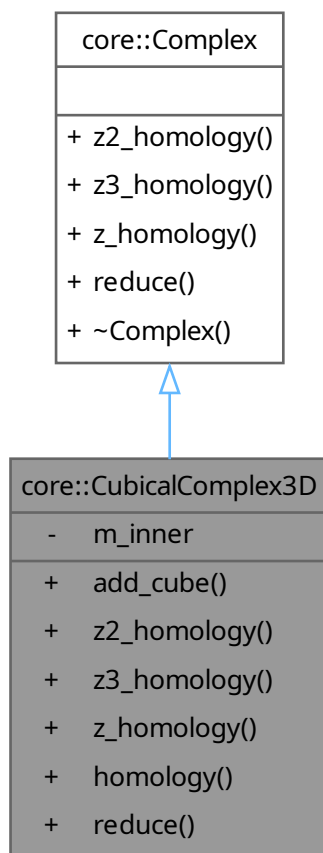
- mc-homology/complexes/include/complexes/[cubical\\_complex.h](#)
- mc-homology/complexes/src/cubical\_complex.cpp

## 6.8 core::CubicalComplex3D Class Reference

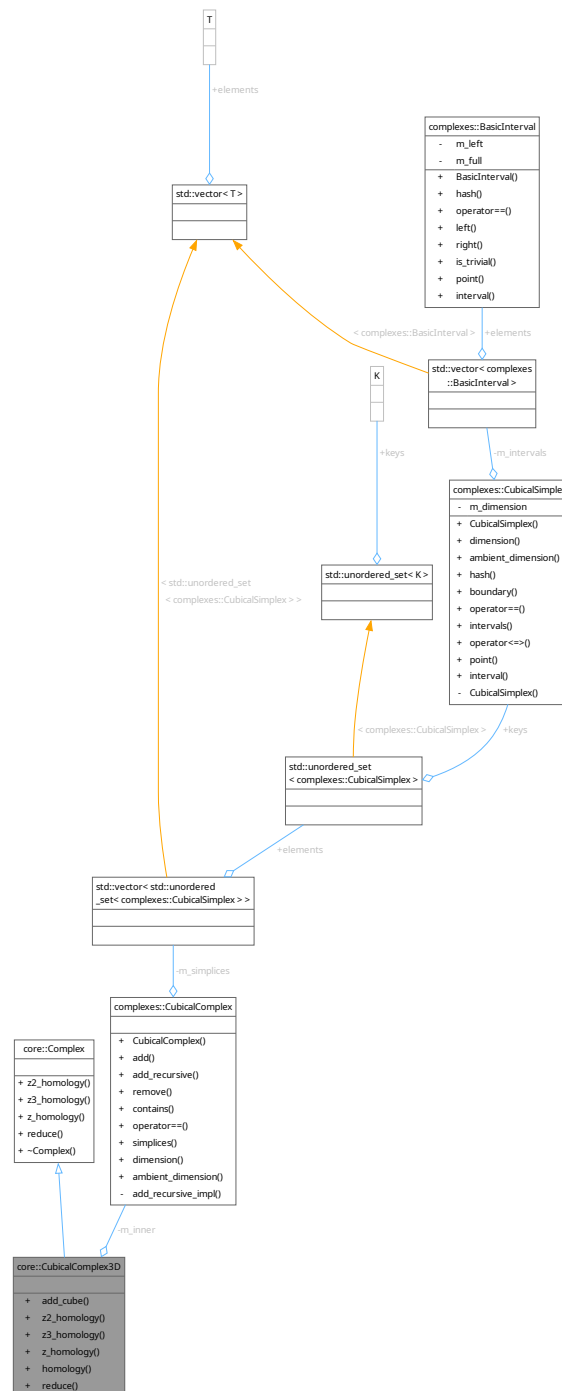
A class representing a cubical complex in 3D space.

```
#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.*

## Private Attributes

- `complexes::CubicalComplex m_inner`  
*Inner representation of the cubical complex.*

## 6.8.1 Detailed Description

A class representing a cubical complex in 3D space.

## 6.8.2 Member Function Documentation

### 6.8.2.1 reduce()

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

Decreases the complex's size without changing its homology.

Implements [core::Complex](#).

### 6.8.2.2 z2\_homology()

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

Computes Z2 homology of the complex.

Implements [core::Complex](#).

### 6.8.2.3 z3\_homology()

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

Computes Z3 homology of the complex.

Implements [core::Complex](#).

#### 6.8.2.4 `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:

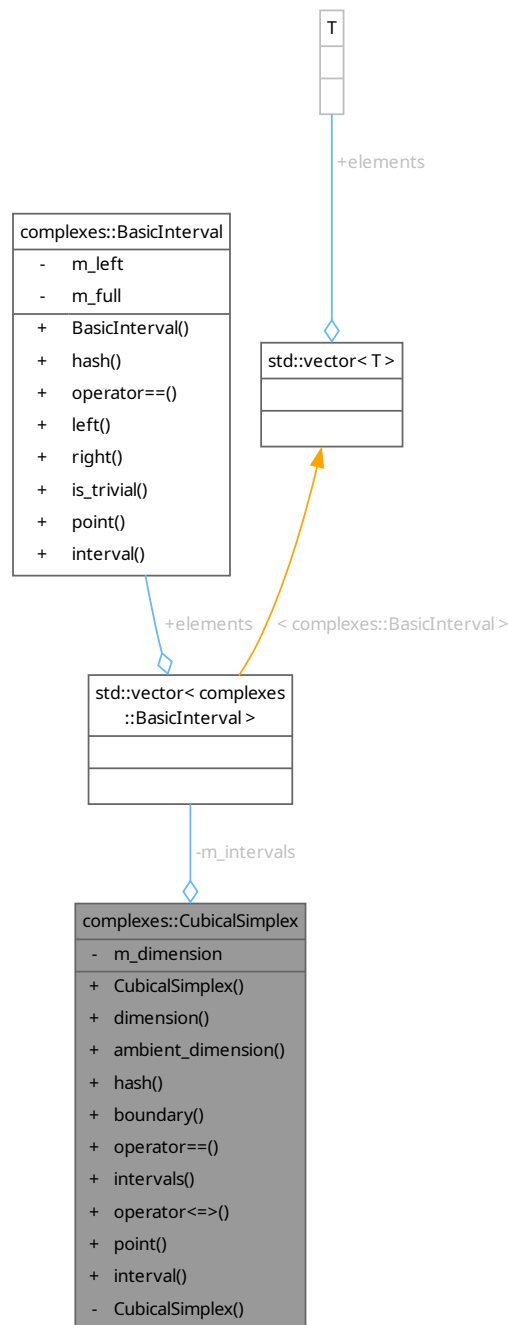
- [mc-homology/core/include/core/cubical\\_complex\\_3d.h](#)
- [mc-homology/core/src/cubical\\_complex\\_3d.cpp](#)

## 6.9 `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.*

### Private Member Functions

- `CubicalSimplex ()`  
*Creates an empty simplex.*

### Private Attributes

- `std::vector< BasicInterval > m_intervals = {}`  
*Stores intervals defining a simplex.*
- `std::size_t m_dimension = 0`  
*Stores the number of non-trivial intervals.*

### Friends

- `CubicalSimplex product (CubicalSimplex const &s1, CubicalSimplex const &s2)`  
*Creates a cubical simplex by joining together two simplices.*

## 6.9.1 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.

## 6.9.2 Constructor & Destructor Documentation

### 6.9.2.1 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

<code>intervals</code>	A nonempty vector of intervals
------------------------	--------------------------------

## 6.9.3 Member Function Documentation

### 6.9.3.1 boundary()

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

Returns the boundary of a simplex.

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

### 6.9.3.2 interval()

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

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

#### Parameters

<code>left</code>	Left bound of the interval
-------------------	----------------------------

### 6.9.3.3 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  $a < b$

#### 6.9.3.4 point()

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

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

##### Parameters

$p$	The point
-----	-----------

### 6.9.4 Friends And Related Symbol Documentation

#### 6.9.4.1 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:

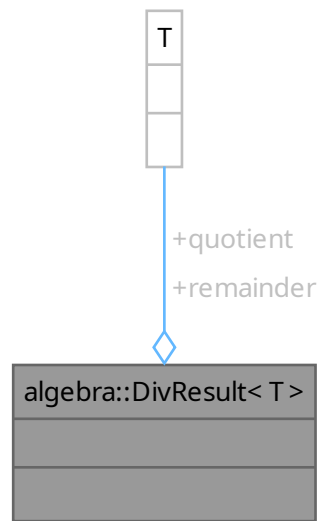
- mc-homology/complexes/include/complexes/[cubical\\_complex.h](#)
- mc-homology/complexes/src/cubical\_complex.cpp

## 6.10 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.*

### 6.10.1 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](#)

## 6.11 algebra::ExtendedGCDResult Struct Reference

Result struct for the extended gcd algorithm.

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

Collaboration diagram for algebra::ExtendedGCDResult:

algebra::ExtendedGCDResult	
+	g
+	x
+	y

### 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.*

### 6.11.1 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](#)

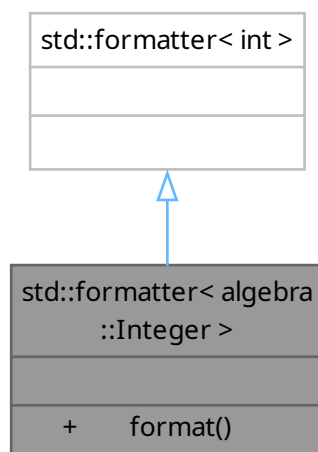


## 6.12 std::formatter< algebra::Integer > Struct Reference

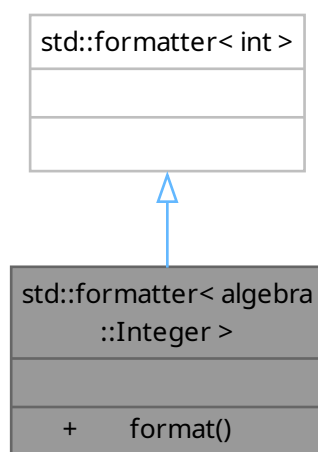
Formatter for Integer type.

```
#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.*

### 6.12.1 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`.

### 6.12.2 Member Function Documentation

#### 6.12.2.1 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`

## 6.13 `std::formatter< algebra::Matrix< T > >` Struct Template Reference

Formatter for the `Matrix` type.

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

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

<code>std::formatter&lt; algebra::Matrix&lt; T &gt; &gt;</code>
- <code>multi_line</code>
- <code>coefficient_formatter</code>
+ <code>parse()</code>
+ <code>format()</code>

## 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.*

## Private Attributes

- bool **multi\_line** = false  
*Flag whether to print the matrix in multiple lines.*
- std::formatter< T > **coefficient\_formatter**  
*Formatter for the coefficients.*

### 6.13.1 Detailed Description

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

Formatter for the `Matrix` type.

Implementation of the formatter for the `Matrix` type.

### 6.13.2 Format syntax

1. `:[-|#][&lt;coefficient_format_string>]`

where `&lt;coefficient_format_string>` 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.

### 6.13.3 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

### 6.13.4 Member Function Documentation

#### 6.13.4.1 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:

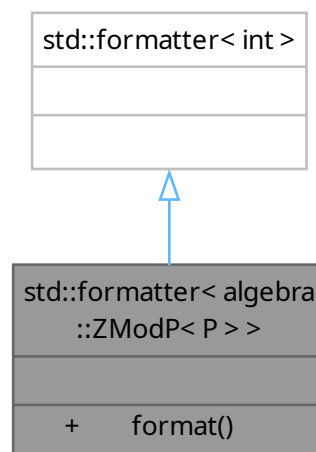
- [mc-homology/algebra/include/algebra/matrix.h](#)

## 6.14 std::formatter< algebra::ZModP< P > > Struct Template Reference

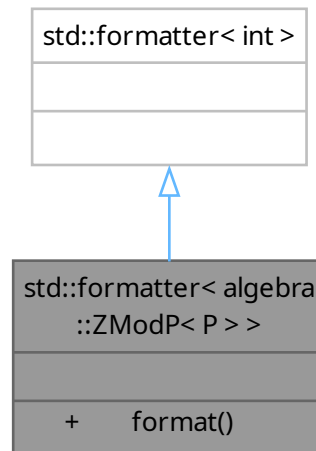
Formatter for ZModP type.

```
#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.*

### 6.14.1 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:

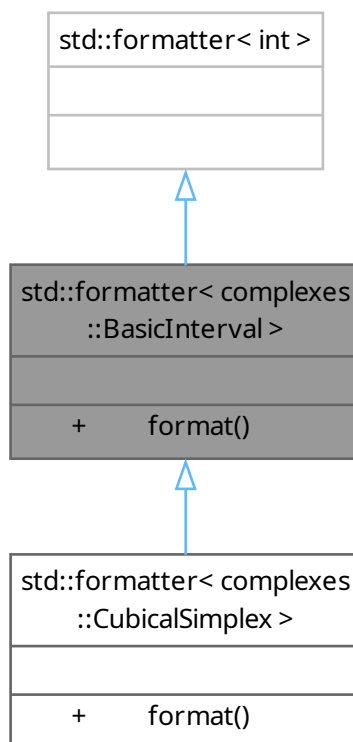
- `mc-homology/algebra/include/algebra/modulo_fields.h`

## 6.15 `std::formatter< complexes::BasicInterval >` Struct Reference

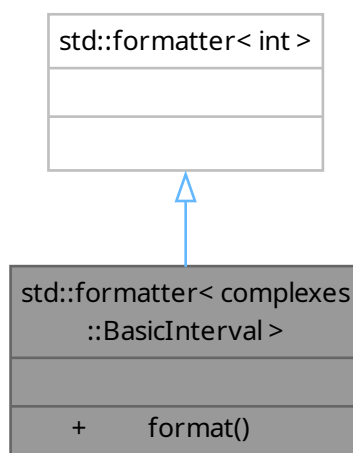
Formatter for BasicInterval type.

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

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



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.*

### 6.15.1 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`.

### 6.15.2 Member Function Documentation

#### 6.15.2.1 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:

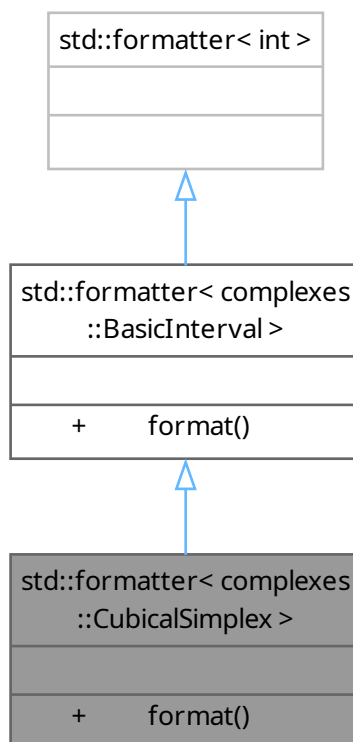
- [mc-homology/complexes/include/complexes/cubical\\_complex.h](#)

## 6.16 std::formatter< complexes::CubicalSimplex > Struct Reference

Formatter for CubicalSimplex type.

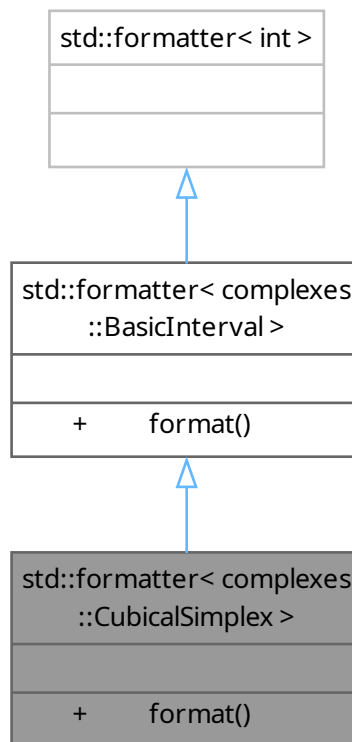
```
#include <cubical_complex.h>
```

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





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



### 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.*

#### 6.16.1 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`.

## 6.16.2 Member Function Documentation

### 6.16.2.1 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:

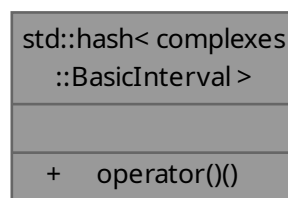
- [mc-homology/complexes/include/complexes/cubical\\_complex.h](#)

## 6.17 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.*

### 6.17.1 Detailed Description

std::hash specialization for Interval

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

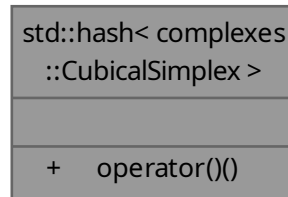
- [mc-homology/complexes/include/complexes/cubical\\_complex.h](#)
- [mc-homology/complexes/src/cubical\\_complex.cpp](#)

## 6.18 `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.*

### 6.18.1 Detailed Description

`std::hash` specialization for `CubicalSimplex`

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

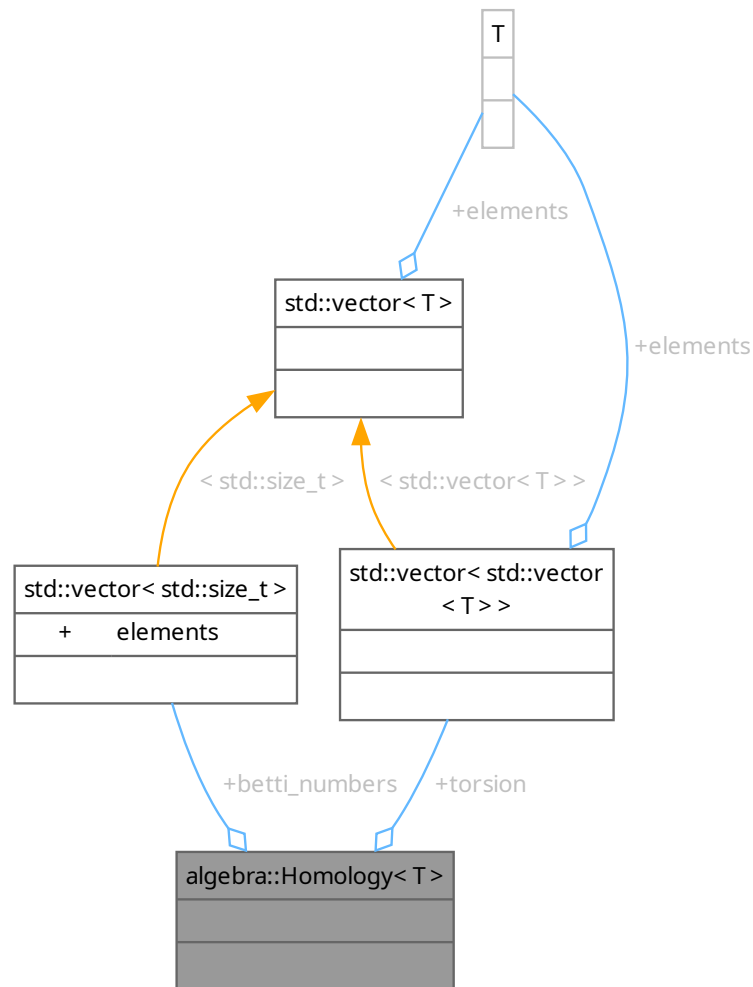
- `mc-homology/complexes/include/complexes/cubical_complex.h`
- `mc-homology/complexes/src/cubical_complex.cpp`

## 6.19 `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.*

## 6.19.1 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.

## 6.19.2 Member Data Documentation

### 6.19.2.1 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:

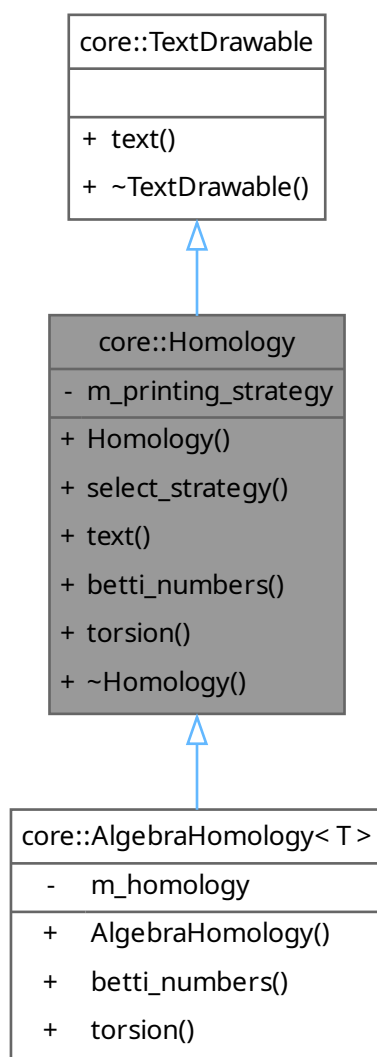
- mc-homology/algebra/include/algebra/[chain\\_complex.h](#)

## 6.20 core::Homology Class Reference

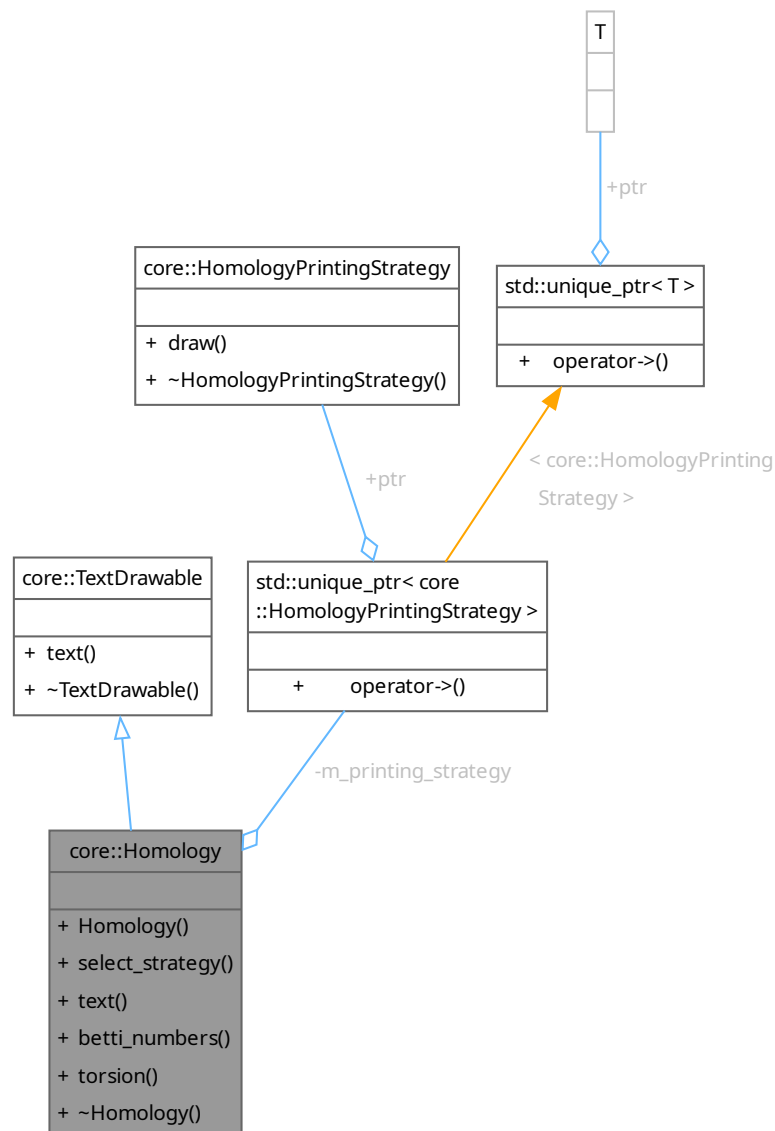
A homology interface.

```
#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.*

## Private Attributes

- `std::unique_ptr< HomologyPrintingStrategy > m_printing_strategy`  
*Printing strategy used to print the homology.*

## 6.20.1 Detailed Description

A homology interface.

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

## 6.20.2 Constructor & Destructor Documentation

### 6.20.2.1 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

<i>printing_strategy</i>	A strategy used with printing
--------------------------	-------------------------------

## 6.20.3 Member Function Documentation

### 6.20.3.1 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 >](#).



### 6.20.3.2 select\_strategy()

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

Select a new `printing_strategy`.

#### Parameters

<code><i>printing_strategy</i></code>	A strategy used with printing
---------------------------------------	-------------------------------

### 6.20.3.3 text()

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

Outputs the test form of the homology using given strategy.

Implements [core::TextDrawable](#).

### 6.20.3.4 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:

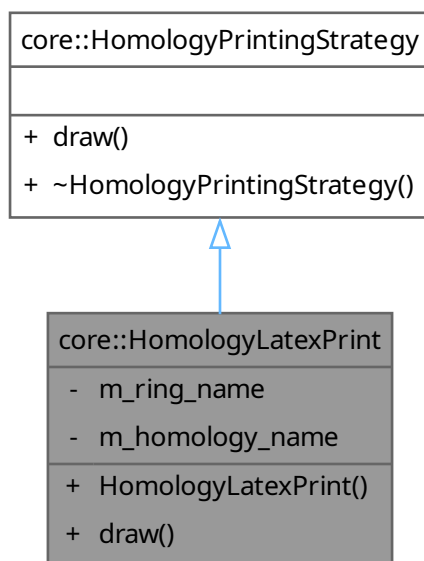
- `mc-homology/core/include/core/homology.h`
- `mc-homology/core/src/homology.cpp`

## 6.21 core::HomologyLatexPrint Class Reference

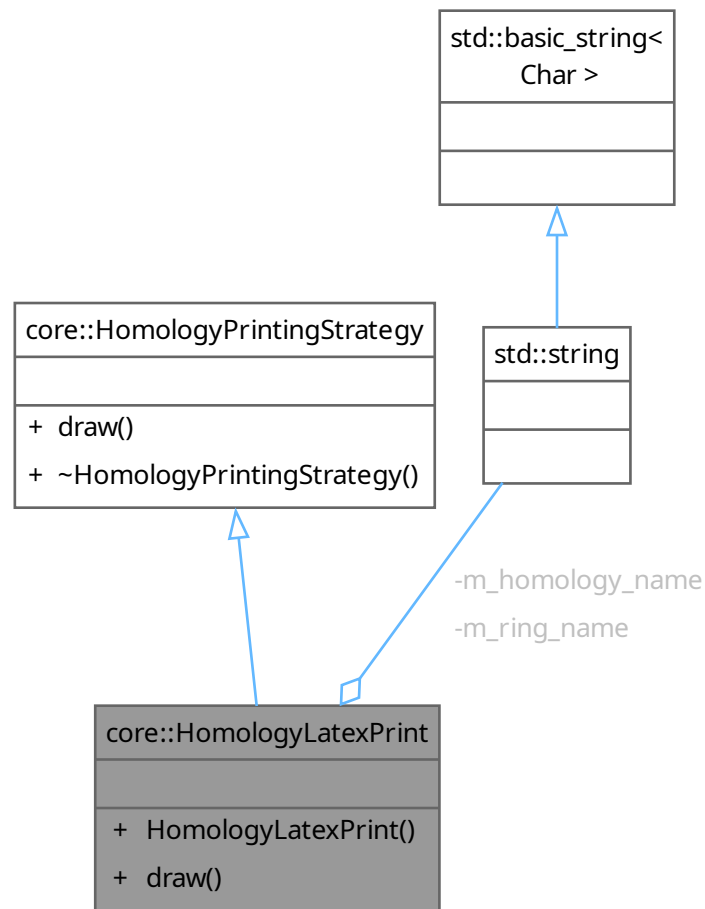
Prints a homology group in Latex syntax.

```
#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.*

### Private Attributes

- std::string **m\_ring\_name**  
*Name of the ring to print.*
- std::string **m\_homology\_name**  
*Name of the homology group.*

### 6.21.1 Detailed Description

Prints a homology group in Latex syntax.

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

### 6.21.2 Constructor & Destructor Documentation

#### 6.21.2.1 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

<i>ring_name</i>	Name for the coefficient ring
<i>homology_name</i>	Name for the homology group

### 6.21.3 Member Function Documentation

#### 6.21.3.1 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:

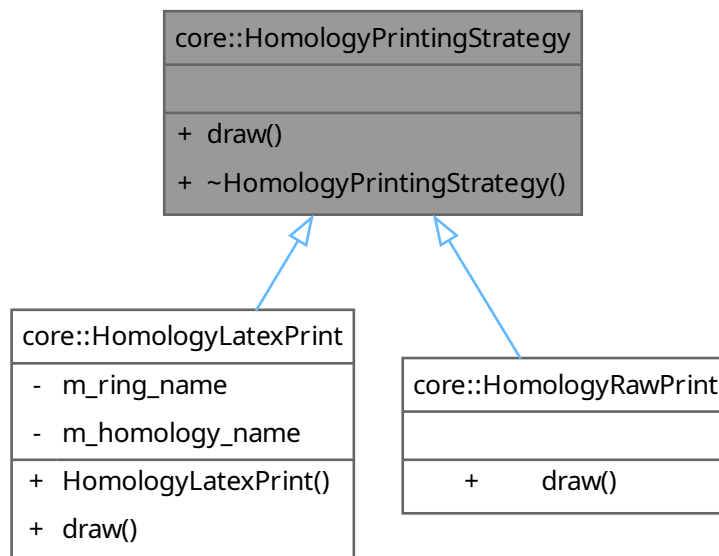
- [mc-homology/core/include/core/homology\\_printing\\_strategy.h](#)
- [mc-homology/core/src/homology\\_printing\\_strategy.cpp](#)

## 6.22 core::HomologyPrintingStrategy Class Reference

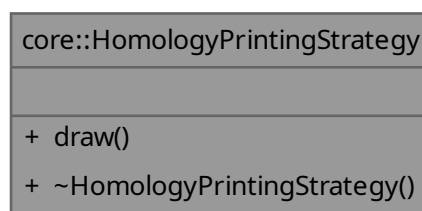
A strategy for printing a homology group.

```
#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.*

### 6.22.1 Detailed Description

A strategy for printing a homology group.

### 6.22.2 Member Function Documentation

#### 6.22.2.1 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:

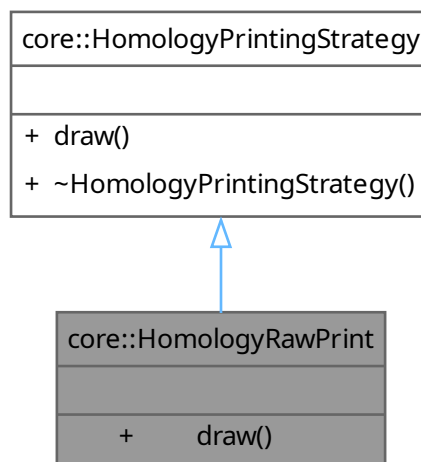
- [mc-homology/core/include/core/homology\\_printing\\_strategy.h](#)
- [mc-homology/core/src/homology\\_printing\\_strategy.cpp](#)

## 6.23 core::HomologyRawPrint Class Reference

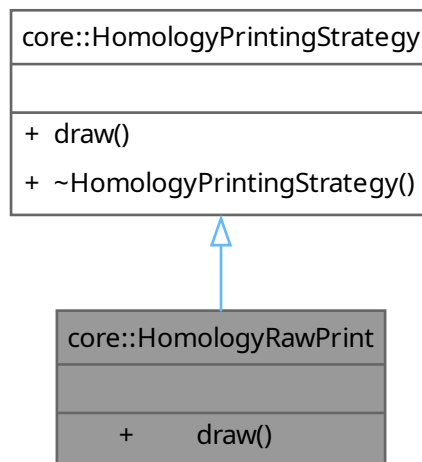
A strategy for basic string representation of homology.

```
#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.*

## 6.23.1 Detailed Description

A strategy for basic string representation of homology.

## 6.23.2 Member Function Documentation

### 6.23.2.1 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`

## 6.24 algebra::Integer Class Reference

Class of Integers.

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

Collaboration diagram for algebra::Integer:

algebra::Integer
- m_inner_representation
+ Integer()
+ Integer()
+ operator int()
+ operator==()
+ operator<=>()
+ operator+=()
+ operator-=()
+ operator+()
+ operator-()
+ operator*=(
+ euclidean_function()
+ zero()
+ one()

### 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.*

### Private Attributes

- int **m\_inner\_representation** = 0  
*Inner representation of the integer.*

## 6.24.1 Detailed Description

Class of Integers.

A class implementing integers, while additionally satisfying [EuclideanDomain](#) constraint

## 6.24.2 Member Function Documentation

### 6.24.2.1 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:

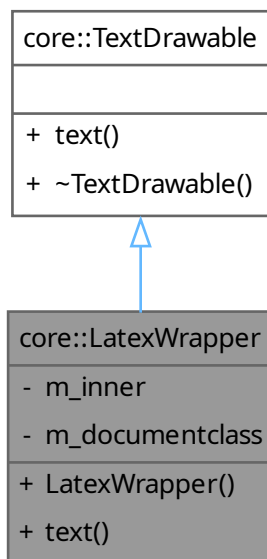
- mc-homology/algebra/include/algebra/[integer.h](#)

## 6.25 core::LatexWrapper Class Reference

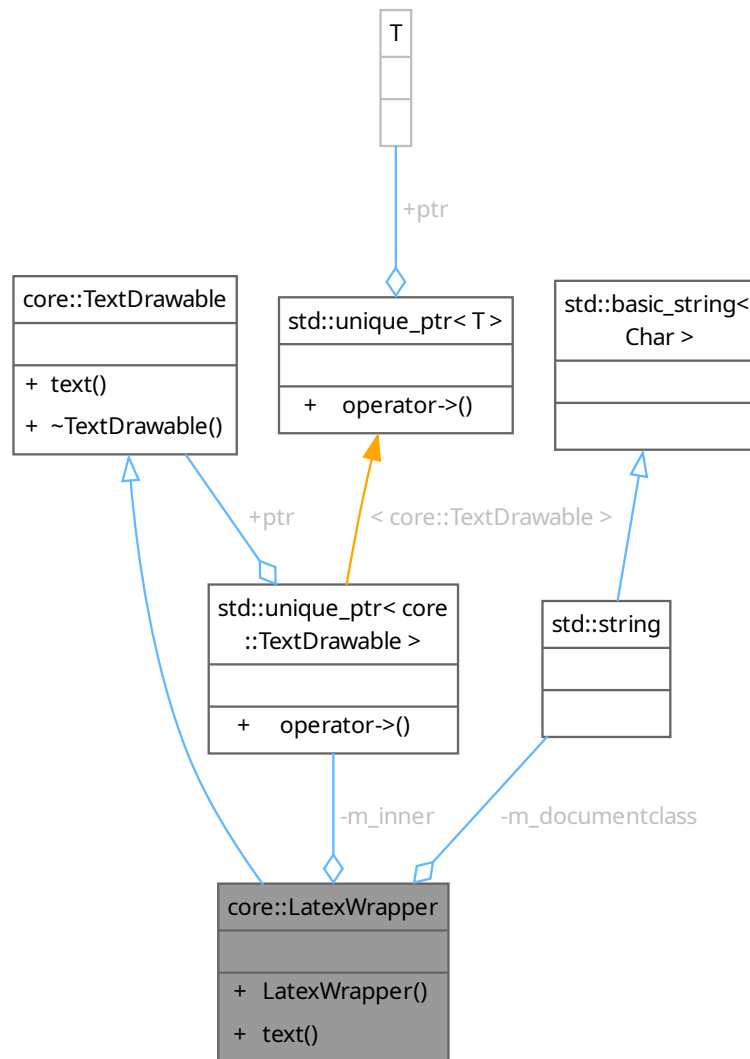
A decorator that wraps a text drawable object in a latex document.

```
#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.*

### Private Attributes

- `std::unique_ptr< TextDrawable > m_inner`  
*Wrapped (decorated) value.*
- `std::string m_documentclass`  
*Document class of the latex file.*

## 6.25.1 Detailed Description

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

## 6.25.2 Member Function Documentation

### 6.25.2.1 `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:

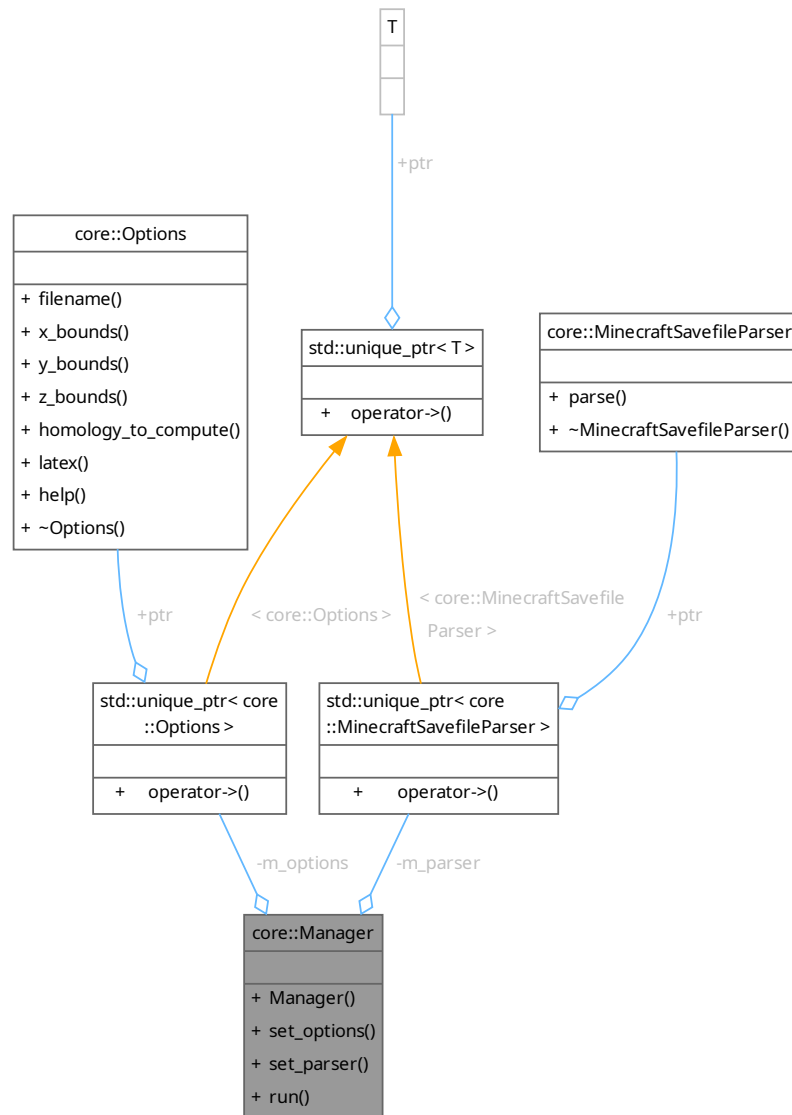
- `mc-homology/core/include/core/latex\_wrapper.h`
- `mc-homology/core/src/latex\_wrapper.cpp`

## 6.26 `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.

### Private Attributes

- `std::unique_ptr< Options > m_options`  
*User options.*
- `std::unique_ptr< MinecraftSavefileParser > m_parser`  
*Parser used to parse the save file.*

## 6.26.1 Detailed Description

Main manager for the program.

## 6.26.2 Member Function Documentation

### 6.26.2.1 `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`

## 6.27 `algebra::Matrix< T >` Class Template Reference

A [Matrix](#) class.

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

Collaboration diagram for algebra::Matrix< T >:

algebra::Matrix< T >
<ul style="list-style-type: none"> <li>- m_data</li> <li>- m_nrows</li> <li>- m_ncols</li> </ul>
<ul style="list-style-type: none"> <li>+ Matrix()</li> <li>+ Matrix()</li> <li>+ begin()</li> <li>+ begin()</li> <li>+ cbegin()</li> <li>+ end()</li> <li>+ end()</li> <li>+ cend()</li> <li>+ rbegin()</li> <li>+ rbegin()</li> <li>and 22 more...</li> <li>+ zero()</li> <li>+ zero()</li> <li>+ id()</li> <li>- to_underlying_index()</li> </ul>

## 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.*

### Private Member Functions

- constexpr [size\\_type to\\_underlying\\_index](#) ([size\\_type](#) row, [size\\_type](#) col) const noexcept
- *Returns the underlying index of an element at given coordinates.*

### Private Attributes

- [storage\\_type m\\_data](#)
- *Stored coefficients.*
- [size\\_type m\\_nrows](#)
- *Number of rows.*
- [size\\_type m\\_ncols](#)
- *Number of columns.*

### 6.27.1 Detailed Description

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

A [Matrix](#) class.

A two-dimensional array representing a mathematical matrix.

### 6.27.2 Constructor & Destructor Documentation

#### 6.27.2.1 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

<a href="#">data</a>	Range with the coefficients
<a href="#">nrows</a>	Number of rows
<a href="#">ncols</a>	Number of columns

### 6.27.3 Member Function Documentation

#### 6.27.3.1 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

<a href="#">row</a>	Accessed row
<a href="#">col</a>	Accessed column

### 6.27.3.2 at() [2/2]

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

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

#### Parameters

<i>row</i>	Accessed row
------------	--------------

<i>col</i>	Accessed column
------------	-----------------

### 6.27.3.3 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.

### 6.27.3.4 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

<i>row</i>	Accessed row
<i>col</i>	Accessed column

### 6.27.3.5 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

<i>row</i>	Accessed row
<i>col</i>	Accessed column

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

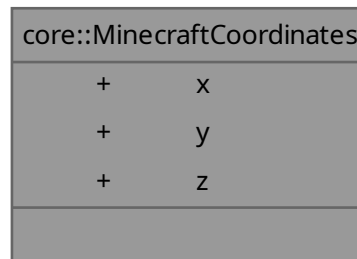
- mc-homology/algebra/include/algebra/[matrix.h](#)

## 6.28 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*

### 6.28.1 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:

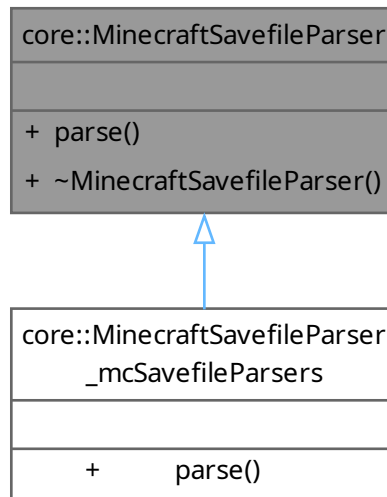
- mc-homology/core/include/core/[parser.h](#)

## 6.29 core::MinecraftSavefileParser Class Reference

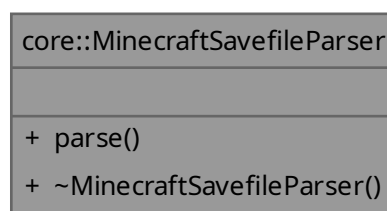
An interface for Minecraft savefile parser.

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

Inheritance diagram for core::MinecraftSavefileParser:



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.*

### 6.29.1 Detailed Description

An interface for Minecraft savefile parser.

### 6.29.2 Member Function Documentation

#### 6.29.2.1 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.  $\text{lower\_corner.x} \leq \text{upper\_corner.x}$   $\text{lower\_corner.y} \leq \text{upper\_corner.y}$   $\text{lower\_corner.z} \leq \text{upper\_corner.z}$

#### Parameters

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

Implemented in [core::MinecraftSavefileParser\\_mcSavefileParsers](#).

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

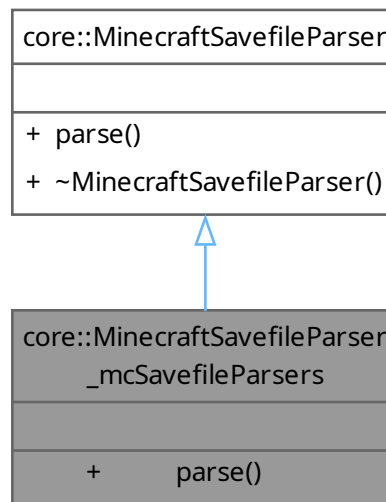
- [mc-homology/core/include/core/parser.h](#)
- [mc-homology/core/src/parser.cpp](#)

## 6.30 core::MinecraftSavefileParser\_mcSavefileParsers Class Reference

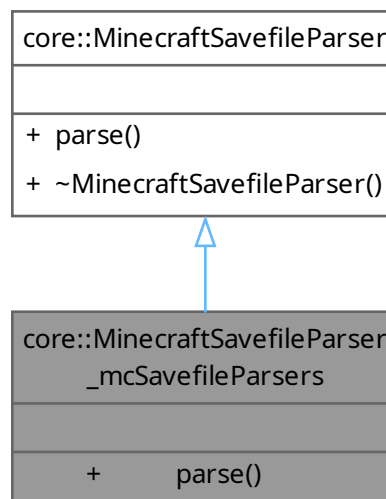
A Minecraft savefile parser based on <https://github.com/TCA166/mcSavefileParsers.git>.

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

Inheritance diagram for core::MinecraftSavefileParser\_mcSavefileParsers:



Collaboration diagram for core::MinecraftSavefileParser\_mcSavefileParsers:



## Public Member Functions

- `std::unique_ptr< Complex > parse` (`std::filesystem::path const &path`, `MinecraftCoordinates lower_corner`, `MinecraftCoordinates upper_corner`) override  
*Parses a Minecraft savefile.*



### 6.30.1 Detailed Description

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

### 6.30.2 Member Function Documentation

#### 6.30.2.1 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

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

Implements [core::MinecraftSavefileParser](#).

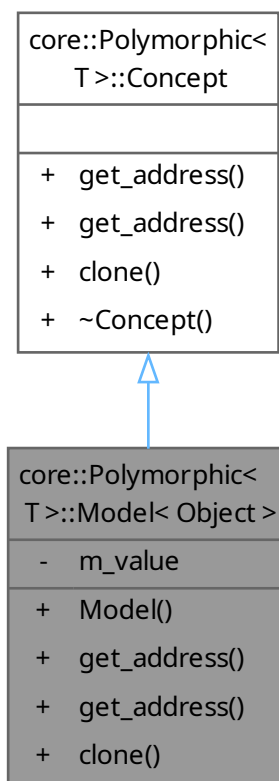
The documentation for this class was generated from the following files:

- mc-homology/core/include/core/[parser.h](#)
- mc-homology/core/src/parser.cpp

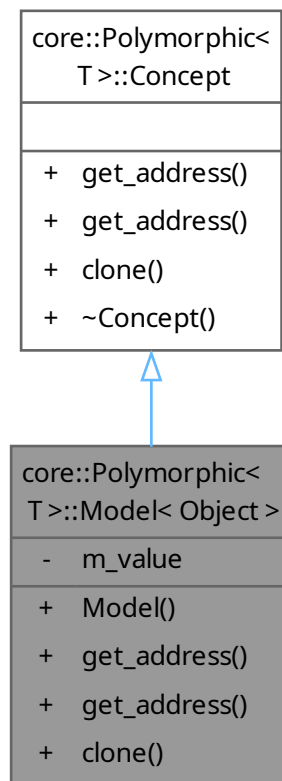
## 6.31 core::Polymorphic< T >::Model< Object > Struct Template Reference

Concrete class used for type erasure.

Inheritance diagram for `core::Polymorphic< T >::Model< Object >`:



Collaboration diagram for core::Polymorphic< T >::Model< Object >:



### Public Member Functions

- `template<class U = Object>`  
`constexpr Model (U &&value)`  
*Constructs a new object.*
- `constexpr T const * get\_address ()` `const` `noexcept` override  
*Returns address of the stored object.*
- `constexpr T * get\_address ()` `noexcept` override  
*Returns address of the stored object.*
- `constexpr std::unique_ptr< Concept > clone ()` `const` override  
*Returns a copy of the stored object as a unique pointer.*

### Public Member Functions inherited from `core::Polymorphic< T >::Concept`

- `virtual constexpr ~Concept ()` `=default`  
*Virtual destructor.*

## Private Attributes

- Object `m_value`  
*The stored object.*

### 6.31.1 Detailed Description

```
template<class T>
template<class Object>
struct core::Polymorphic< T >::Model< Object >
```

Concrete class used for type erasure.

Since [Model](#) knows what type of Object it stores, it can correctly implement a [clone\(\)](#) method

### 6.31.2 Member Function Documentation

#### 6.31.2.1 clone()

```
template<class T>
template<class Object>
std::unique_ptr< Concept > core::Polymorphic< T >::Model< Object >::clone \(\) const [inline],
[constexpr], [override], [virtual]
```

Returns a copy of the stored object as a unique pointer.

Implements [core::Polymorphic< T >::Concept](#).

#### 6.31.2.2 get\_address() [1/2]

```
template<class T>
template<class Object>
T const * core::Polymorphic< T >::Model< Object >::get\_address \(\) const [inline], [constexpr],
[override], [virtual], [noexcept]
```

Returns address of the stored object.

Implements [core::Polymorphic< T >::Concept](#).

#### 6.31.2.3 get\_address() [2/2]

```
template<class T>
template<class Object>
T * core::Polymorphic< T >::Model< Object >::get\_address \(\) [inline], [constexpr], [override],
[virtual], [noexcept]
```

Returns address of the stored object.

Implements [core::Polymorphic< T >::Concept](#).

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

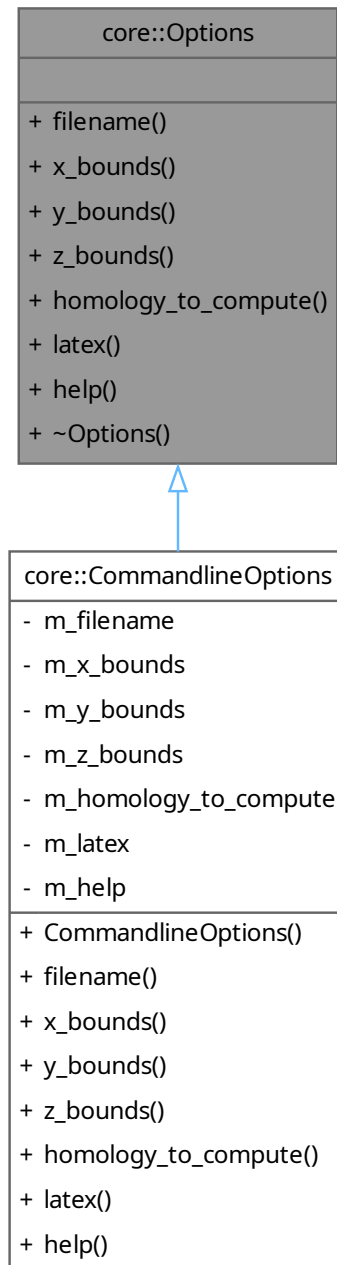
- [mc-homology/core/include/core/polymorphic.h](#)

## 6.32 core::Options Class Reference

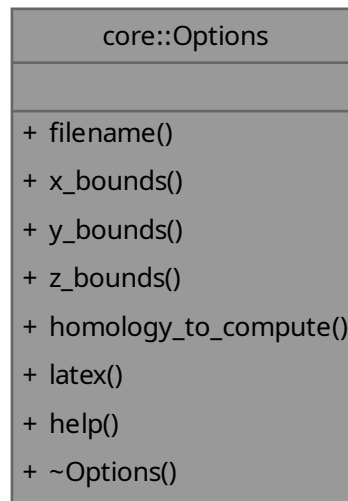
A class for storing user options.

```
#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 z 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*

### 6.32.1 Detailed Description

A class for storing user options.

## 6.32.2 Member Function Documentation

### 6.32.2.1 filename()

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

Path to the region directory.

Implemented in [core::CommandLineOptions](#).

### 6.32.2.2 help()

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

Whether the user requested help.

Implemented in [core::CommandLineOptions](#).

### 6.32.2.3 homology\_to\_compute()

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

Type of homology to compute.

Implemented in [core::CommandLineOptions](#).

### 6.32.2.4 latex()

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

Whether to print latex output.

Implemented in [core::CommandLineOptions](#).

### 6.32.2.5 x\_bounds()

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

Bounds on x axis.

Implemented in [core::CommandLineOptions](#).

### 6.32.2.6 y\_bounds()

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

Bounds on y axis.

Implemented in [core::CommandLineOptions](#).

### 6.32.2.7 `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](#)

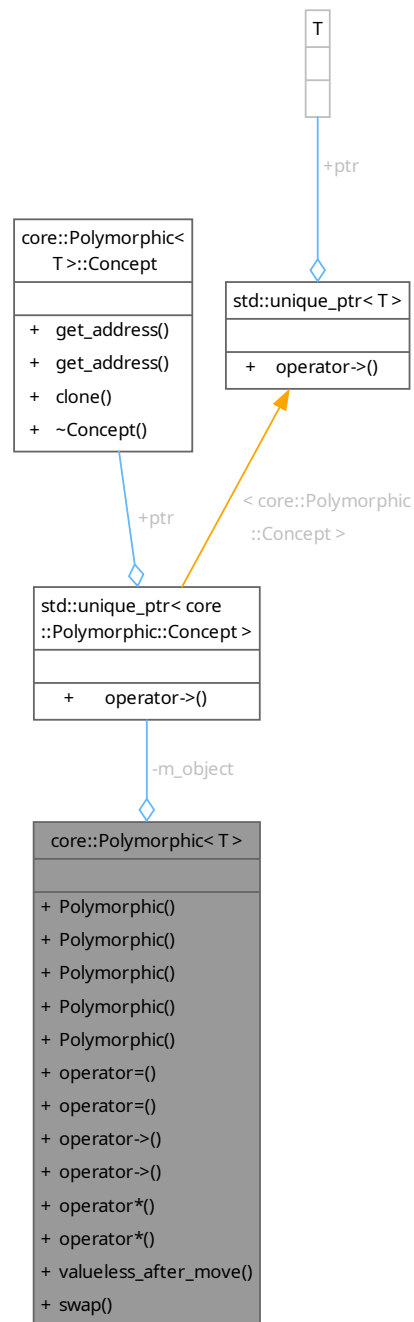
## 6.33 `core::Polymorphic< T >` Class Template Reference

A polymorphic class wrapper with value semantics.

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



Collaboration diagram for core::Polymorphic< T >:



## Classes

- struct [Concept](#)  
*Base class used for type erasure.*
- struct [Model](#)  
*Concrete class used for type erasure.*

## 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.*

## Private Attributes

- std::unique\_ptr< [Concept](#) > **m\_object**  
*Pointer to the [Concept](#) class, which in turn holds a [Model](#) with our object.*

### 6.33.1 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](#)

## 6.34 algebra::RowEchelonFormResult< T > Struct Template Reference

Result struct for the row echelon algorithm.

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

Collaboration diagram for algebra::RowEchelonFormResult< T >:

algebra::RowEchelonFormResult< T >	
+	row_echelon_form
+	non_empty_rows

### 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.*

### 6.34.1 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:

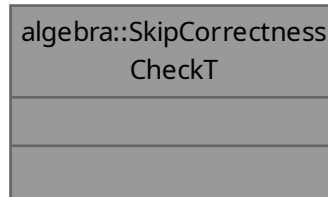
- mc-homology/algebra/include/algebra/[matrix\\_algorithms.h](#)

## 6.35 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:



### 6.35.1 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:

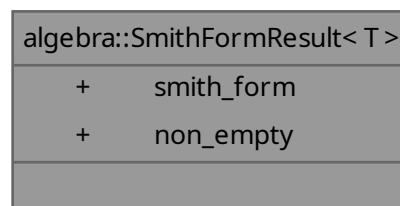
- mc-homology/algebra/include/algebra/[chain\\_complex.h](#)

## 6.36 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.*

#### 6.36.1 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:

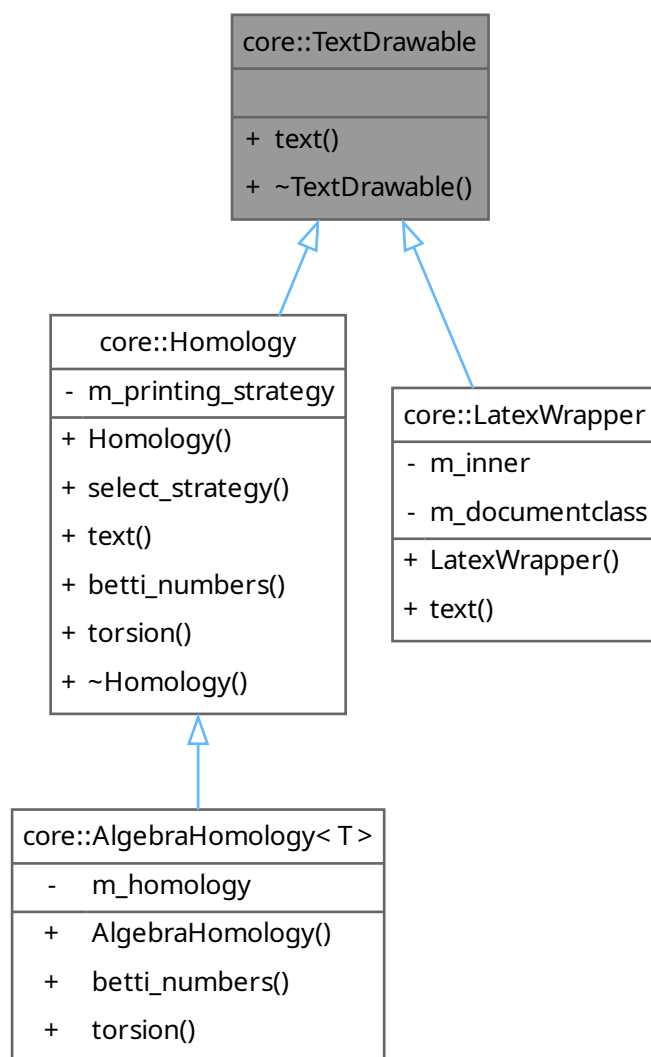
- mc-homology/algebra/include/algebra/[matrix\\_algorithms.h](#)

## 6.37 core::TextDrawable Class Reference

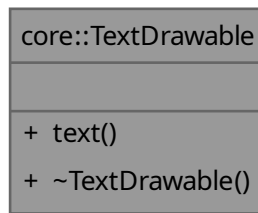
Marks a class text drawable - it can be represented as a string.

```
#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.*

### 6.37.1 Detailed Description

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

### 6.37.2 Member Function Documentation

#### 6.37.2.1 `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:

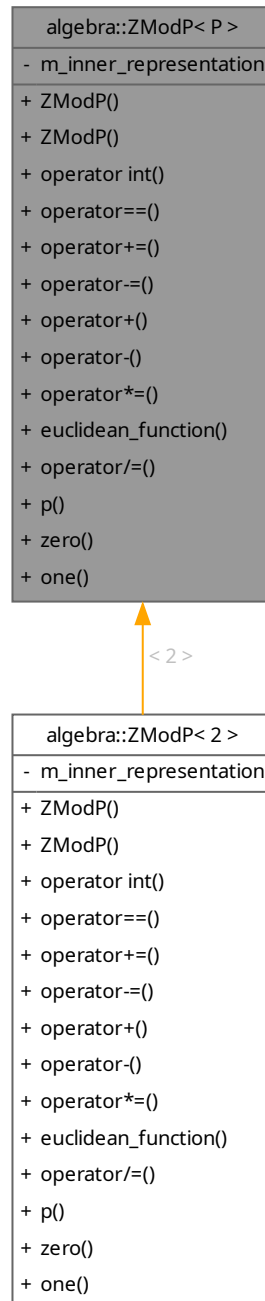
- mc-homology/core/include/core/[text\\_drawable.h](#)
- mc-homology/core/src/[text\\_drawable.cpp](#)

## 6.38 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 >:

algebra::ZModP< P >
- m_inner_representation
+ ZModP()
+ ZModP()
+ operator int()
+ operator==()
+ operator+=()
+ operator-=()
+ operator+()
+ operator-()
+ operator*=( )
+ euclidean_function()
+ operator/=( )
+ p()
+ zero()
+ one()

## Public Member Functions

- constexpr **ZModP** ()=default  
*Returns 0.*
- constexpr **ZModP** (int n) noexcept  
*Returns  $n \bmod P$ .*
- 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.*

### Private Attributes

- int **m\_inner\_representation** = 0

*Inner representation of the integer.*

## 6.38.1 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

<i>P</i>	A prime number
----------	----------------

## 6.38.2 Member Function Documentation

### 6.38.2.1 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:

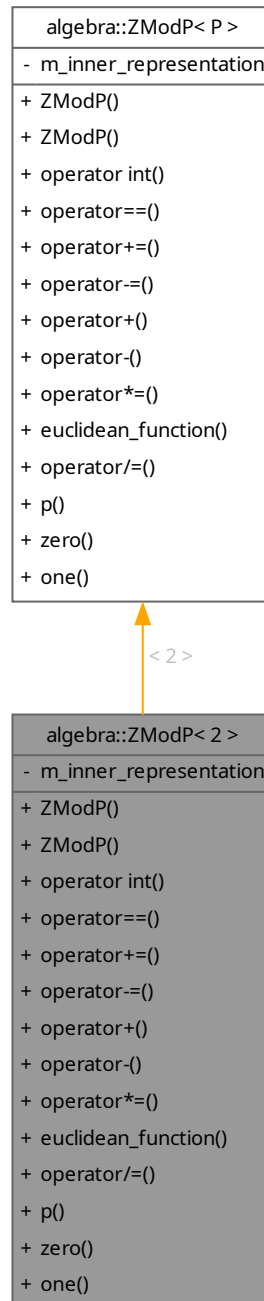
- mc-homology/algebra/include/algebra/[modulo\\_fields.h](#)

## 6.39 algebra::ZModP< 2 > Class Reference

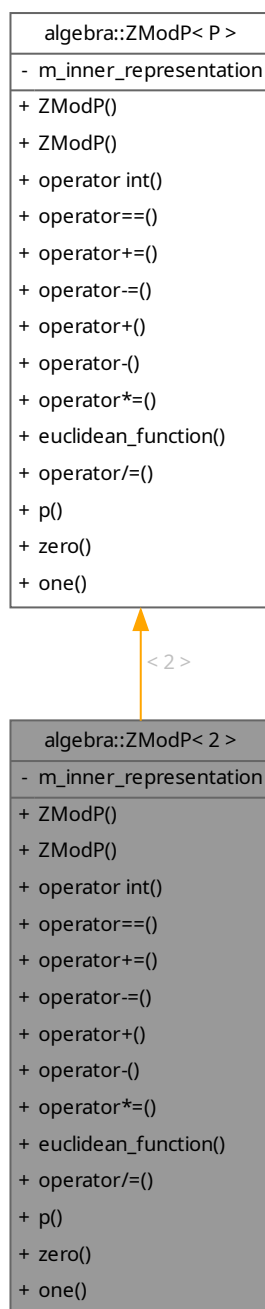
Template specialization for  $P == 2$ .

```
#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  
*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 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.*

### Private Attributes

- bool **m\_inner\_representation** = false  
*Represents 0 or 1 as a binary value.*

## 6.39.1 Detailed Description

Template specialization for  $P = 2$ .

## 6.39.2 Member Function Documentation

### 6.39.2.1 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

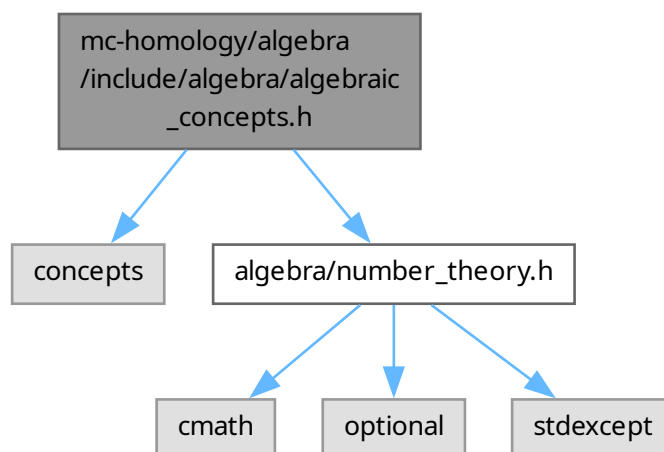
# File Documentation

### 7.1 mc-homology/algebra/include/algebra/algebraic\_concepts.h File Reference

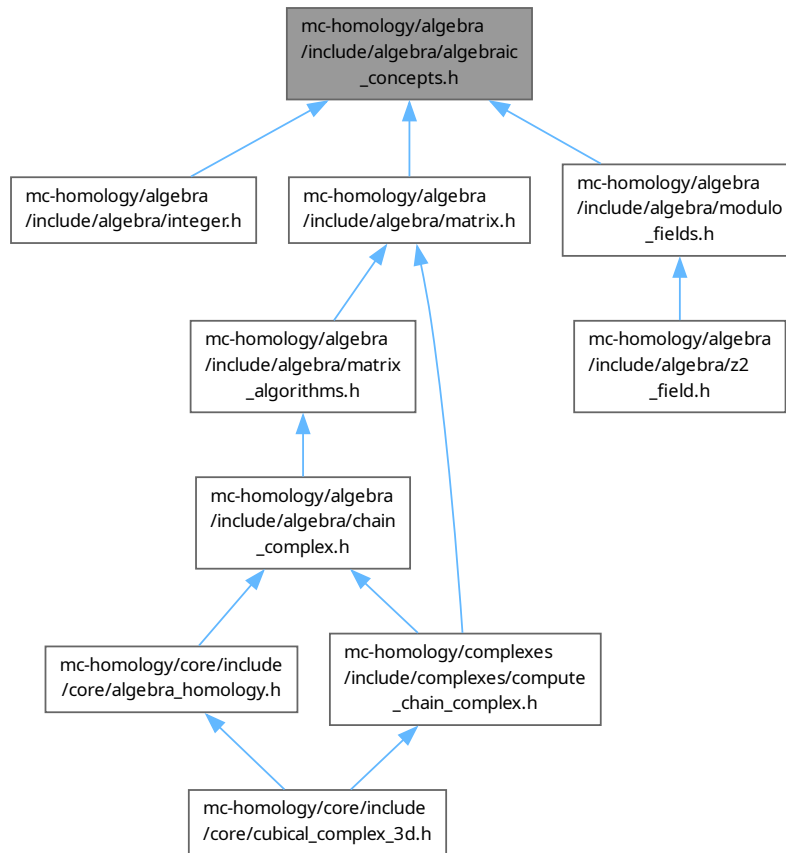
A file defining multiple concepts for algebraic structures.

```
#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:



## 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.](#)

```

00001
00003
00004 #pragma once
00005
00006 #include <concepts>
00007
00008 #include "algebra/number_theory.h"
00009
00010 namespace algebra {
00011
00014 template<class T>
00015 constexpr inline bool is_commutative_v = false;
00016
00027 template<class T>
00028 concept Commutative = is_commutative_v<T>;
00029
00042 template<class T>
00043 concept Group =
00044     std::regular<T> && std::default_initializable<T> && requires(T x, T y) {
00045         { x * y } -> std::convertible_to<T>;
00046         { x / y } -> std::convertible_to<T>;
00047         { x * y } -> std::same_as<T&>;
00048         { x / y } -> std::same_as<T&>;
00049         { T::one() } -> std::convertible_to<T>;
00050     };
00051
00065 template<class T>
00066 concept AdditiveGroup = std::regular<T> && std::default_initializable<T>
00067     && Commutative<T> && requires(T x, T y) {
00068         { x + y } -> std::convertible_to<T>;
00069         { x - y } -> std::convertible_to<T>;
00070         { x += y } -> std::same_as<T&>;
00071         { x -= y } -> std::same_as<T&>;
00072         { +x } -> std::convertible_to<T>;
00073         { -x } -> std::convertible_to<T>;
00074         { T::zero() } -> std::convertible_to<T>;
00075     };
00076
00091 template<class T>
00092 concept Ring = AdditiveGroup<T> && requires(T x, T y) {
00093     { x * y } -> std::convertible_to<T>;
00094     { x * y } -> std::same_as<T&>;
00095     { T::one() } -> std::convertible_to<T>;
00096 };
00097
00109 template<class T>
00110 concept CommutativeRing = Ring<T> && Commutative<T>;
00111
00131 template<class T>
00132 concept EuclideanDomain = CommutativeRing<T> && requires(T a, T b, T x, T y) {
00133     { divide(x, y) } -> std::same_as<DivResult<T>>;
00134     { x.euclidean_function() } -> std::same_as<int>;
00135 };
00136
00149 template<class T>
00150 concept Field = CommutativeRing<T> && requires(T x, T y) {
00151     { x / y } -> std::convertible_to<T>;
00152     { x / y } -> std::same_as<T&>;
00153 };
00154
00155 } // namespace algebra

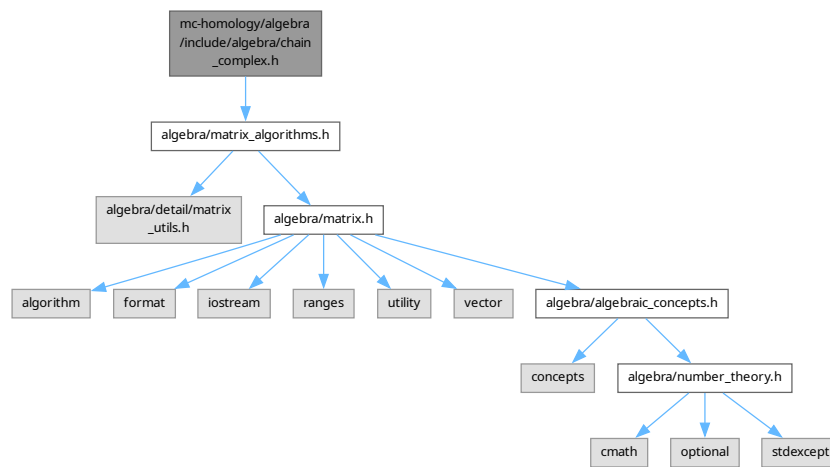
```

## 7.3 mc-homology/algebra/include/algebra/chain\_complex.h File Reference

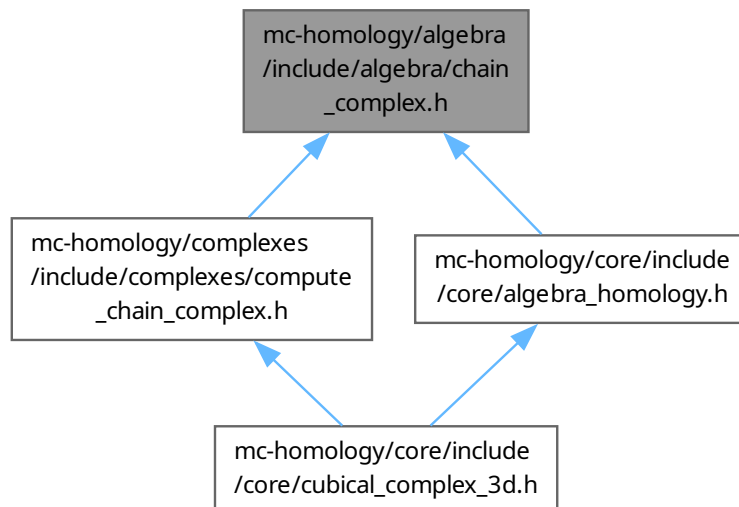
A file containing chain complex and homology implementations.

```
#include "algebra/matrix_algorithms.h"
```

Include dependency graph for chain\_complex.h:



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



## 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.](#)

```

00001
00002
00003
00004 #pragma once
00005
00006 #include "algebra/matrix_algorithms.h"
00007
00008 namespace algebra {
00009
00016 struct SkipCorrectnessCheckT {};
00017
00019 constexpr inline SkipCorrectnessCheckT skip_correctness_check {};
00020
00029 template<class T>
00030 class ChainComplex {
00031 public:
00032     constexpr ChainComplex() = default;
00033
00038     template<std::ranges::sized_range R>
00039         requires std::convertible_to<std::ranges::range_value_t<R>, Matrix<T>>
00040     constexpr ChainComplex(R&& boundaries) :
00041         ChainComplex(skip_correctness_check, std::forward<R>(boundaries)) {
00042         if (!check_boundary_correctness()) {
00043             throw std::domain_error(
00044                 "The boundary matrices do not satisfy chain complex condition"
00045             );
00046         }
00047     }
00048
00055     template<std::ranges::sized_range R>
00056         requires std::convertible_to<std::ranges::range_value_t<R>, Matrix<T>>
00057     constexpr ChainComplex(SkipCorrectnessCheckT, R&& boundaries) :
00058         m_boundaries(
00059             std::ranges::to<std::vector<Matrix<T>>>(std::forward<R>(boundaries))

```

```

00060     ) {}
00061
00062     constexpr bool check_boundary_correctness() const {
00063         if (m_boundaries.size() < 2) {
00064             return true;
00065         }
00066         namespace rs = std::ranges;
00067         namespace vs = std::views;
00068         auto b_n_plus_1_view = m_boundaries | vs::drop(1);
00069         auto b_n_view = m_boundaries | vs::take(m_boundaries.size() - 1);
00070         return rs::all_of(
00071             vs::zip(b_n_plus_1_view, b_n_view),
00072             [](auto const& boundaries) {
00073                 try {
00074                     auto const& [b_n_plus_1, b_n] = boundaries;
00075                     return (b_n * b_n_plus_1).is_zero();
00076                 } catch (std::domain_error&) {
00077                     return false;
00078                 }
00079             }
00080         );
00081     };
00082
00083     constexpr std::size_t dimension() const noexcept {
00084         return m_boundaries.size();
00085     }
00086
00087     constexpr Matrix<T> const& boundary(std::size_t dim) const {
00088         return m_boundaries.at(dim);
00089     }
00090
00091     constexpr std::vector<Matrix<T>> const& boundaries() const noexcept {
00092         return m_boundaries;
00093     }
00094
00095 private:
00096     std::vector<Matrix<T>> m_boundaries;
00097 };
00098
00099 template<class T>
00100 struct Homology {
00101     std::vector<std::size_t> betti_numbers {};
00102
00103     std::vector<std::vector<T>> torsion {};
00104 };
00105
00106 template<EuclideanDomain T>
00107 Homology<T> homology(ChainComplex<T> const& chain_complex) {
00108     namespace rs = std::ranges;
00109     namespace vs = std::views;
00110     auto const& boundaries = chain_complex.boundaries();
00111     Homology<T> homology;
00112     homology.betti_numbers.resize(boundaries.size());
00113     homology.torsion.resize(boundaries.size());
00114     std::size_t prev_smith_units_count = 0;
00115     std::vector<T> prev_smith_diagonal_without_units {};
00116     for (std::size_t k = boundaries.size(); k > 0; --k) {
00117         auto const n = k - 1;
00118         auto const& boundary = boundaries[n];
00119         auto [smith, rank] = smith_form(boundary);
00120         auto nullity = boundary.ncols() - rank;
00121         auto smith_diagonal_without_units = rs::to<std::vector>(
00122             vs::iota(0u, rank)
00123             | vs::transform([&smith](std::size_t n) { return smith[n, n]; })
00124             | vs::drop_while([](T const& x) {
00125                 return x.euclidean_function() == 1;
00126             })
00127         );
00128         auto smith_units = rank - smith_diagonal_without_units.size();
00129         homology.betti_numbers[n] = nullity - prev_smith_units_count
00130             - prev_smith_diagonal_without_units.size();
00131         homology.torsion[n] = std::move(prev_smith_diagonal_without_units);
00132         prev_smith_units_count = smith_units;
00133         prev_smith_diagonal_without_units =
00134             std::move(smith_diagonal_without_units);
00135     }
00136     return homology;
00137 }
00138
00139 template<Field T>
00140 Homology<T> homology(ChainComplex<T> const& chain_complex) {
00141     auto const& boundaries = chain_complex.boundaries();
00142     Homology<T> homology;
00143     homology.betti_numbers.resize(boundaries.size());
00144     homology.torsion.resize(boundaries.size());
00145     std::size_t prev_rank = 0;
00146     for (std::size_t k = boundaries.size(); k > 0; --k) {

```

```

00172     auto const n = k - 1;
00173     auto const& boundary = boundaries[n];
00174     auto [_, rank] = row_echelon_form(boundary);
00175     auto nullity = boundary.ncols() - rank;
00176     homology.betti_numbers[n] = nullity - prev_rank;
00177     prev_rank = rank;
00178 }
00179 return homology;
00180 }
00181
00182 template<std::ranges::sized_range R>
00183 ChainComplex(SkipCorrectnessCheckT, R&&)
00184     -> ChainComplex<typename std::ranges::range_value_t<R>::value_type>;
00185
00186 template<std::ranges::sized_range R>
00187 ChainComplex(R&&)
00188     -> ChainComplex<typename std::ranges::range_value_t<R>::value_type>;
00189
00190 } // namespace algebra
00191
00192 }

```

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

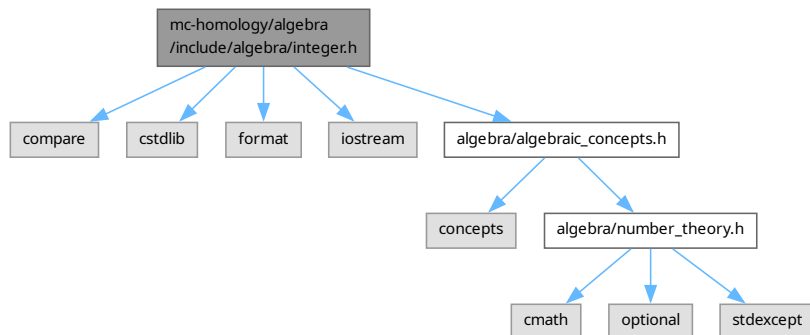
A file containing Integer implementation satisfying algebraic concepts.

```

#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.  $0 \leq r < b.euclidean\_function()$

### 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,  $0 \leq r < n$  satisfying  $a == q * n + r$

## 7.6 integer.h

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

```

00001
00004
00005 #pragma once
00006
00007 #include <compare>
00008 #include <cstdlib>
00009 #include <format>
00010 #include <iostream>
00011
00012 #include "algebra/algebraic_concepts.h"
00013
00014 namespace algebra {
00015
00020 class Integer {
00021 public:
00023     constexpr Integer() = default;
00024
00026     constexpr Integer(int k) noexcept : m_inner_representation(k) {}
00027
00029     constexpr static Integer zero() noexcept {
00030         return Integer(0);
00031     }
00032
00034     constexpr static Integer one() noexcept {
00035         return Integer(1);
00036     }
00037
00039     constexpr explicit operator int() noexcept {
00040         return m_inner_representation;
00041     }
00042
00044     constexpr bool operator==(Integer const&) const = default;
00045
00047     constexpr std::strong_ordering operator<=>(Integer const&) const = default;
00048
00050     constexpr Integer& operator+=(Integer rhs) noexcept {
00051         m_inner_representation += rhs.m_inner_representation;
00052         return *this;
00053     }
00054
00056     constexpr Integer& operator-=(Integer rhs) noexcept {
00057         m_inner_representation -= rhs.m_inner_representation;
00058         return *this;
00059     }
00060
00062     constexpr Integer operator+() const noexcept {
00063         return *this;
00064     }
00065
00067     constexpr Integer operator-() const noexcept {
00068         return Integer(-m_inner_representation);
00069     }
00070
00072     constexpr Integer& operator*=(Integer rhs) noexcept {
00073         m_inner_representation *= rhs.m_inner_representation;
00074         return *this;
00075     }
00076
00081     constexpr int euclidean_function() const noexcept {
00082         using std::abs;
00083         return abs(m_inner_representation);
00084     }
00085
00086 private:
00088     int m_inner_representation = 0;
00089 };
00090
00092 constexpr Integer operator+(Integer lhs, Integer rhs) noexcept {
00093     return lhs += rhs;
00094 }
00095
00097 constexpr Integer operator-(Integer lhs, Integer rhs) noexcept {
00098     return lhs -= rhs;
00099 }
00100
00102 constexpr Integer operator*(Integer lhs, Integer rhs) noexcept {
00103     return lhs *= rhs;
00104 }
00105
00107 constexpr Integer abs(Integer k) noexcept {
00108     return std::abs(static_cast<int>(k));
00109 }

```

```

00110
00116 constexpr DivResult<Integer> divide(Integer a, Integer b) {
00117     auto div_result = divide(static_cast<int>(a), static_cast<int>(b));
00118     return DivResult<Integer> {
00119         .quotient = div_result.quotient,
00120         .remainder = div_result.remainder
00121     };
00122 }
00123
00128 constexpr Integer modulo(Integer a, Integer n) {
00129     return divide(a, n).remainder;
00130 }
00131
00133 template<>
00134 constexpr inline bool is_commutative_v<Integer> = true;
00135
00137 inline std::ostream& operator<<(std::ostream& output, Integer k) {
00138     return output << static_cast<int>(k);
00139 }
00140
00141 } // namespace algebra
00142
00147 template<>
00148 struct std::formatter<algebra::Integer>: public std::formatter<int> {
00152     template<class FmtContext>
00153     FmtContext::iterator format(algebra::Integer k, FmtContext& ctx) const {
00154         return std::formatter<int>::format(static_cast<int>(k), ctx);
00155     }
00156 };

```

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

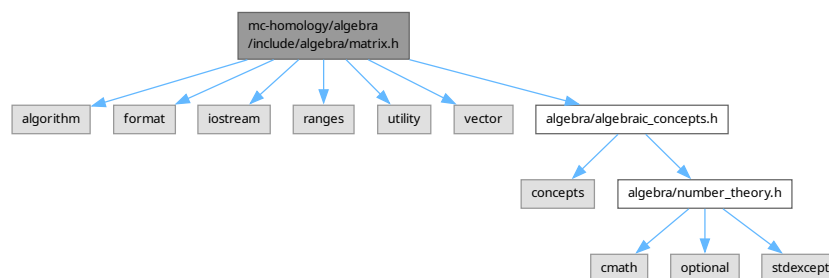
A file containing matrix implementation.

```

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

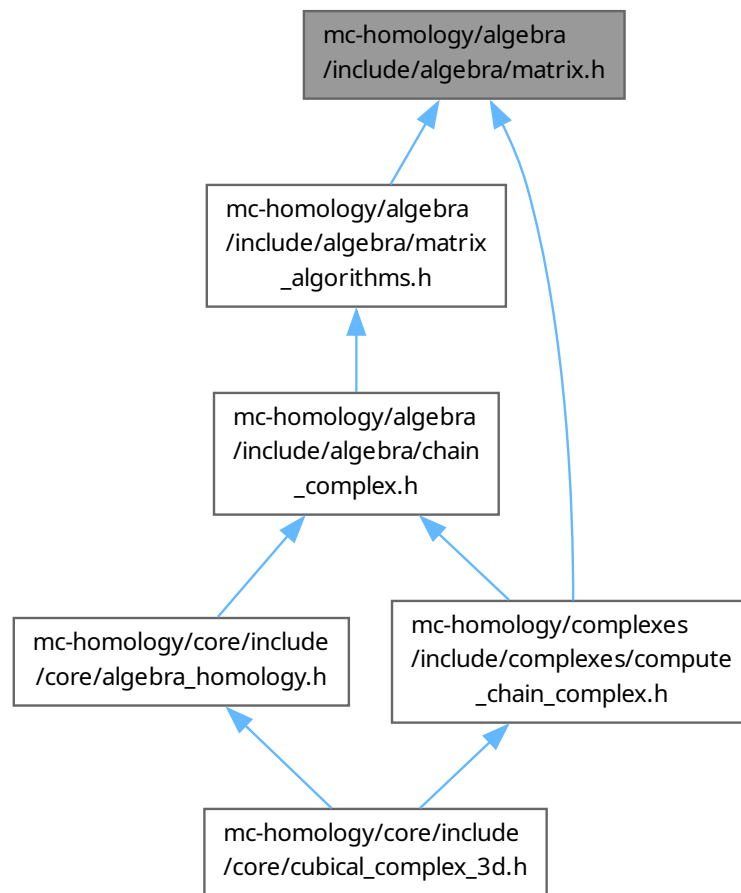
```

Include dependency graph for matrix.h:





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



## 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.](#)

```
00001
00002
00003
00004 #pragma once
00005
00006 #include <algorithm>
00007 #include <format>
00008 #include <iostream>
00009 #include <ranges>
00010 #include <utility>
00011 #include <vector>
00012
00013 #include "algebra/algebraic_concepts.h"
00014
00015 namespace algebra {
00016
00020 template<class T>
00021 class Matrix {
00022 public:
00024     using storage_type = std::vector<T>;
00026     using value_type = std::vector<T>::value_type;
00028     using allocator_type = std::vector<T>::allocator_type;
00030     using size_type = std::vector<T>::size_type;
00032     using difference_type = std::vector<T>::difference_type;
00034     using reference = std::vector<T>::reference;
00036     using const_reference = std::vector<T>::const_reference;
00038     using pointer = std::vector<T>::pointer;
00040     using const_pointer = std::vector<T>::const_pointer;
00042     using iterator = std::vector<T>::iterator;
00044     using const_iterator = std::vector<T>::const_iterator;
00046     using reverse_iterator = std::vector<T>::reverse_iterator;
00048     using const_reverse_iterator = std::vector<T>::const_reverse_iterator;
00049
00050     constexpr Matrix() = default;
00051
```

```

00061     template<std::ranges::sized_range R>
00062         requires std::convertible_to<std::ranges::range_value_t<R>, T>
00063     constexpr explicit Matrix(R&& data, size_type nrows, size_type ncols) :
00064         m_data {},
00065         m_nrows {nrows},
00066         m_ncols {ncols} {
00067         if (std::ranges::size(data) != nrows * ncols) [[unlikely]] {
00068             throw std::domain_error(
00069                 "Size of the array is not equal to the number"
00070                 " of rows times the number of columns"
00071             );
00072         }
00073         m_data = std::ranges::to<std::vector<T>(std::forward<R>(data));
00074     }
00075
00077     constexpr iterator begin() noexcept {
00078         return m_data.begin();
00079     }
00080
00082     constexpr const_iterator begin() const noexcept {
00083         return m_data.begin();
00084     }
00085
00087     constexpr const_iterator cbegin() const noexcept {
00088         return m_data.cbegin();
00089     }
00090
00092     constexpr iterator end() noexcept {
00093         return m_data.end();
00094     }
00095
00097     constexpr const_iterator end() const noexcept {
00098         return m_data.end();
00099     }
00100
00102     constexpr const_iterator cend() const noexcept {
00103         return m_data.cend();
00104     }
00105
00107     constexpr reverse_iterator rbegin() noexcept {
00108         return m_data.rbegin();
00109     }
00110
00112     constexpr const_reverse_iterator rbegin() const noexcept {
00113         return m_data.rbegin();
00114     }
00115
00117     constexpr const_reverse_iterator crbegin() const noexcept {
00118         return m_data.crbegin();
00119     }
00120
00122     constexpr reverse_iterator rend() noexcept {
00123         return m_data.rend();
00124     }
00125
00127     constexpr const_reverse_iterator rend() const noexcept {
00128         return m_data.rend();
00129     }
00130
00132     constexpr const_reverse_iterator crend() const noexcept {
00133         return m_data.crend();
00134     }
00135
00137     constexpr bool empty() const noexcept {
00138         return m_data.empty();
00139     }
00140
00142     constexpr size_type size() const noexcept {
00143         return m_data.size();
00144     }
00145
00147     constexpr void swap(Matrix& other) noexcept(
00148         m_data.swap(std::declval<std::vector<value_type>>()
00149     ) {
00150         namespace rs = std::ranges;
00151         m_data.swap(other.m_data);
00152         rs::swap(m_nrows, other.m_nrows);
00153         rs::swap(m_ncols, other.m_ncols);
00154     }
00155
00157     constexpr size_type nrows() const noexcept {
00158         return m_nrows;
00159     }
00160
00162     constexpr size_type ncols() const noexcept {
00163         return m_ncols;
00164     }

```

```

00165
00167     constexpr bool operator==(Matrix const& const) = default;
00168
00173     constexpr reference operator[](size_type row, size_type col) {
00174         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00175             throw std::out_of_range("Indices out of matrix range");
00176         }
00177         return m_data[to_underlying_index(row, col)];
00178     }
00179
00184     constexpr const_reference operator[](size_type row, size_type col) const {
00185         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00186             throw std::out_of_range("Indices out of matrix range");
00187         }
00188         return m_data[to_underlying_index(row, col)];
00189     }
00190
00195     constexpr reference at(size_type row, size_type col) {
00196         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00197             throw std::out_of_range("Indices out of matrix range");
00198         }
00199         return m_data.at(to_underlying_index(row, col));
00200     }
00201
00206     constexpr const_reference at(size_type row, size_type col) const {
00207         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00208             throw std::out_of_range("Indices out of matrix range");
00209         }
00210         return m_data.at(to_underlying_index(row, col));
00211     }
00212
00214     constexpr storage_type const& data() const noexcept {
00215         return m_data;
00216     }
00217
00219     constexpr Matrix transpose() const {
00220         auto transposed = zero(m_ncols, m_nrows);
00221         for (size_type i = 0; i < m_nrows; ++i) {
00222             for (size_type j = 0; j < m_ncols; ++j) {
00223                 transposed[j, i] = at(i, j);
00224             }
00225         }
00226         return transposed;
00227     }
00228
00230     constexpr Matrix& operator+=(Matrix const& rhs)
00231         requires AdditiveGroup<T>
00232     {
00233         if (m_nrows != rhs.m_nrows || m_ncols != rhs.m_ncols) {
00234             throw std::domain_error("Adding matrices of different dimensions");
00235         }
00236         namespace rs = std::ranges;
00237         rs::transform(m_data, rhs.m_data, rs::begin(m_data), std::plus {});
00238         return *this;
00239     }
00240
00242     constexpr Matrix& operator-=(Matrix const& rhs)
00243         requires AdditiveGroup<T>
00244     {
00245         if (m_nrows != rhs.m_nrows || m_ncols != rhs.m_ncols) {
00246             throw std::domain_error(
00247                 "Subtracting matrices of different dimensions"
00248             );
00249         }
00250         namespace rs = std::ranges;
00251         rs::transform(m_data, rhs.m_data, rs::begin(m_data), std::minus {});
00252     }
00253
00255     constexpr Matrix operator+() const
00256         requires AdditiveGroup<T>
00257     {
00258         return *this;
00259     }
00260
00262     constexpr Matrix operator-() const
00263         requires AdditiveGroup<T>
00264     {
00265         namespace rs = std::ranges;
00266         std::vector<T> negated;
00267         negated.reserve(m_data.capacity());
00268         rs::transform(m_data, rs::begin(negated), std::negate {});
00269         return Matrix(negated, m_nrows, m_ncols);
00270     }
00271
00275     constexpr Matrix operator*(Matrix const& rhs)
00276         requires Ring<T>
00277     {

```

```

00278     *this = *this * rhs;
00279 }
00280
00282 constexpr static Matrix zero(size_type n)
00283     requires AdditiveGroup<T>
00284 {
00285     std::vector<data>(n * n, value_type::zero());
00286     return Matrix(std::move(data), n, n);
00287 }
00288
00290 constexpr static Matrix zero(size_type n, size_type m)
00291     requires AdditiveGroup<T>
00292 {
00293     std::vector<data>(n * m, value_type::zero());
00294     return Matrix(std::move(data), n, m);
00295 }
00296
00298 constexpr bool is_zero() const noexcept {
00299     return std::ranges::all_of(m_data, [](T const& x) {
00300         return x == T::zero();
00301     });
00302 }
00303
00305 constexpr static Matrix id(size_type n)
00306     requires CommutativeRing<T>
00307 {
00308     auto identity = zero(n);
00309     for (size_type i = 0; i < n; ++i) {
00310         identity[i, i] = value_type::one();
00311     }
00312     return identity;
00313 }
00314
00315 private:
00318     constexpr size_type
00319     to_underlying_index(size_type row, size_type col) const noexcept {
00320         return row * m_ncols + col;
00321     }
00322
00324     storage_type m_data;
00326     size_type m_nrows;
00328     size_type m_ncols;
00329 };
00330
00332 template<AdditiveGroup T>
00333 constexpr Matrix<T> operator+(Matrix<T> lhs, Matrix<T> const& rhs) {
00334     return lhs += rhs;
00335 }
00336
00338 template<AdditiveGroup T>
00339 constexpr Matrix<T> operator-(Matrix<T> lhs, Matrix<T> const& rhs) {
00340     return lhs -= rhs;
00341 }
00342
00346 template<Ring T>
00347 constexpr Matrix<T> operator*(Matrix<T> const& lhs, Matrix<T> const& rhs) {
00348     using Matrix = Matrix<T>;
00349     using size_type = Matrix::size_type;
00350     if (lhs.ncols() != rhs.nrows()) {
00351         throw std::domain_error(
00352             "The number of columns of lhs is different "
00353             "than the number of rows of rhs"
00354         );
00355     }
00356     auto const inner_dim = lhs.ncols();
00357     Matrix product = Matrix::zero(lhs.nrows(), rhs.ncols());
00358     for (size_type i = 0; i < product.nrows(); ++i) {
00359         for (size_type j = 0; j < product.ncols(); ++j) {
00360             for (size_type k = 0; k < inner_dim; ++k) {
00361                 product[i, j] += lhs[i, k] * rhs[k, j];
00362             }
00363         }
00364     }
00365     return product;
00366 }
00367
00369 template<std::ranges::sized_range R>
00370 Matrix(R&&, std::size_t, std::size_t) -> Matrix<std::ranges::range_value_t<R>>;
00371
00373 template<class T>
00374 std::ostream& operator<<(std::ostream& output, Matrix<T> const& matrix) {
00375     return output << std::format("{} ", matrix);
00376 }
00377
00378 } // namespace algebra
00379
00403 template<class T>

```

```

00404     requires std::formattable<T, char>
00405 struct std::formatter<algebra::Matrix<T> {
00407     template<class ParseContext>
00408     constexpr ParseContext::iterator parse(ParseContext& ctx) {
00409         auto it = ctx.begin();
00410         if (it == ctx.end() || *it == '}') {
00411             return it;
00412         } else if (*it == '-') {
00413             multi_line = false;
00414             ++it;
00415         } else if (*it == '#') {
00416             multi_line = true;
00417             ++it;
00418         }
00419
00420         if (it == ctx.end() || *it == '}') {
00421             return it;
00422         } else if (*it == ':') {
00423             ctx.advance_to(++it);
00424             return coefficient_formatter.parse(ctx);
00425         } else {
00426             throw std::format_error("Invalid formatting option for Matrix");
00427         }
00428     }
00429
00434     template<class FmtContext>
00435     FmtContext::iterator
00436     format(algebra::Matrix<T> const& matrix, FmtContext& ctx) const {
00437         auto const maybe_new_line = multi_line ? "\n" : "";
00438         if (matrix.empty()) {
00439             std::format_to(ctx.out(), "[ ]");
00440         } else {
00441             std::format_to(ctx.out(), "[");
00442             for (std::size_t i = 0; i < matrix.nrows(); ++i) {
00443                 std::format_to(ctx.out(), "[");
00444                 for (std::size_t j = 0; j < matrix.ncols(); ++j) {
00445                     coefficient_formatter.format(matrix[i, j], ctx);
00446                     if (j + 1 < matrix.ncols()) {
00447                         std::format_to(ctx.out(), ", ");
00448                     }
00449                 }
00450                 std::format_to(ctx.out(), "]");
00451                 if (i + 1 < matrix.nrows()) {
00452                     std::format_to(ctx.out(), ",{} ", maybe_new_line);
00453                 }
00454             }
00455             std::format_to(ctx.out(), "]");
00456         }
00457         if (multi_line) {
00458             std::format_to(
00459                 ctx.out(),
00460                 "\nMatrix {} x {}\n",
00461                 matrix.nrows(),
00462                 matrix.ncols()
00463             );
00464         }
00465         return ctx.out();
00466     }
00467
00468 private:
00470     bool multi_line = false;
00472     std::formatter<T> coefficient_formatter;
00473 };

```

## 7.9 mc-homology/algebra/include/algebra/matrix\_algorithms.h File Reference

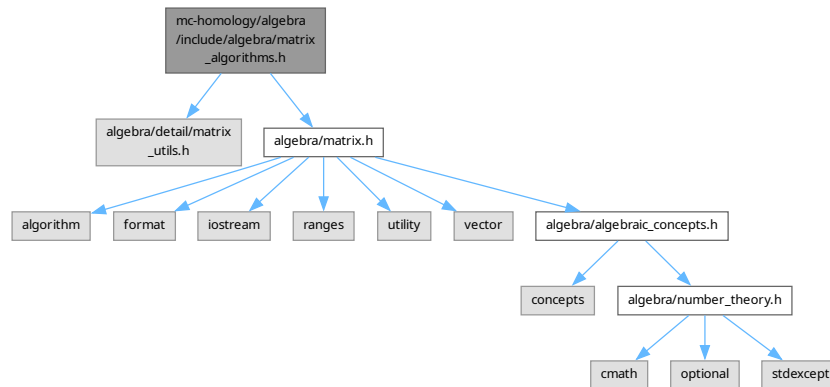
A file containing selected matrix algorithms.

```

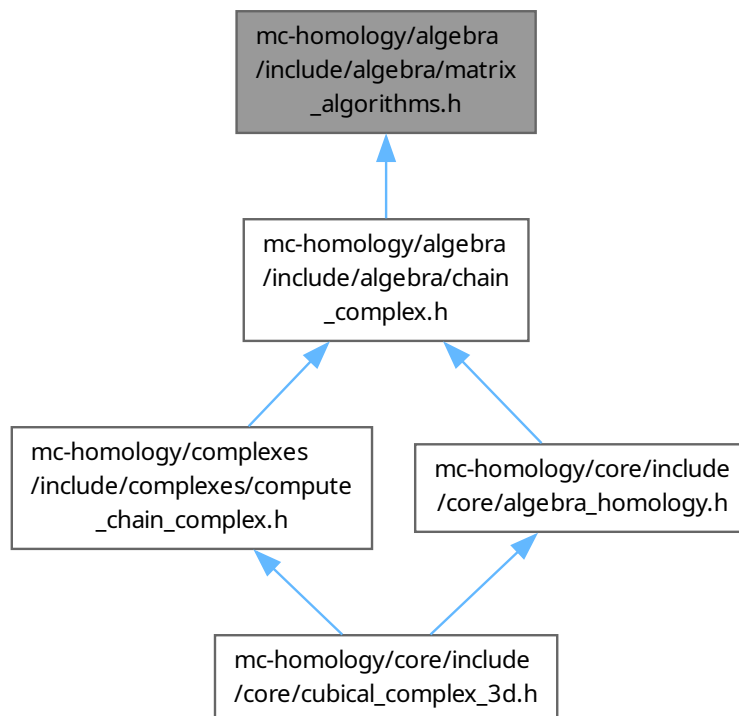
#include "algebra/detail/matrix_utils.h"
#include "algebra/matrix.h"

```

Include dependency graph for matrix\_algorithms.h:



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



## 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

<i>matrix</i>	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**

<i>in, out</i>	<i>matrix</i>	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**

<i>matrix</i>	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**

<i>in, out</i>	<i>matrix</i>	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.](#)

```

00001
00002
00003
00004 #pragma once
00005
00006 #include "algebra/detail/matrix_utils.h"
00007 #include "algebra/matrix.h"
00008
00009 namespace algebra {
00010
00011 template<Field T>
00012 constexpr std::size_t row_echelon_form(std::in_place_t, Matrix<T>& matrix) {
00013     std::size_t i = 0;
00014     for (std::size_t j = 0; j < matrix.ncols(); ++j) {
00015         auto maybe_i =
00016             detail::first_nonzero_submatrix_column_coefficient(matrix, i, j);
00017         if (!maybe_i) {
00018             continue;
00019         }
00020         if (*maybe_i != i) {
00021             detail::submatrix_swap_rows(matrix, i, *maybe_i, j);
00022         }
00023         for (auto k = i + 1; k < matrix.nrows(); ++k) {
00024             auto mult = -matrix[k, j] / matrix[i, j];
00025             detail::submatrix_add_row(matrix, mult, i, k, j);
00026         }
00027         ++i;
00028     }
00029     return i;
00030 }
00031
00032 template<class T>
00033 struct RowEchelonFormResult {
00034     Matrix<T> row_echelon_form = {};
00035     std::size_t non_empty_rows = 0;
00036 };
00037
00038 template<Field T>
00039 constexpr RowEchelonFormResult<T> row_echelon_form(Matrix<T> matrix) {
00040     auto i = row_echelon_form(std::in_place, matrix);
00041     return RowEchelonFormResult {
00042         .row_echelon_form = matrix,
00043         .non_empty_rows = i
00044     };
00045 }
00046
00047 template<EuclideanDomain T>
00048 constexpr std::size_t smith_form(std::in_place_t, Matrix<T>& matrix) {
00049     std::size_t k = 0;
00050     auto move_min_to_corner = [&] {
00051         auto min_element =
00052             detail::minimal_nonzero_submatrix_element(matrix, k, k);
00053         if (matrix[min_element.first, min_element.second] == T::zero()) {
00054             return;
00055         }
00056         if (min_element.first != k) {
00057             detail::submatrix_swap_rows(matrix, k, min_element.first, k);
00058         }
00059         if (min_element.second != k) {
00060             detail::submatrix_swap_cols(matrix, k, min_element.second, k);
00061         }
00062     };
00063     for (; k < std::min(matrix.nrows(), matrix.ncols()); ++k) {
00064         bool col_all_zeros = false;
00065         while (!col_all_zeros) {
00066             move_min_to_corner();
00067             if (matrix[k, k] == T::zero()) {
00068                 return k;
00069             }
00070             col_all_zeros = true;
00071             for (std::size_t i = k + 1; i < matrix.nrows(); ++i) {
00072                 auto [q, r] = divide(matrix[i, k], matrix[k, k]);
00073                 detail::submatrix_add_row(matrix, -q, k, i, k);
00074                 if (r != T::zero()) {
00075                     col_all_zeros = false;
00076                 }
00077             }
00078         }
00079         bool row_all_zeros = false;
00080         while (!row_all_zeros) {
00081             move_min_to_corner();
00082             if (matrix[k, k] == T::zero()) {
00083

```

```

00114         return k;
00115     }
00116     row_all_zeros = true;
00117     for (std::size_t j = k + 1; j < matrix.ncols(); ++j) {
00118         auto [q, r] = divide(matrix[k, j], matrix[k, k]);
00119         detail::submatrix_add_col(matrix, -q, k, j, k);
00120         if (r != T::zero()) {
00121             row_all_zeros = false;
00122         }
00123     }
00124 }
00125 }
00126 if constexpr (std::totally_ordered<T>) {
00127     for (std::size_t i = 0; i < k; ++i) {
00128         if (matrix[i, i] < T::zero()) {
00129             matrix[i, i] = -matrix[i, i];
00130         }
00131     }
00132 }
00133 return k;
00134 }
00135
00137 template<class T>
00138 struct SmithFormResult {
00140     Matrix<T> smith_form = {};
00141
00143     std::size_t non_empty = 0;
00144 };
00145
00156 template<EuclideanDomain T>
00157 constexpr SmithFormResult<T> smith_form(Matrix<T> matrix) {
00158     auto k = smith_form(std::in_place, matrix);
00159     return SmithFormResult {.smith_form = matrix, .non_empty = k};
00160 }
00161
00162 } // namespace algebra

```

## 7.11 mc-homology/algebra/include/algebra/modulo\_fields.h File Reference

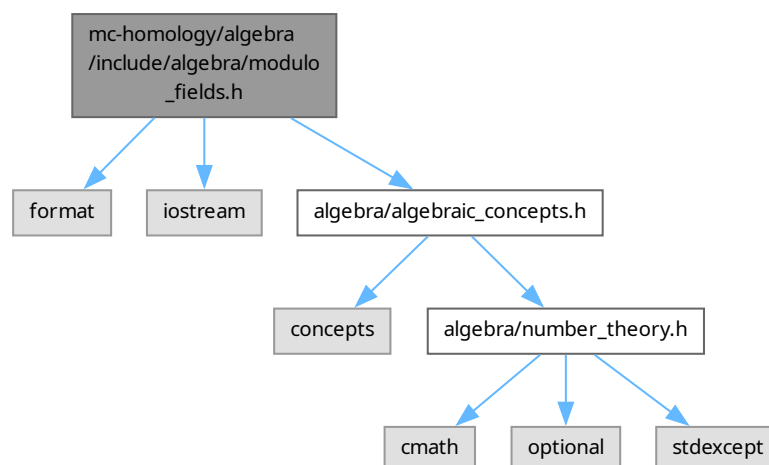
A file containing implementation of fields of integers mod P.

```

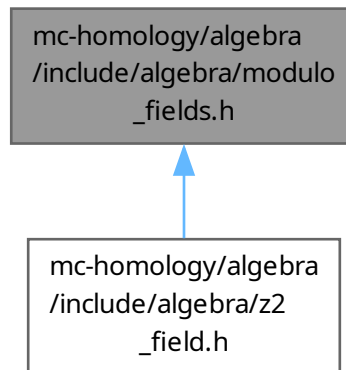
#include <format>
#include <iostream>
#include "algebra/algebraic_concepts.h"

```

Include dependency graph for modulo\_fields.h:



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



## 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.](#)

```

00001
00003
00004 #pragma once
00005
00006 #include <format>
00007 #include <iostream>
00008
00009 #include "algebra/algebraic_concepts.h"
00010
00011 namespace algebra {
00012
00013     template<int P>
00014     class ZModP {
00015     public:
00016         static_assert(is_prime(P));
00017
00018         constexpr ZModP() = default;
00019
00020         constexpr ZModP(int n) noexcept : m_inner_representation(modulo(n, P)) {}
00021
00022         constexpr static int p() noexcept {
00023             return P;
00024         }
00025
00026         constexpr static ZModP zero() noexcept {
00027             return ZModP(0);
00028         }
00029
00030         constexpr static ZModP one() noexcept {
00031             return ZModP(1);
00032         }
00033
00034         constexpr explicit operator int() const noexcept {
00035             return m_inner_representation;
00036         }
00037
00038         constexpr bool operator==(ZModP const&) const = default;
00039
00040         constexpr ZModP& operator+=(ZModP rhs) noexcept {
00041             m_inner_representation =
00042                 modulo(m_inner_representation + rhs.m_inner_representation, P);
00043             return *this;
00044         }
00045
00046         constexpr ZModP& operator-=(ZModP rhs) noexcept {
00047             m_inner_representation =
00048                 modulo(m_inner_representation - rhs.m_inner_representation, P);
00049             return *this;
00050         }
00051
00052         constexpr ZModP operator+() const noexcept {
00053             return *this;
00054         }
00055
00056         constexpr ZModP operator-() const noexcept {
00057             return ZModP(modulo(-m_inner_representation, P));
00058         }
00059
00060         constexpr ZModP& operator*=(ZModP rhs) noexcept {
00061             m_inner_representation =
00062                 modulo(m_inner_representation * rhs.m_inner_representation, P);
00063             return *this;
00064         }
00065
00066         constexpr int euclidean_function() const noexcept {
00067             return 1;
00068         }
00069
00070         constexpr ZModP& operator/=(ZModP rhs) {
00071             auto inverse = inverse_mod(rhs.m_inner_representation, P);
00072             if (!inverse) [[unlikely]] {
00073                 throw std::domain_error("Division by 0");
00074             }
00075         }
00076     };
00077
00078     constexpr ZModP operator+(ZModP lhs, ZModP rhs) {
00079         return lhs + rhs;
00080     }
00081
00082     constexpr ZModP operator-(ZModP lhs, ZModP rhs) {
00083         return lhs - rhs;
00084     }
00085
00086     constexpr ZModP operator*(ZModP lhs, ZModP rhs) {
00087         return lhs * rhs;
00088     }
00089
00090     constexpr ZModP operator/(ZModP lhs, ZModP rhs) {
00091         return lhs / rhs;
00092     }
00093
00094     constexpr int euclidean_function(int n) {
00095         return 1;
00096     }

```

```

00096     }
00097     return *this == *inverse;
00098 }
00099
00100 private:
00102     int m_inner_representation = 0;
00103 };
00104
00106 template<int P>
00107 constexpr ZModP<P> operator+(ZModP<P> lhs, ZModP<P> rhs) noexcept {
00108     return lhs += rhs;
00109 }
00110
00112 template<int P>
00113 constexpr ZModP<P> operator-(ZModP<P> lhs, ZModP<P> rhs) noexcept {
00114     return lhs -= rhs;
00115 }
00116
00118 template<int P>
00119 constexpr ZModP<P> operator*(ZModP<P> lhs, ZModP<P> rhs) noexcept {
00120     return lhs *= rhs;
00121 }
00122
00124 template<int P>
00125 constexpr ZModP<P> operator/(ZModP<P> lhs, ZModP<P> rhs) {
00126     return lhs /= rhs;
00127 }
00128
00130 template<int P>
00131 std::ostream& operator<<(std::ostream& output, ZModP<P> x) {
00132     return output << static_cast<int>(x);
00133 }
00134
00136 template<int P>
00137 constexpr inline bool is_commutative_v<ZModP<P>> = true;
00138
00139 } // namespace algebra
00140
00145 template<int P>
00146 struct std::formatter<algebra::ZModP<P>>: public std::formatter<int> {
00147     template<class FmtContext>
00148     FmtContext::iterator format(algebra::ZModP<P> x, FmtContext& ctx) const {
00149         return std::formatter<int>::format(static_cast<int>(x), ctx);
00150     }
00151 }
00152 };

```

## 7.13 mc-homology/algebra/include/algebra/number\_theory.h File Reference

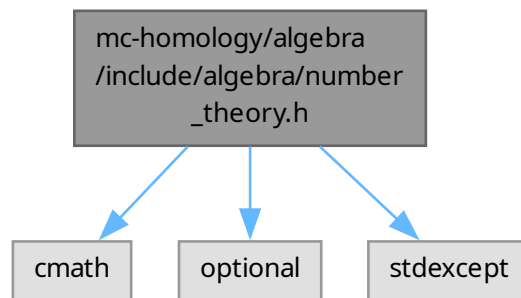
A file containing implementations of selected number theoretic algorithms.

```

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

```

Include dependency graph for number\_theory.h:



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



## 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

<i>a</i>	An integer
<i>b</i>	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

<i>a</i>	An integer
<i>b</i>	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

<i>a</i>	Inverted number
<i>n</i>	The modulus

#### Returns

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

### 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

<i>n</i>	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

<i>a</i>	An integer
----------	------------

$n$	an integer
-----	------------

### Returns

The remainder

## 7.14 number\_theory.h

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

```

00001
00004
00005 #pragma once
00006
00007 #include <cmath>
00008 #include <optional>
00009 #include <stdexcept>
00010
00011 namespace algebra {
00012
00021 constexpr bool is_prime(int n) noexcept {
00022     if (n < 2) {
00023         return false;
00024     }
00025     for (int i = 2; i * i <= n; ++i) {
00026         if (n % i == 0) {
00027             return false;
00028         }
00029     }
00030     return true;
00031 }
00032
00034 template<class T>
00035 struct DivResult {
00037     T quotient = {};
00039     T remainder = {};
00040 };
00041
00053 constexpr DivResult<int> divide(int a, int b) {
00054     if (b != 0) [[likely]] {
00055         auto [q, r] = std::div(a, b);
00056         if (r < 0) {
00057             q -= b > 0 ? 1 : -1;
00058             r += b > 0 ? b : -b;
00059         }
00060         return {.quotient = q, .remainder = r};
00061     } else [[unlikely]] {
00062         throw std::domain_error("Division by 0");
00063     }
00064 }
00065
00072 constexpr int modulo(int a, int n) {
00073     return divide(a, n).remainder;
00074 }
00075
00080 struct ExtendedGCDResult {
00082     int g = 0;
00084     int x = 0;
00086     int y = 0;
00087 };
00088
00101 constexpr ExtendedGCDResult extended_gcd(int a, int b) noexcept {
00102     a = std::abs(a);
00103     b = std::abs(b);
00104     int x1 = 1;
00105     int x2 = 0;
00106     int y1 = 0;
00107     int y2 = 1;
00108     while (b > 0) {
00109         auto [q, r] = divide(a, b);
00110         std::tie(a, b) = std::pair {b, r};
00111         std::tie(x1, x2) = std::pair {x2, x1 - q * x2};
00112         std::tie(y1, y2) = std::pair {y2, y1 - q * y2};
00113     }
00114     return {.g = a, .x = x1, .y = y1};
00115 }
00116

```

```

00125 constexpr std::optional<int> inverse_mod(int a, int n) noexcept {
00126     auto [g, x, _] = extended_gcd(a, n);
00127     if (g == 1) [[likely]] {
00128         return x;
00129     } else [[unlikely]] {
00130         return std::nullopt;
00131     }
00132 }
00133
00134 } // namespace algebra

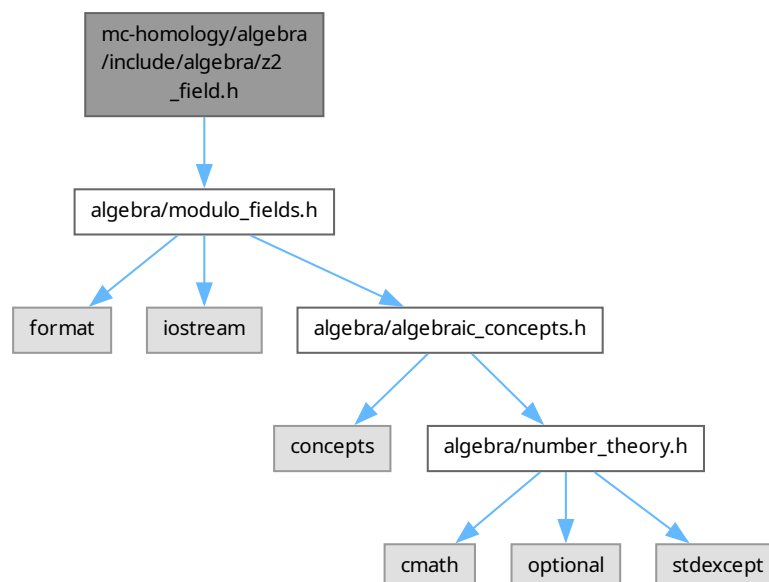
```

## 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.](#)

```

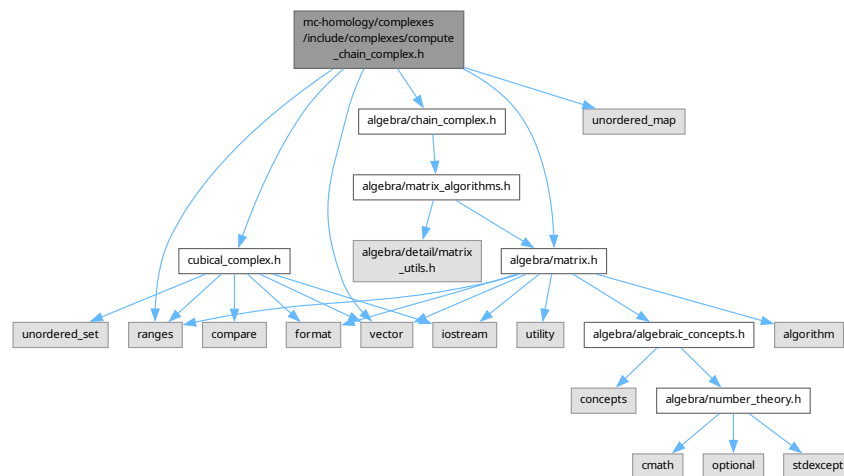
00001
00002
00003
00004 #pragma once
00005
00006 #include "algebra/modulo_fields.h"
00007
00008 namespace algebra {
00009
00010 template<
00011 class ZModP<2> {
00012 public:
00013     constexpr ZModP() = default;
00014
00015     constexpr ZModP(int n) noexcept : m_inner_representation(modulo(n, 2)) {}
00016
00017     constexpr static int p() noexcept {
00018         return 2;
00019     }
00020
00021     constexpr static ZModP zero() noexcept {
00022         return ZModP(0);
00023     }
00024
00025     constexpr static ZModP one() noexcept {
00026         return ZModP(1);
00027     }
00028
00029     constexpr explicit operator int() const noexcept {
00030         return m_inner_representation;
00031     }
00032
00033     constexpr bool operator==(ZModP const&) const = default;
00034
00035     constexpr ZModP& operator+=(ZModP rhs) noexcept {
00036         m_inner_representation ^= rhs.m_inner_representation;
00037         return *this;
00038     }
00039
00040     constexpr ZModP& operator-=(ZModP rhs) noexcept {
00041         m_inner_representation ^= rhs.m_inner_representation;
00042         return *this;
00043     }
00044
00045     constexpr ZModP operator+() const noexcept {
00046         return *this;
00047     }
00048
00049     constexpr ZModP operator-() const noexcept {
00050         return *this;
00051     }
00052
00053     constexpr ZModP& operator*=(ZModP rhs) noexcept {
00054         m_inner_representation &= rhs.m_inner_representation;
00055         return *this;
00056     }
00057
00058     constexpr int euclidean_function() const noexcept {
00059         return 1;
00060     }
00061
00062     constexpr ZModP& operator/=(ZModP rhs) {
00063         if (!rhs.m_inner_representation) [[unlikely]] {
00064             throw std::domain_error("Division by 0");
00065         }
00066         return *this;
00067     }
00068
00069 private:
00070     bool m_inner_representation = false;
00071 };
00072
00073 using Z2 = ZModP<2>;
00074 } // namespace algebra

```

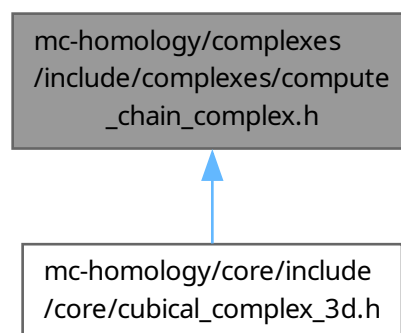
## 7.17 mc-homology/complexes/include/complexes/compute\_chain\_complex.h File Reference

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

<code>cubical_complex</code>	Complex to transform
------------------------------	----------------------

#### Returns

A chain complex

## 7.18 compute\_chain\_complex.h

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

```
00001
00004 #pragma once
00005
00006 #include <ranges>
00007 #include <unordered_map>
00008 #include <vector>
00009
00010 #include "algebra/chain_complex.h"
00011 #include "algebra/matrix.h"
00012 #include "cubical_complex.h"
00013
00014 namespace complexes {
00015
00023 template<class T>
00024 algebra::ChainComplex<T>
00025 compute_chain_complex(CubicalComplex const& cubical_complex) {
00026     namespace vs = std::views;
00027     auto const& simplices = cubical_complex.simplices();
00028     if (simplices.empty()) {
00029         return algebra::ChainComplex<T> {};
00030     }
00031     std::vector<algebra::Matrix<T> boundaries(simplices.size());
00032     // 0'th dimensional matrix is empty, we are computing non-reduced
```

```

00033 // homology
00034 boundaries[0] = algebra::Matrix<T>::zero(0, simplices[0].size());
00035 for (std::size_t dim = 1; dim < simplices.size(); ++dim) {
00036     boundaries[dim] = algebra::Matrix<T>::zero(
00037         simplices[dim - 1].size(),
00038         simplices[dim].size()
00039     );
00040     std::unordered_map<CubicalSimplex, std::size_t> assigned_rows;
00041     for (auto const& [s, i] : vs::zip(simplices[dim - 1], vs::iota(0))) {
00042         assigned_rows.emplace(s, i);
00043     }
00044     for (auto const& [j, simplex] : simplices[dim] | vs::enumerate) {
00045         std::array sgn = {1, -1, -1, 1};
00046         for (auto const& [k, bd] : simplex.boundary() | vs::enumerate) {
00047             boundaries[dim][assigned_rows[bd], j] = sgn[k % sgn.size()];
00048         }
00049     }
00050 }
00051 return algebra::ChainComplex<T> {std::move(boundaries)};
00052 }
00053
00054 } // namespace complexes

```

## 7.19 mc-homology/complexes/include/complexes/cubical\_complex.h

### File Reference

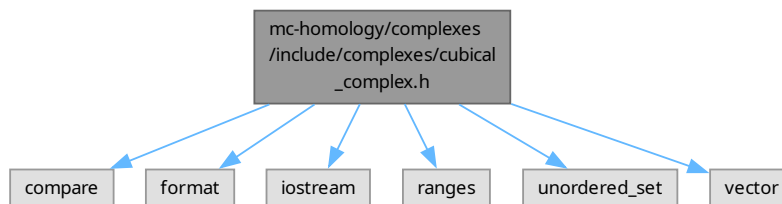
A file containing the CubicalComplex class.

```

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

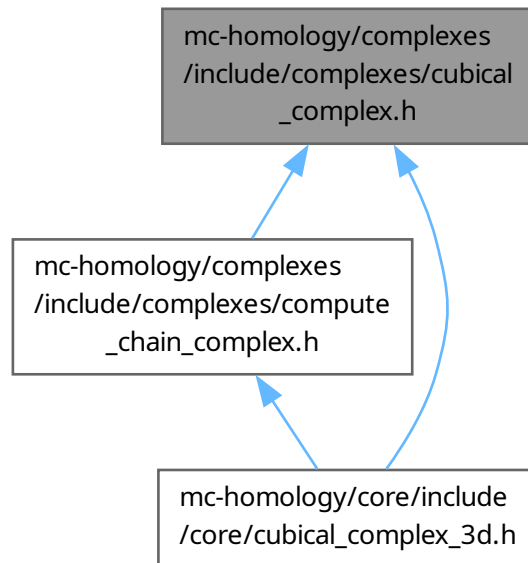
```

Include dependency graph for cubical\_complex.h:





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



## 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.](#)

```

00001
00003
00004 #pragma once
00005
00006 #include <compare>
00007 #include <format>
00008 #include <iostream>
00009 #include <ranges>
00010 #include <unordered_set>
00011 #include <vector>
00012
00013 namespace complexes {
00014
00015     class BasicInterval {
00016     public:
00017         BasicInterval();
00018
00019         std::size_t hash() const;
00020
00021         bool operator==(BasicInterval const&) const;
00022
00023         int left() const;
00024
00025         int right() const;
00026
00027         bool is_trivial() const;
00028
00029         static BasicInterval point(int p);
00030
00031         static BasicInterval interval(int left);
00032
00033     private:
00034         int m_left = 0;
00035         bool m_full = false;
00036     };
00037
00038     std::ostream& operator<<(std::ostream& output, BasicInterval i);
00039
00040     class CubicalSimplex {
00041     public:
00042         CubicalSimplex(std::vector<BasicInterval> intervals);
00043
00044         std::size_t dimension() const;
00045
00046         std::size_t ambient_dimension() const;
00047
00048         std::size_t hash() const;
00049
00050         std::vector<CubicalSimplex> boundary() const;
00051
00052         bool operator==(CubicalSimplex const&) const;
00053
00054         std::vector<BasicInterval> const& intervals() const;
00055
00056         std::strong_ordering operator<=>(CubicalSimplex const& simplex) const;
00057
00058         static CubicalSimplex point(int p);
00059
00060         static CubicalSimplex interval(int left);
00061
00062         friend CubicalSimplex
00063         product(CubicalSimplex const& s1, CubicalSimplex const& s2);
00064
00065     private:
00066         CubicalSimplex();
00067
00068         std::vector<BasicInterval> m_intervals = {};
00069         std::size_t m_dimension = 0;
00070     };
00071
00072     std::ostream& operator<<(std::ostream& output, CubicalSimplex const& s);
00073

```

```

00146 class CubicalComplex {
00147 public:
00151     CubicalComplex();
00152
00163     bool add(CubicalSimplex simplex);
00164
00168     void add_recursive(CubicalSimplex simplex);
00169
00179     bool remove(CubicalSimplex const& simplex);
00180
00187     bool contains(CubicalSimplex const& simplex) const;
00188
00190     bool operator==(CubicalComplex const&) const;
00191
00196     std::vector<std::unordered_set<CubicalSimplex>> const& simplices() const;
00197
00199     std::size_t dimension() const;
00200
00202     std::size_t ambient_dimension() const;
00203
00204 private:
00206     void add_recursive_impl(CubicalSimplex simplex);
00207
00209     std::vector<std::unordered_set<CubicalSimplex>> m_simplices;
00210 };
00211
00212 } // namespace complexes
00213
00215 template<>
00216 struct std::hash<complexes::BasicInterval> {
00218     std::size_t operator()(complexes::BasicInterval const& i) const;
00219 };
00220
00222 template<>
00223 struct std::hash<complexes::CubicalSimplex> {
00225     std::size_t operator()(complexes::CubicalSimplex const& s) const;
00226 };
00227
00232 template<>
00233 struct std::formatter<complexes::BasicInterval>: public std::formatter<int> {
00237     template<class FmtContext>
00238     FmtContext::iterator
00239     format(complexes::BasicInterval i, FmtContext& ctx) const {
00240         std::format_to(ctx.out(), "[");
00241         std::formatter<int>::format(i.left(), ctx);
00242         if (!i.is_trivial()) {
00243             std::format_to(ctx.out(), ", ");
00244             std::formatter<int>::format(i.right(), ctx);
00245         }
00246         std::format_to(ctx.out(), "];");
00247         return ctx.out();
00248     }
00249 };
00250
00255 template<>
00256 struct std::formatter<complexes::CubicalSimplex>:
00257     public std::formatter<complexes::BasicInterval> {
00261     template<class FmtContext>
00262     FmtContext::iterator
00263     format(complexes::CubicalSimplex const& s, FmtContext& ctx) const {
00264         namespace vs = std::views;
00265         std::formatter<complexes::BasicInterval>::format(
00266             s.intervals().front(),
00267             ctx
00268         );
00269         for (auto i : s.intervals() | vs::drop(1)) {
00270             std::format_to(ctx.out(), "x");
00271             std::formatter<complexes::BasicInterval>::format(i, ctx);
00272         }
00273         return ctx.out();
00274     }
00275 };

```

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

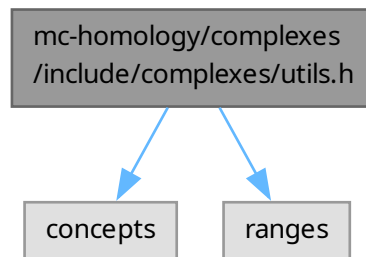
Contains auxillary classes and functions.

```

#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.](#)

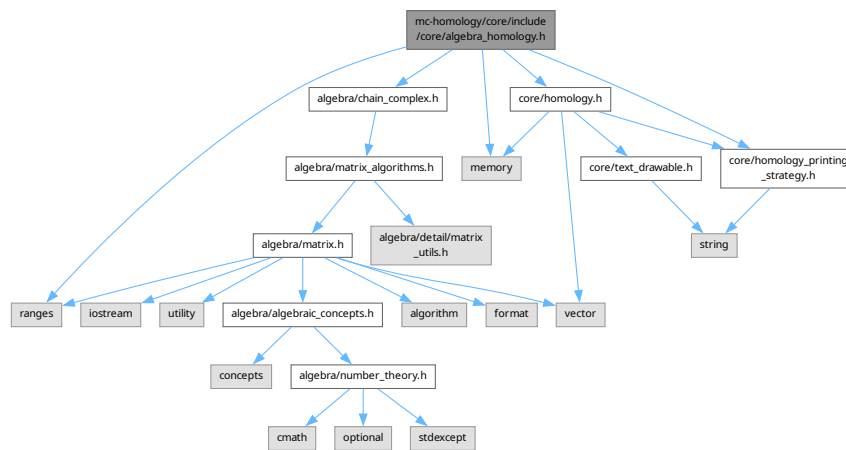
```

00001
00002
00003
00004 #include <concepts>
00005 #include <ranges>
00006
00007 namespace complexes {
00008     namespace utils {
00009
00010     std::size_t combine_hashes(std::size_t hash1, std::size_t hash2);
00011
00012     template<std::ranges::range R>
00013         requires requires(std::ranges::range_value_t<R> x) {
00014             {
00015                 std::hash<std::ranges::range_value_t<R> {}>(x)
00016             } -> std::convertible_to<std::size_t>;
00017         }
00018
00019     std::size_t hash_range(R&& r) {
00020         std::size_t hash = std::hash<std::size_t> {}(0);
00021         for (auto&& x : std::forward<R>(r)) {
00022             hash = combine_hashes(
00023                 hash,
00024                 std::hash<std::ranges::range_value_t<R> {}>(
00025                     std::forward<decltype(x)>(x)
00026                 )
00027             );
00028         }
00029     }
00030     return hash;
00031 }
00032
00033 } // namespace utils
00034 } // namespace complexes
  
```

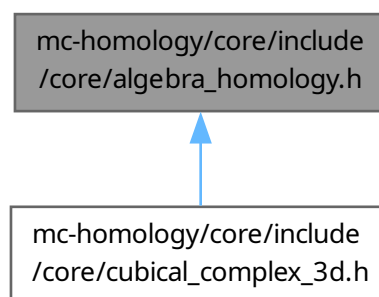
## 7.23 mc-homology/core/include/core/algebra\_homology.h File Reference

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.](#)

```

00001
00003 #pragma once
00004
00005 #include <memory>
00006 #include <ranges>
00007
00008 #include "algebra/chain_complex.h"
00009 #include "core/homology.h"
00010 #include "core/homology_printing_strategy.h"
00011
00012 namespace core {
00013
00015 template<class T>
00016 class AlgebraHomology: public Homology {
00017 public:
00019     AlgebraHomology(
00020         algebra::Homology<T> homology,
00021         std::unique_ptr<HomologyPrintingStrategy> printing_strategy =
00022             std::make_unique<HomologyRawPrint>()
00023     ) :
00024         Homology(std::move(printing_strategy)),
00025         m_homology(std::move(homology)) {}
00026
00028     std::vector<std::size_t> betti_numbers() const override {
00029         return m_homology.betti_numbers;
00030     }
00031
00033     std::vector<std::vector<std::string>> torsion() const override {
00034         namespace rs = std::ranges;
00035         namespace vs = std::views;
00036         return rs::to<std::vector>(
00037             m_homology.torsion | vs::transform([](auto const& tor) {
00038                 return rs::to<std::vector>(tor | vs::transform([](auto x) {
00039                     return std::format("{} ", x);
00040                 }));
00041             })
00042         );
00043     }
00044
00045 private:
00047     algebra::Homology<T> m_homology;
00048 };
00049
00050 } // namespace core

```

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

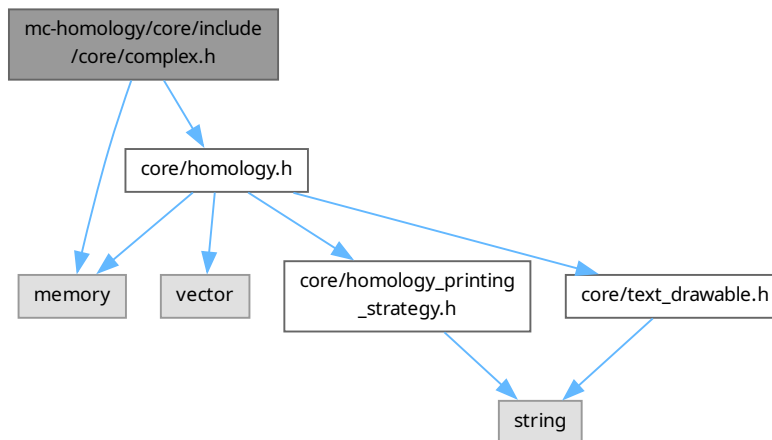
A file containing interface for the Complex class.

```

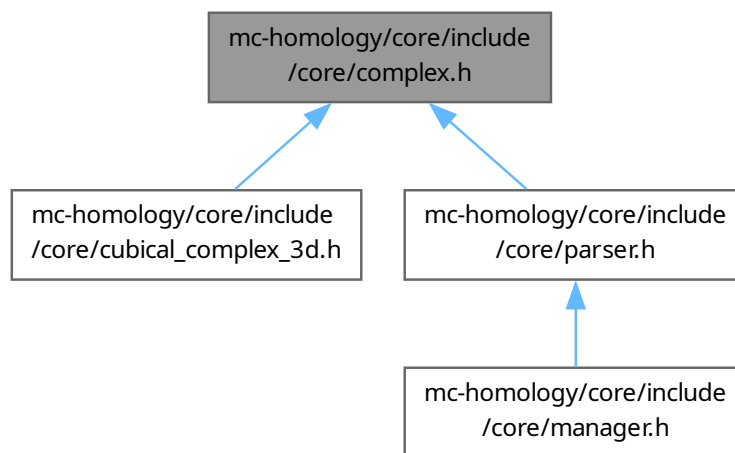
#include <memory>
#include "core/homology.h"

```

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.](#)

```

00001
00003 #pragma once
00004
00005 #include <memory>
00006
00007 #include "core/homology.h"
00008
00009 namespace core {
00010
00012 class Complex {
00013 public:
00015     virtual std::unique_ptr<Homology> z2_homology() const = 0;
00016
00018     virtual std::unique_ptr<Homology> z3_homology() const = 0;
00019
00021     virtual std::unique_ptr<Homology> z_homology() const = 0;
00022
00025     virtual void reduce() = 0;
00026
00028     virtual ~Complex();
00029 };
00030
00031 } // namespace core

```

## 7.27 mc-homology/core/include/core/cubical\_complex\_3d.h File Reference

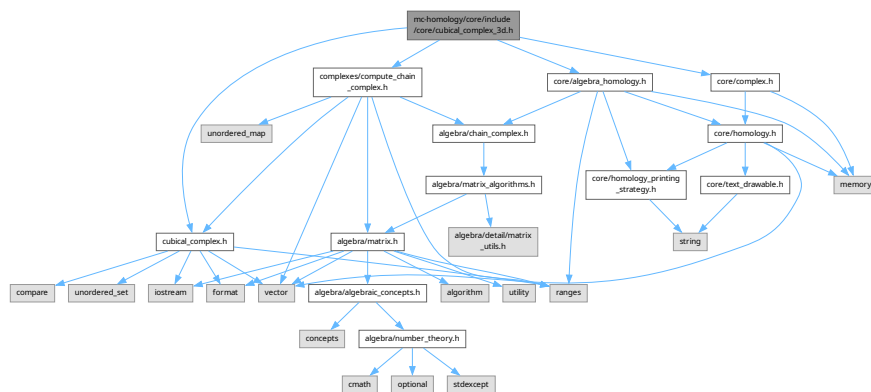
A file containing a concrete class CubicalComplex3D.

```

#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.](#)

```

00001
00003 #pragma once
00004
00005 #include "complexes/compute_chain_complex.h"
00006 #include "complexes/cubical_complex.h"
00007 #include "core/algebra_homology.h"
00008 #include "core/complex.h"
00009
00010 namespace core {
00011
00013 class CubicalComplex3D: public Complex {
00014 public:
00016     void add_cube(int x, int y, int z);
00017
00019     std::unique_ptr<Homology> z2_homology() const override;
00020
00022     std::unique_ptr<Homology> z3_homology() const override;
00023
00025     std::unique_ptr<Homology> z_homology() const override;
00026
00029     template<class T>
00030     std::unique_ptr<AlgebraHomology<T>> homology() const {
00031         return std::make_unique<AlgebraHomology<T>>(
00032             algebra::homology(complexes::compute_chain_complex<T>(m_inner))
00033         );
00034     }
00035
00038     void reduce() override;
00039
00040 private:
00042     complexes::CubicalComplex m_inner;
00043 };
00044
00045 } // namespace core

```

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

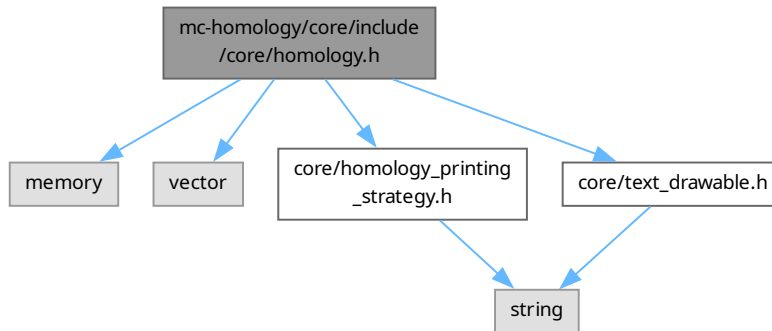
A file containing a Homology interface.

```

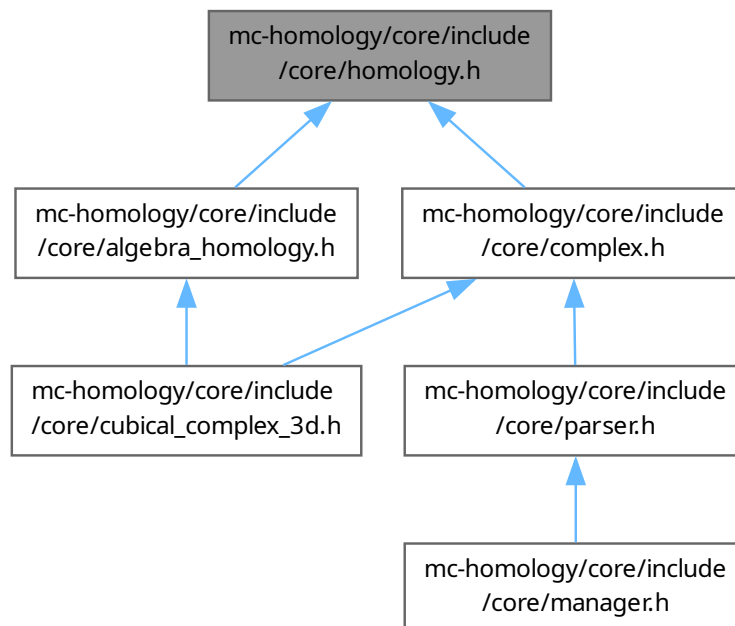
#include <memory>
#include <vector>
#include "core/homology_printing_strategy.h"
#include "core/text_drawable.h"

```

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.](#)

```

00001
00003 #pragma once
00004
00005 #include <memory>
00006 #include <vector>
00007
00008 #include "core/homology_printing_strategy.h"
00009 #include "core/text_drawable.h"
00010
00011 namespace core {
00012
00017 class Homology: public TextDrawable {
00018 public:
00025     Homology(
00026         std::unique_ptr<HomologyPrintingStrategy> printing_strategy =
00027             std::make_unique<HomologyRawPrint>()
00028     );
00029
00033     void select_strategy(
00034         std::unique_ptr<HomologyPrintingStrategy> printing_strategy
00035     );
00036
00039     std::string text() const override;
00040
00042     virtual std::vector<std::size_t> betti_numbers() const = 0;
00043
00046     virtual std::vector<std::vector<std::string>> torsion() const = 0;
00047
00048     // \brief Virtual destructor
00049     virtual ~Homology();
00050
00051 private:
00053     std::unique_ptr<HomologyPrintingStrategy> m_printing_strategy;
00054 };
00055
00056 } // namespace core

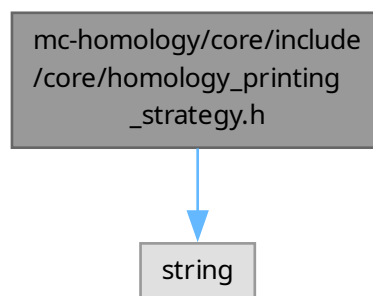
```

## 7.31 mc-homology/core/include/core/homology\_printing\_strategy.h File Reference

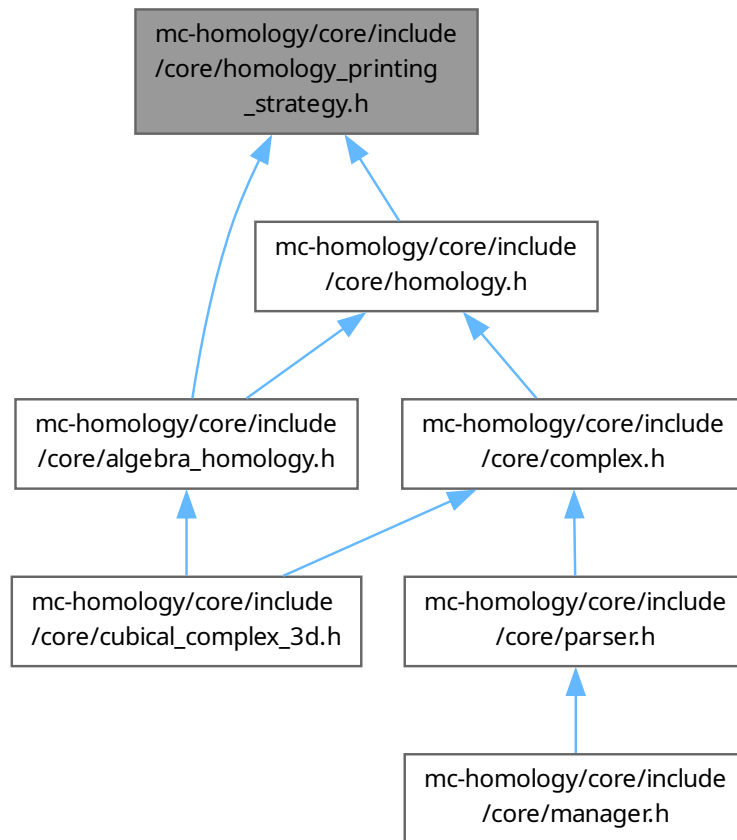
File containing strategies for printing homology.

```
#include <string>
```

Include dependency graph for homology\_printing\_strategy.h:



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



## 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.](#)

```

00001
00003 #pragma once
00004
00005 #include <string>
00006
00007 namespace core {
00008
00009 class Homology;
00010
00012 class HomologyPrintingStrategy {
00013 public:
00015     virtual std::string draw(Homology const& homology) const = 0;
00016
00018     virtual ~HomologyPrintingStrategy();
00019 };
00020
00022 class HomologyRawPrint: public HomologyPrintingStrategy {
00023 public:
00025     std::string draw(Homology const& homology) const override;
00026 };
00027
00032 class HomologyLatexPrint: public HomologyPrintingStrategy {
00033 public:
00041     HomologyLatexPrint(std::string ring_name, std::string homology_name = "H");
00042
00044     std::string draw(Homology const& homology) const override;
00045
00046 private:
00048     std::string m_ring_name;
00050     std::string m_homology_name;
00051 };
00052
00053 } // namespace core

```

## 7.33 mc-homology/core/include/core/latex\_wrapper.h File Reference

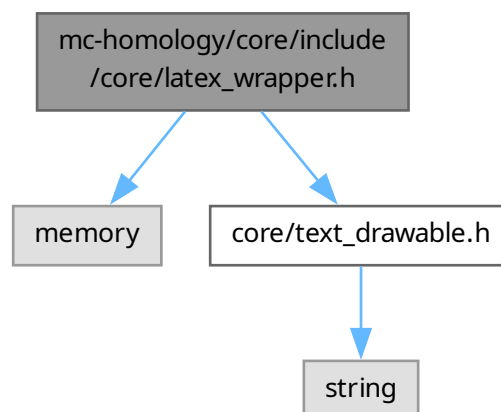
Turns a TextDrawable into a simple latex document.

```

#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.](#)

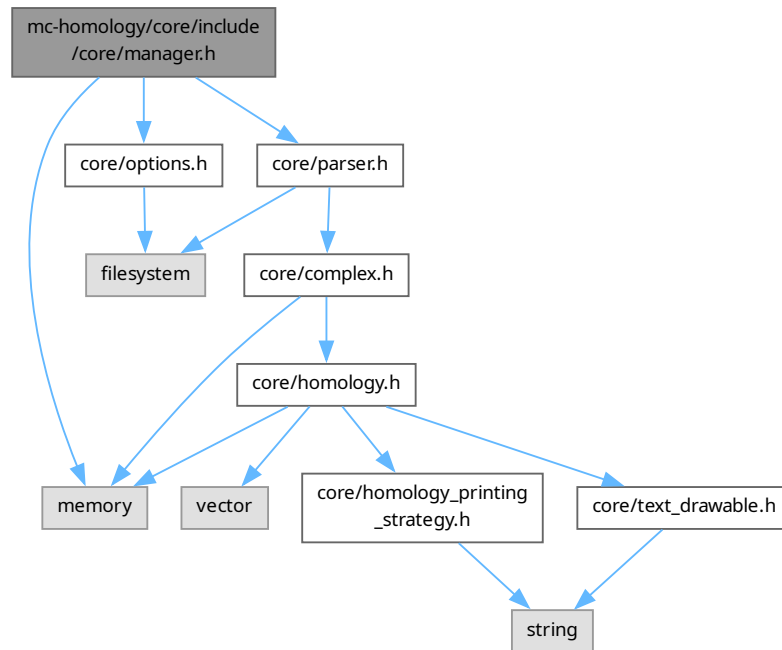
```
00001
00003 #pragma once
00004
00005 #include <memory>
00006
00007 #include "core/text_drawable.h"
00008
00009 namespace core {
00010
00013 class LatexWrapper: public TextDrawable {
00014 public:
00016     LatexWrapper(
00017         std::unique_ptr<TextDrawable> inner,
00018         std::string documentclass = "article"
00019     );
00020
00022     std::string text() const override;
00023
00024 private:
00026     std::unique_ptr<TextDrawable> m_inner;
00028     std::string m_documentclass;
00029 };
00030
00031 } // 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.](#)

```

00001
00002 #pragma once
00003
00004
00005 #include <memory>
00006
00007 #include "core/options.h"
00008 #include "core/parser.h"
00009
00010 namespace core {
00011
00012 class Manager {
00013 public:
00014     Manager(
00015         std::unique_ptr<Options> options,
00016         std::unique_ptr<MinecraftSavefileParser> parser
00017     );
00018
00019 };
00020

```

```

00022     void set_options(std::unique_ptr<Options> options);
00023
00025     void set_parser(std::unique_ptr<MinecraftSavefileParser> parser);
00026
00030     int run();
00031
00032 private:
00034     std::unique_ptr<Options> m_options;
00036     std::unique_ptr<MinecraftSavefileParser> m_parser;
00037 };
00038
00039 } // namespace core

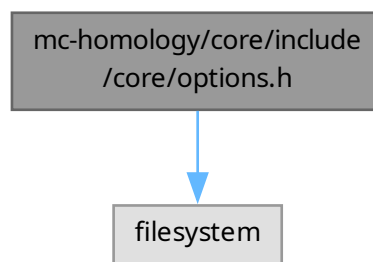
```

## 7.37 mc-homology/core/include/core/options.h File Reference

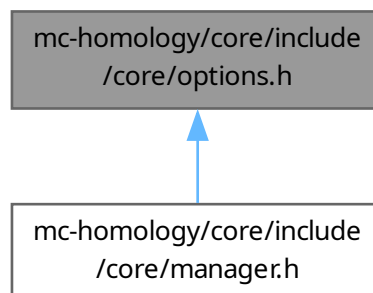
File containing an Options class for storing user options.

```
#include <filesystem>
```

Include dependency graph for options.h:



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





**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.](#)

```

00001
00003 #pragma once
00004
00005 #include <filesystem>
00006
00007 namespace core {
00008
00010 enum class HomologyChoice {
00011     Z,
00012     Z2,
00013     Z3,
00014 };
00015
00017 class Options {
00018 public:
00020     virtual std::filesystem::path filename() const = 0;
00021
00023     virtual std::pair<int, int> x_bounds() const = 0;
00024
00026     virtual std::pair<int, int> y_bounds() const = 0;
00027
00029     virtual std::pair<int, int> z_bounds() const = 0;
00030
00032     virtual HomologyChoice homology_to_compute() const = 0;
00033
00035     virtual bool latex() const = 0;
00036
00038     virtual bool help() const = 0;
00039
00041     virtual ~Options();
00042 };
00043
00046 class CommandLineOptions: public Options {
00047 public:
00049     CommandLineOptions(int argc, char** argv);
00050
00052     std::filesystem::path filename() const override;
00053
00055     std::pair<int, int> x_bounds() const override;
00056
00058     std::pair<int, int> y_bounds() const override;
00059
00061     std::pair<int, int> z_bounds() const override;
00062
00064     HomologyChoice homology_to_compute() const override;
00065
00067     bool latex() const override;
00068
00070     bool help() const override;

```

```

00071
00072 private:
00073     std::filesystem::path m_filename = "";
00074     std::pair<int, int> m_x_bounds = {0, 0};
00075     std::pair<int, int> m_y_bounds = {0, 0};
00076     std::pair<int, int> m_z_bounds = {0, 0};
00077     HomologyChoice m_homology_to_compute = HomologyChoice::Z2;
00078     bool m_latex = false;
00079     bool m_help = false;
00080 };
00081 // namespace core

```

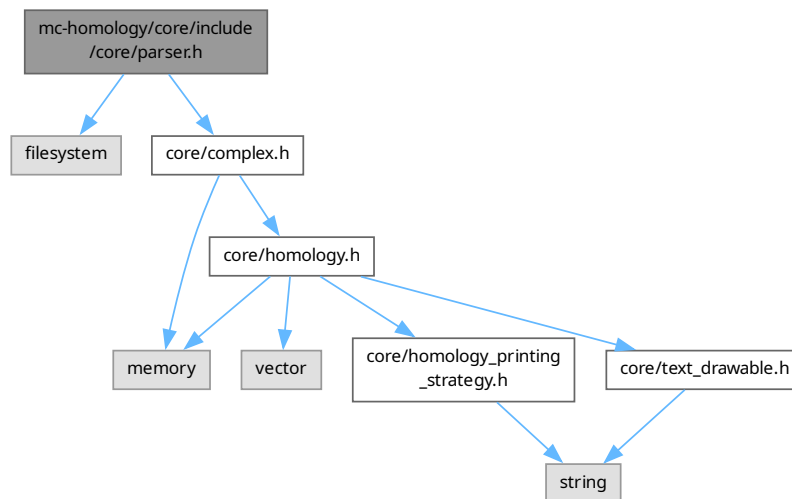
## 7.39 mc-homology/core/include/core/parser.h File Reference

A file containing a minecraft savefile parser.

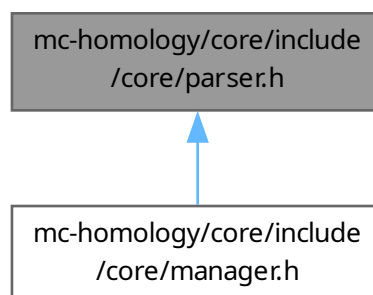
```
#include <filesystem>
```

```
#include "core/complex.h"
```

Include dependency graph for parser.h:



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



## 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.](#)

```

00001
00003 #pragma once
00004
00005 #include <filesystem>
00006
00007 #include "core/complex.h"
00008
00009 namespace core {
00010
00014 struct MinecraftCoordinates {
00016     int x = 0;
00018     int y = 0;
00020     int z = 0;
00021 };
00022
00024 class MinecraftSavefileParser {
00025 public:
00036     virtual std::unique_ptr<Complex> parse(
00037         std::filesystem::path const& path,
00038         MinecraftCoordinates lower_corner,
00039         MinecraftCoordinates upper_corner
00040     ) = 0;
00041     virtual ~MinecraftSavefileParser();
00042 };
00043
00046 class MinecraftSavefileParser_mcSavefileParsers:
00047     public MinecraftSavefileParser {
00048 public:
00059     std::unique_ptr<Complex> parse(
00060         std::filesystem::path const& path,
00061         MinecraftCoordinates lower_corner,
00062         MinecraftCoordinates upper_corner
00063     ) override;
00064 };
00065
00066 } // namespace core

```

## 7.41 mc-homology/core/include/core/polymorphic.h File Reference

A polymorphic type wrapper with value semantics.

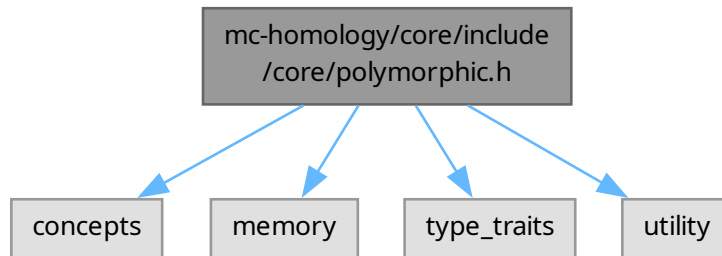
```

#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.*
- struct [core::Polymorphic< T >::Concept](#)  
*Base class used for type erasure.*
- struct [core::Polymorphic< T >::Model< Object >](#)  
*Concrete class used for type erasure.*

### 7.41.1 Detailed Description

A polymorphic type wrapper with value semantics.

This file contains a simplified implementation of `std::polymorphic` from C++26

## 7.42 polymorphic.h

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

```

00001
00006 #pragma once
00007
00008 #include <concepts>
00009 #include <memory>
00010 #include <type_traits>
00011 #include <utility>
00012
00013 namespace core {
00014
00015 #ifdef __cpp_lib_polymorphic
00016
00017 template<class T>
00018 using Polymorphic = std::polymorphic<T>
00019 #else
00020
00022 template<class T>
00023 class Polymorphic {
00024 public:
00026     constexpr explicit Polymorphic()
00027         requires std::default_initializable<T> && std::copy_constructible<T>

```

```

00028         : Polymorphic(T {}) {}
00029
00031     template<class U = T>
00032         requires std::derived_from<std::remove_cvref_t<U>, T>
00033         && std::copy_constructible<std::remove_cvref_t<U>
00034         && std::constructible_from<std::remove_cvref_t<U>, U>
00035     constexpr explicit Polymorphic(U&& u) :
00036         m_object(
00037             std::make_unique<Model<std::remove_cvref_t<U>>>(std::forward<U>(u))
00038         ) {}
00039
00041     template<class U, class... Args>
00042         requires std::derived_from<std::remove_cvref_t<U>, T>
00043         && std::copy_constructible<std::remove_cvref_t<U>
00044         && std::constructible_from<std::remove_cvref_t<U>, Args...>
00045         && std::same_as<std::remove_cvref_t<U>, U>
00046     constexpr explicit Polymorphic(std::in_place_type_t<U>, Args&&... args) :
00047         m_object(
00048             std::make_unique<Model<std::remove_cvref_t<U>>>(
00049                 std::forward<Args>(args)...
00050             )
00051         ) {}
00052
00054     constexpr Polymorphic(Polymorphic const& other) :
00055         m_object(other.m_object->clone()) {}
00056
00058     constexpr Polymorphic(Polymorphic&& other) noexcept :
00059         m_object(std::move(other.m_object)) {}
00060
00062     constexpr Polymorphic& operator=(Polymorphic const& other) {
00063         m_object = other.m_object->clone();
00064         return *this;
00065     }
00066
00068     constexpr Polymorphic& operator=(Polymorphic&& other) noexcept {
00069         m_object = std::move(other.m_object);
00070         return *this;
00071     }
00072
00074     constexpr T const* operator->() const noexcept {
00075         return m_object->get_address();
00076     }
00077
00079     constexpr T* operator->() noexcept {
00080         return m_object->get_address();
00081     }
00082
00084     constexpr T const& operator*() const noexcept {
00085         return *m_object->get_address();
00086     }
00087
00089     constexpr T& operator*() noexcept {
00090         return *m_object->get_address();
00091     }
00092
00094     constexpr bool valueless_after_move() const noexcept {
00095         return m_object == nullptr;
00096     }
00097
00099     constexpr void swap(Polymorphic& other) noexcept {
00100         using std::swap;
00101         swap(m_object, other.m_object);
00102     }
00103
00104 private:
00106     struct Concept {
00108         constexpr virtual T const* get_address() const noexcept = 0;
00110         constexpr virtual T* get_address() noexcept = 0;
00112         constexpr virtual std::unique_ptr<Concept> clone() const = 0;
00114         constexpr virtual ~Concept() = default;
00115     };
00116
00121     template<class Object>
00122     struct Model: public Concept {
00124         template<class U = Object>
00125         constexpr Model(U&& value) : m_value(std::forward<U>(value)) {}
00126
00128         constexpr T const* get_address() const noexcept override {
00129             return std::addressof(m_value);
00130         }
00131
00133         constexpr T* get_address() noexcept override {
00134             return std::addressof(m_value);
00135         }
00136
00139         constexpr std::unique_ptr<Concept> clone() const override {
00140             return std::make_unique<Model<Object>>(m_value);

```

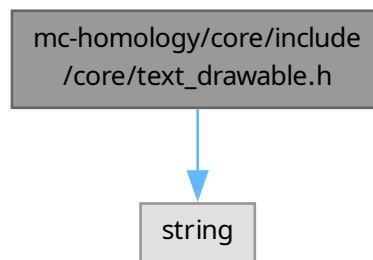
```
00141     }
00142
00143     private:
00144         Object m_value;
00145     };
00146
00147     std::unique_ptr<Concept> m_object;
00151 };
00152
00153 #endif
00154
00155 } // namespace core
```

## 7.43 mc-homology/core/include/core/text\_drawable.h File Reference

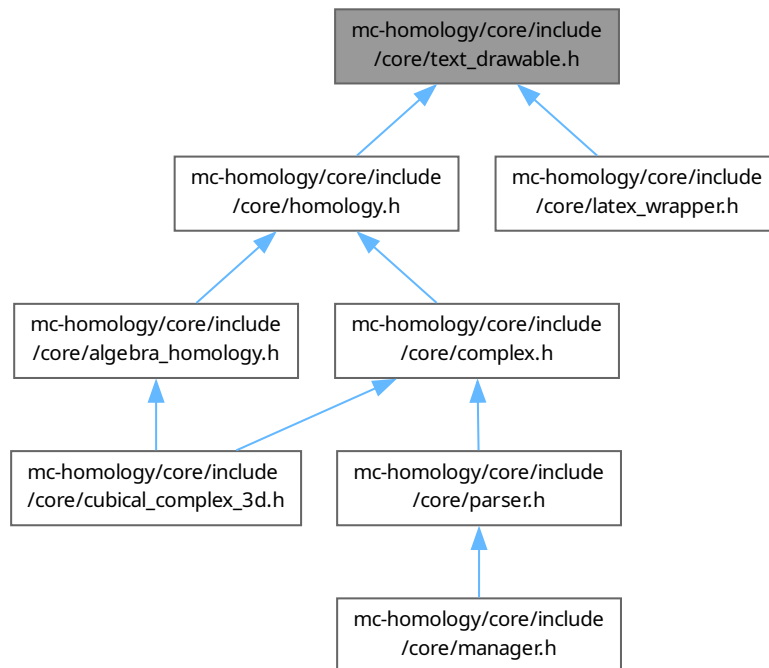
A file containing a TextDrawable interface, for classes that can be drawn on screen in the form of text.

```
#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.](#)

```

00001
00004 #pragma once
00005
00006 #include <string>
00007
00008 namespace core {
00009
00012 class TextDrawable {
00013 public:
00015     virtual std::string text() const = 0;
00016
00018     virtual ~TextDrawable();
00019 };
00020
00021 } // namespace core
  
```





# Index

- add
  - complexes::CubicalComplex, 35
- add\_recursive
  - complexes::CubicalComplex, 35
- algebra::AdditiveGroup, 9
- algebra::ChainComplex< T >, 21
  - boundary, 23
  - ChainComplex, 22
- algebra::Commutative, 10
- algebra::CommutativeRing, 10
- algebra::DivResult< T >, 44
- algebra::EuclideanDomain, 11
- algebra::ExtendedGCDResult, 46
- algebra::Field, 11
- algebra::Group, 12
- algebra::Homology< T >, 57
  - torsion, 59
- algebra::Integer, 70
  - euclidean\_function, 71
- algebra::Matrix< T >, 76
  - at, 80
  - Matrix, 80
  - operator[], 82
  - operator\*=: 82
- algebra::Ring, 13
- algebra::RowEchelonFormResult< T >, 97
- algebra::SkipCorrectnessCheckT, 98
- algebra::SmithFormResult< T >, 98
- algebra::ZModP< 2 >, 105
  - euclidean\_function, 107
- algebra::ZModP< P >, 102
  - euclidean\_function, 104
- at
  - algebra::Matrix< T >, 80
- beti\_numbers
  - core::AlgebraHomology< T >, 18
  - core::Homology, 62
- boundary
  - algebra::ChainComplex< T >, 23
  - complexes::CubicalSimplex, 43
- ChainComplex
  - algebra::ChainComplex< T >, 22
- clone
  - core::Polymorphic< T >::Concept, 32
  - core::Polymorphic< T >::Model< Object >, 90
- complexes::BasicInterval, 19
  - interval, 20
- complexes::CubicalComplex, 33
  - add, 35
  - add\_recursive, 35
  - contains, 35
  - CubicalComplex, 34
  - remove, 35
  - simplices, 36
- complexes::CubicalSimplex, 40
  - boundary, 43
  - CubicalSimplex, 43
  - interval, 43
  - operator<= >, 43
  - point, 43
  - product, 44
- compute\_chain\_complex
  - compute\_chain\_complex.h, 141
- compute\_chain\_complex.h
  - compute\_chain\_complex, 141
- contains
  - complexes::CubicalComplex, 35
- core::AlgebraHomology< T >, 15
  - beti\_numbers, 18
  - torsion, 18
- core::CommandLineOptions, 23
  - filename, 26
  - help, 26
  - homology\_to\_compute, 27
  - latex, 27
  - x\_bounds, 27
  - y\_bounds, 27
  - z\_bounds, 27
- core::Complex, 28
  - reduce, 29
  - z2\_homology, 29
  - z3\_homology, 30
  - z\_homology, 30
- core::CubicalComplex3D, 36
  - reduce, 39
  - z2\_homology, 39
  - z3\_homology, 39
  - z\_homology, 39
- core::Homology, 59
  - beti\_numbers, 62
  - Homology, 62
  - select\_strategy, 62
  - text, 63
  - torsion, 63
- core::HomologyLatexPrint, 63
  - draw, 66
  - HomologyLatexPrint, 66

- core::HomologyPrintingStrategy, 67
  - draw, 68
- core::HomologyRawPrint, 68
  - draw, 69
- core::LatexWrapper, 72
  - text, 74
- core::Manager, 74
  - run, 76
- core::MinecraftCoordinates, 83
- core::MinecraftSavefileParser, 84
  - parse, 85
- core::MinecraftSavefileParser\_mcSavefileParsers, 85
  - parse, 87
- core::Options, 91
  - filename, 93
  - help, 93
  - homology\_to\_compute, 93
  - latex, 93
  - x\_bounds, 93
  - y\_bounds, 93
  - z\_bounds, 93
- core::Polymorphic< T >, 94
- core::Polymorphic< T >::Concept, 30
  - clone, 32
  - get\_address, 32
- core::Polymorphic< T >::Model< Object >, 87
  - clone, 90
  - get\_address, 90
- core::TextDrawable, 99
  - text, 101
- CubicalComplex
  - complexes::CubicalComplex, 34
- CubicalSimplex
  - complexes::CubicalSimplex, 43
- divide
  - integer.h, 116
  - number\_theory.h, 135
- draw
  - core::HomologyLatexPrint, 66
  - core::HomologyPrintingStrategy, 68
  - core::HomologyRawPrint, 69
- euclidean\_function
  - algebra::Integer, 71
  - algebra::ZModP< 2 >, 107
  - algebra::ZModP< P >, 104
- extended\_gcd
  - number\_theory.h, 135
- filename
  - core::CommandLineOptions, 26
  - core::Options, 93
- format
  - std::formatter< algebra::Integer >, 48
  - std::formatter< algebra::Matrix< T > >, 50
  - std::formatter< complexes::BasicInterval >, 53
  - std::formatter< complexes::CubicalSimplex >, 56
- get\_address
  - core::Polymorphic< T >::Concept, 32
  - core::Polymorphic< T >::Model< Object >, 90
- help
  - core::CommandLineOptions, 26
  - core::Options, 93
- Homology
  - core::Homology, 62
- homology\_to\_compute
  - core::CommandLineOptions, 27
  - core::Options, 93
- HomologyLatexPrint
  - core::HomologyLatexPrint, 66
- integer.h
  - divide, 116
  - modulo, 116
- interval
  - complexes::BasicInterval, 20
  - complexes::CubicalSimplex, 43
- inverse\_mod
  - number\_theory.h, 135
- is\_prime
  - number\_theory.h, 136
- latex
  - core::CommandLineOptions, 27
  - core::Options, 93
- Matrix
  - algebra::Matrix< T >, 80
- matrix.h
  - operator\*, 120
- matrix\_algorithms.h
  - row\_echelon\_form, 126
  - smith\_form, 127
- mc-homology/algebra/include/algebra/algebraic\_concepts.h, 109, 111
- mc-homology/algebra/include/algebra/chain\_complex.h, 111, 113
- mc-homology/algebra/include/algebra/integer.h, 115, 117
- mc-homology/algebra/include/algebra/matrix.h, 118, 120
- mc-homology/algebra/include/algebra/matrix\_algorithms.h, 124, 128
- mc-homology/algebra/include/algebra/modulo\_fields.h, 129, 131
- mc-homology/algebra/include/algebra/number\_theory.h, 132, 137
- mc-homology/algebra/include/algebra/z2\_field.h, 138, 139
- mc-homology/complexes/include/complexes/compute\_chain\_complex.h, 140, 141
- mc-homology/complexes/include/complexes/cubical\_complex.h, 142, 144
- mc-homology/complexes/include/complexes/utils.h, 145, 146

- mc-homology/core/include/core/algebra\_homology.h, 147, 148
- mc-homology/core/include/core/complex.h, 148, 150
- mc-homology/core/include/core/cubical\_complex\_3d.h, 150, 151
- mc-homology/core/include/core/homology.h, 151, 153
- mc-homology/core/include/core/homology\_printing\_strategy.h, 153, 155
- mc-homology/core/include/core/latex\_wrapper.h, 155, 156
- mc-homology/core/include/core/manager.h, 156, 157
- mc-homology/core/include/core/options.h, 158, 159
- mc-homology/core/include/core/parser.h, 160, 161
- mc-homology/core/include/core/polymorphic.h, 161, 162
- mc-homology/core/include/core/text\_drawable.h, 164, 165
- modulo
  - integer.h, 116
  - number\_theory.h, 136
- number\_theory.h
  - divide, 135
  - extended\_gcd, 135
  - inverse\_mod, 135
  - is\_prime, 136
  - modulo, 136
- operator<=>
  - complexes::CubicalSimplex, 43
- operator[]
  - algebra::Matrix< T >, 82
- operator\*
  - matrix.h, 120
- operator\*=
  - algebra::Matrix< T >, 82
- parse
  - core::MinecraftSavefileParser, 85
  - core::MinecraftSavefileParser\_mcSavefileParsers, 87
- point
  - complexes::CubicalSimplex, 43
- product
  - complexes::CubicalSimplex, 44
- reduce
  - core::Complex, 29
  - core::CubicalComplex3D, 39
- remove
  - complexes::CubicalComplex, 35
- row\_echelon\_form
  - matrix\_algorithms.h, 126
- run
  - core::Manager, 76
- select\_strategy
  - core::Homology, 62
- simplices
  - complexes::CubicalComplex, 36
- smith\_form
  - matrix\_algorithms.h, 127
- std::formatter< algebra::Integer >, 47
  - format, 48
- std::formatter< algebra::Matrix< T > >, 48
  - format, 50
- std::formatter< algebra::ZModP< P > >, 50
- std::formatter< complexes::BasicInterval >, 51
  - format, 53
- std::formatter< complexes::CubicalSimplex >, 53
  - format, 56
- std::hash< complexes::BasicInterval >, 56
- std::hash< complexes::CubicalSimplex >, 57
- text
  - core::Homology, 63
  - core::LatexWrapper, 74
  - core::TextDrawable, 101
- torsion
  - algebra::Homology< T >, 59
  - core::AlgebraHomology< T >, 18
  - core::Homology, 63
- x\_bounds
  - core::CommandLineOptions, 27
  - core::Options, 93
- y\_bounds
  - core::CommandLineOptions, 27
  - core::Options, 93
- z2\_homology
  - core::Complex, 29
  - core::CubicalComplex3D, 39
- z3\_homology
  - core::Complex, 30
  - core::CubicalComplex3D, 39
- z\_bounds
  - core::CommandLineOptions, 27
  - core::Options, 93
- z\_homology
  - core::Complex, 30
  - core::CubicalComplex3D, 39