

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() . . . . .                                       | 18 |
| 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 . . . . . | 20 |
| 6.3.1 Detailed Description . . . . .                              | 21 |
| 6.3.2 Constructor & Destructor Documentation . . . . .            | 21 |
| 6.3.2.1 ChainComplex() [1/2] . . . . .                            | 21 |
| 6.3.2.2 ChainComplex() [2/2] . . . . .                            | 22 |
| 6.3.3 Member Function Documentation . . . . .                     | 22 |
| 6.3.3.1 boundary() . . . . .                                      | 22 |
| 6.4 core::CommandlineOptions Class Reference . . . . .            | 22 |
| 6.4.1 Detailed Description . . . . .                              | 25 |
| 6.4.2 Member Function Documentation . . . . .                     | 25 |
| 6.4.2.1 filename() . . . . .                                      | 25 |
| 6.4.2.2 help() . . . . .  | 25 |
| 6.4.2.3 homology_to_compute() . . . . .                           | 25 |
| 6.4.2.4 latex() . . . . .   | 26 |
| 6.4.2.5 x_bounds() . . . . .                                      | 26 |
| 6.4.2.6 y_bounds() . . . . .                                      | 26 |
| 6.4.2.7 z_bounds() . . . . .                                      | 26 |
| 6.5 core::Complex Class Reference . . . . .                       | 27 |
| 6.5.1 Detailed Description . . . . .                              | 28 |
| 6.5.2 Member Function Documentation . . . . .                     | 28 |
| 6.5.2.1 reduce() . . . . .  | 28 |
| 6.5.2.2 z2_homology() . . . . .                                   | 29 |
| 6.5.2.3 z3_homology() . . . . .                                   | 29 |
| 6.5.2.4 z_homology() . . . . .                                    | 29 |
| 6.6 complexes::CubicalComplex Class Reference . . . . .           | 29 |
| 6.6.1 Detailed Description . . . . .                              | 30 |
| 6.6.2 Constructor & Destructor Documentation . . . . .            | 30 |
| 6.6.2.1 CubicalComplex() . . . . .                                | 30 |
| 6.6.3 Member Function Documentation . . . . .                     | 30 |
| 6.6.3.1 add() . . . . .   | 30 |
| 6.6.3.2 add_recursive() . . . . .                                 | 31 |
| 6.6.3.3 contains() . . . . .                                      | 31 |
| 6.6.3.4 remove() . . . . .  | 31 |
| 6.6.3.5 simplices() . . . . .                                     | 32 |

---

|   |    |
|---|----|
| 6.7 core::CubicalComplex3D Class Reference . . . . .                            | 32 |
| 6.7.1 Detailed Description . . . . .  | 34 |
| 6.7.2 Member Function Documentation . . . . .                                   | 34 |
| 6.7.2.1 reduce() . . . . .  | 34 |
| 6.7.2.2 z2_homology() . . . . .   | 34 |
| 6.7.2.3 z3_homology() . . . . .   | 34 |
| 6.7.2.4 z_homology() . . . . .  | 34 |
| 6.8 complexes::CubicalSimplex Class Reference . . . . .                         | 35 |
| 6.8.1 Detailed Description . . . . .  | 36 |
| 6.8.2 Constructor & Destructor Documentation . . . . .                          | 36 |
| 6.8.2.1 CubicalSimplex() . . . . .  | 36 |
| 6.8.3 Member Function Documentation . . . . .                                   | 36 |
| 6.8.3.1 boundary() . . . . .  | 36 |
| 6.8.3.2 interval() . . . . .  | 37 |
| 6.8.3.3 operator<=>() . . . . .   | 37 |
| 6.8.3.4 point() . . . . .   | 37 |
| 6.8.4 Friends And Related Symbol Documentation . . . . .                        | 37 |
| 6.8.4.1 product . . . . .   | 37 |
| 6.9 algebra::DivResult< T > Struct Template Reference . . . . .                 | 38 |
| 6.9.1 Detailed Description . . . . .  | 39 |
| 6.10 algebra::ExtendedGCDResult Struct Reference . . . . .                      | 39 |
| 6.10.1 Detailed Description . . . . .   | 39 |
| 6.11 std::formatter< algebra::Integer > Struct Reference . . . . .              | 40 |
| 6.11.1 Detailed Description . . . . .   | 41 |
| 6.11.2 Member Function Documentation . . . . .                                  | 41 |
| 6.11.2.1 format() . . . . .   | 41 |
| 6.12 std::formatter< algebra::Matrix< T > > Struct Template Reference . . . . . | 41 |
| 6.12.1 Detailed Description . . . . .   | 42 |
| 6.12.2 Format syntax . . . . .  | 42 |
| 6.12.3 Example . . . . .  | 42 |
| 6.12.4 Member Function Documentation . . . . .                                  | 42 |
| 6.12.4.1 format() . . . . .   | 42 |
| 6.13 std::formatter< algebra::ZModP< P > > Struct Template Reference . . . . .  | 43 |
| 6.13.1 Detailed Description . . . . .   | 44 |
| 6.14 std::formatter< complexes::BasicInterval > Struct Reference . . . . .      | 44 |
| 6.14.1 Detailed Description . . . . .   | 45 |
| 6.14.2 Member Function Documentation . . . . .                                  | 45 |
| 6.14.2.1 format() . . . . .   | 45 |
| 6.15 std::formatter< complexes::CubicalSimplex > Struct Reference . . . . .     | 46 |
| 6.15.1 Detailed Description . . . . .   | 47 |
| 6.15.2 Member Function Documentation . . . . .                                  | 48 |
| 6.15.2.1 format() . . . . .   | 48 |

|  |    |
|--|----|
| 6.16 std::hash< complexes::BasicInterval > Struct Reference . . . . .  | 48 |
| 6.16.1 Detailed Description . . . . .                                  | 48 |
| 6.17 std::hash< complexes::CubicalSimplex > Struct Reference . . . . . | 49 |
| 6.17.1 Detailed Description . . . . .                                  | 49 |
| 6.18 algebra::Homology< T > Struct Template Reference . . . . .        | 49 |
| 6.18.1 Detailed Description . . . . .                                  | 50 |
| 6.18.2 Member Data Documentation . . . . .                             | 51 |
| 6.18.2.1 torsion . . . . .   | 51 |
| 6.19 core::Homology Class Reference . . . . .                          | 51 |
| 6.19.1 Detailed Description . . . . .                                  | 54 |
| 6.19.2 Constructor & Destructor Documentation . . . . .                | 54 |
| 6.19.2.1 Homology() . . . . .  | 54 |
| 6.19.3 Member Function Documentation . . . . .                         | 54 |
| 6.19.3.1 betti_numbers() . . . . .                                     | 54 |
| 6.19.3.2 select_strategy() . . . . .                                   | 54 |
| 6.19.3.3 text() . . . . .  | 54 |
| 6.19.3.4 torsion() . . . . .   | 55 |
| 6.20 core::HomologyLatexPrint Class Reference . . . . .                | 55 |
| 6.20.1 Detailed Description . . . . .                                  | 56 |
| 6.20.2 Constructor & Destructor Documentation . . . . .                | 57 |
| 6.20.2.1 HomologyLatexPrint() . . . . .                                | 57 |
| 6.20.3 Member Function Documentation . . . . .                         | 58 |
| 6.20.3.1 draw() . . . . .  | 58 |
| 6.21 core::HomologyPrintingStrategy Class Reference . . . . .          | 58 |
| 6.21.1 Detailed Description . . . . .                                  | 59 |
| 6.21.2 Member Function Documentation . . . . .                         | 59 |
| 6.21.2.1 draw() . . . . .  | 59 |
| 6.22 core::HomologyRawPrint Class Reference . . . . .                  | 60 |
| 6.22.1 Detailed Description . . . . .                                  | 61 |
| 6.22.2 Member Function Documentation . . . . .                         | 61 |
| 6.22.2.1 draw() . . . . .  | 61 |
| 6.23 algebra::Integer Class Reference . . . . .                        | 61 |
| 6.23.1 Detailed Description . . . . .                                  | 63 |
| 6.23.2 Member Function Documentation . . . . .                         | 63 |
| 6.23.2.1 euclidean_function() . . . . .                                | 63 |
| 6.24 core::LatexWrapper Class Reference . . . . .                      | 63 |
| 6.24.1 Detailed Description . . . . .                                  | 65 |
| 6.24.2 Member Function Documentation . . . . .                         | 65 |
| 6.24.2.1 text() . . . . .  | 65 |
| 6.25 core::Manager Class Reference . . . . .                           | 65 |
| 6.25.1 Detailed Description . . . . .                                  | 66 |
| 6.25.2 Member Function Documentation . . . . .                         | 66 |

---

|  |    |
|--|----|
| 6.25.2.1 run()   | 66 |
| 6.26 algebra::Matrix< T > Class Template Reference                   | 66 |
| 6.26.1 Detailed Description  | 69 |
| 6.26.2 Constructor & Destructor Documentation                        | 70 |
| 6.26.2.1 Matrix()  | 70 |
| 6.26.3 Member Function Documentation                                 | 70 |
| 6.26.3.1 at() [1/2]  | 70 |
| 6.26.3.2 at() [2/2]  | 70 |
| 6.26.3.3 operator*=( )   | 71 |
| 6.26.3.4 operator[]() [1/2]  | 71 |
| 6.26.3.5 operator[]() [2/2]  | 71 |
| 6.27 core::MinecraftCoordinates Struct Reference                     | 72 |
| 6.27.1 Detailed Description  | 72 |
| 6.28 core::MinecraftSavefileParser Class Reference                   | 73 |
| 6.28.1 Detailed Description  | 74 |
| 6.28.2 Member Function Documentation                                 | 74 |
| 6.28.2.1 parse()   | 74 |
| 6.29 core::MinecraftSavefileParser_mcSavefileParsers Class Reference | 74 |
| 6.29.1 Detailed Description  | 76 |
| 6.29.2 Member Function Documentation                                 | 76 |
| 6.29.2.1 parse()   | 76 |
| 6.30 core::Options Class Reference                                   | 76 |
| 6.30.1 Detailed Description  | 78 |
| 6.30.2 Member Function Documentation                                 | 79 |
| 6.30.2.1 filename()  | 79 |
| 6.30.2.2 help()  | 79 |
| 6.30.2.3 homology_to_compute()                                       | 79 |
| 6.30.2.4 latex()   | 79 |
| 6.30.2.5 x_bounds()  | 79 |
| 6.30.2.6 y_bounds()  | 79 |
| 6.30.2.7 z_bounds()  | 80 |
| 6.31 core::Polymorphic< T > Class Template Reference                 | 80 |
| 6.31.1 Detailed Description  | 81 |
| 6.32 algebra::RowEchelonFormResult< T > Struct Template Reference    | 82 |
| 6.32.1 Detailed Description  | 82 |
| 6.33 algebra::SkipCorrectnessCheckT Struct Reference                 | 83 |
| 6.33.1 Detailed Description  | 83 |
| 6.34 algebra::SmithFormResult< T > Struct Template Reference         | 83 |
| 6.34.1 Detailed Description  | 84 |
| 6.35 core::TextDrawable Class Reference                              | 84 |
| 6.35.1 Detailed Description  | 86 |
| 6.35.2 Member Function Documentation                                 | 86 |

---

|  |           |
|--|-----------|
| 6.35.2.1 <code>text()</code>   | 86        |
| 6.36 <code>algebra::ZModP&lt; P &gt;</code> Class Template Reference                     | 87        |
| 6.36.1 Detailed Description  | 89        |
| 6.36.2 Member Function Documentation   | 89        |
| 6.36.2.1 <code>euclidean_function()</code>   | 89        |
| 6.37 <code>algebra::ZModP&lt; 2 &gt;</code> Class Reference                              | 90        |
| 6.37.1 Detailed Description  | 92        |
| 6.37.2 Member Function Documentation   | 92        |
| 6.37.2.1 <code>euclidean_function()</code>   | 92        |
| <b>7 File Documentation</b>  | <b>93</b> |
| 7.1 <code>mc-homology/algebra/include/algebra/algebraic_concepts.h</code> File Reference | 93        |
| 7.1.1 Detailed Description   | 95        |
| 7.2 <code>algebraic_concepts.h</code>  | 95        |
| 7.3 <code>mc-homology/algebra/include/algebra/chain_complex.h</code> File Reference      | 95        |
| 7.3.1 Detailed Description   | 97        |
| 7.4 <code>chain_complex.h</code>   | 97        |
| 7.5 <code>mc-homology/algebra/include/algebra/integer.h</code> File Reference            | 99        |
| 7.5.1 Detailed Description   | 100       |
| 7.5.2 Function Documentation   | 100       |
| 7.5.2.1 <code>divide()</code>  | 100       |
| 7.5.2.2 <code>modulo()</code>  | 100       |
| 7.6 <code>integer.h</code>   | 101       |
| 7.7 <code>mc-homology/algebra/include/algebra/matrix.h</code> File Reference             | 102       |
| 7.7.1 Detailed Description   | 104       |
| 7.7.2 Function Documentation   | 104       |
| 7.7.2.1 <code>operator*()</code>   | 104       |
| 7.8 <code>matrix.h</code>  | 104       |
| 7.9 <code>mc-homology/algebra/include/algebra/matrix_algorithms.h</code> File Reference  | 108       |
| 7.9.1 Detailed Description   | 110       |
| 7.9.2 Function Documentation   | 110       |
| 7.9.2.1 <code>row_echelon_form() [1/2]</code>  | 110       |
| 7.9.2.2 <code>row_echelon_form() [2/2]</code>  | 111       |
| 7.9.2.3 <code>smith_form() [1/2]</code>  | 111       |
| 7.9.2.4 <code>smith_form() [2/2]</code>  | 111       |
| 7.10 <code>matrix_algorithms.h</code>  | 112       |
| 7.11 <code>mc-homology/algebra/include/algebra/modulo_fields.h</code> File Reference     | 113       |
| 7.11.1 Detailed Description  | 115       |
| 7.12 <code>modulo_fields.h</code>  | 115       |
| 7.13 <code>mc-homology/algebra/include/algebra/number_theory.h</code> File Reference     | 116       |
| 7.13.1 Detailed Description  | 119       |
| 7.13.2 Function Documentation  | 119       |

---

|   |     |
|---|-----|
| 7.13.2.1 divide()   | 119 |
| 7.13.2.2 extended_gcd()   | 119 |
| 7.13.2.3 inverse_mod()  | 120 |
| 7.13.2.4 is_prime()   | 120 |
| 7.13.2.5 modulo()   | 120 |
| 7.14 number_theory.h  | 121 |
| 7.15 mc-homology/algebra/include/algebra/z2_field.h File Reference                  | 122 |
| 7.15.1 Detailed Description   | 122 |
| 7.16 z2_field.h   | 123 |
| 7.17 mc-homology/complexes/include/complexes/compute_chain_complex.h File Reference | 124 |
| 7.17.1 Detailed Description   | 125 |
| 7.17.2 Function Documentation   | 125 |
| 7.17.2.1 compute_chain_complex()  | 125 |
| 7.18 compute_chain_complex.h  | 125 |
| 7.19 mc-homology/complexes/include/complexes/cubical_complex.h File Reference       | 126 |
| 7.19.1 Detailed Description   | 128 |
| 7.20 cubical_complex.h  | 128 |
| 7.21 mc-homology/complexes/include/complexes/utils.h File Reference                 | 129 |
| 7.21.1 Detailed Description   | 130 |
| 7.22 utils.h  | 130 |
| 7.23 mc-homology/core/include/core/algebra_homology.h File Reference                | 131 |
| 7.23.1 Detailed Description   | 132 |
| 7.24 algebra_homology.h   | 132 |
| 7.25 mc-homology/core/include/core/complex.h File Reference                         | 132 |
| 7.25.1 Detailed Description   | 133 |
| 7.26 complex.h  | 134 |
| 7.27 mc-homology/core/include/core/cubical_complex_3d.h File Reference              | 134 |
| 7.27.1 Detailed Description   | 135 |
| 7.28 cubical_complex_3d.h   | 135 |
| 7.29 mc-homology/core/include/core/homology.h File Reference                        | 135 |
| 7.29.1 Detailed Description   | 136 |
| 7.30 homology.h   | 137 |
| 7.31 mc-homology/core/include/core/homology_printing_strategy.h File Reference      | 137 |
| 7.31.1 Detailed Description   | 138 |
| 7.32 homology_printing_strategy.h   | 139 |
| 7.33 mc-homology/core/include/core/latex_wrapper.h File Reference                   | 139 |
| 7.33.1 Detailed Description   | 140 |
| 7.34 latex_wrapper.h  | 140 |
| 7.35 mc-homology/core/include/core/manager.h File Reference                         | 140 |
| 7.35.1 Detailed Description   | 141 |
| 7.36 manager.h  | 141 |
| 7.37 mc-homology/core/include/core/options.h File Reference                         | 142 |

|   |            |
|---|------------|
| 7.37.1 Detailed Description . . . . .                                       | 143        |
| 7.38 options.h . . . . .  | 143        |
| 7.39 mc-homology/core/include/core/parser.h File Reference . . . . .        | 144        |
| 7.39.1 Detailed Description . . . . .                                       | 145        |
| 7.40 parser.h . . . . .   | 145        |
| 7.41 mc-homology/core/include/core/polymorphic.h File Reference . . . . .   | 145        |
| 7.41.1 Detailed Description . . . . .                                       | 146        |
| 7.42 polymorphic.h . . . . .  | 146        |
| 7.43 mc-homology/core/include/core/text_drawable.h File Reference . . . . . | 148        |
| 7.43.1 Detailed Description . . . . .                                       | 149        |
| 7.44 text_drawable.h . . . . .  | 149        |
| <b>Index</b>  | <b>151</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 > . . . . .                      | 20 |
| core::Complex . . . . .                                   | 27 |
| core::CubicalComplex3D . . . . .                          | 32 |
| complexes::CubicalComplex . . . . .                       | 29 |
| complexes::CubicalSimplex . . . . .                       | 35 |
| algebra::DivResult< T > . . . . .                         | 38 |
| algebra::ExtendedGCDResult . . . . .                      | 39 |
| std::formatter  |    |
| std::formatter< algebra::Integer > . . . . .              | 40 |
| std::formatter< algebra::ZModP< P > > . . . . .           | 43 |
| std::formatter< complexes::BasicInterval > . . . . .      | 44 |
| std::formatter< complexes::CubicalSimplex > . . . . .     | 46 |
| std::formatter< algebra::Matrix< T > > . . . . .          | 41 |
| std::hash< complexes::BasicInterval > . . . . .           | 48 |
| std::hash< complexes::CubicalSimplex > . . . . .          | 49 |
| algebra::Homology< T > . . . . .                          | 49 |
| core::HomologyPrintingStrategy . . . . .                  | 58 |
| core::HomologyLatexPrint . . . . .                        | 55 |
| core::HomologyRawPrint . . . . .                          | 60 |
| algebra::Integer . . . . .                                | 61 |
| core::Manager . . . . .                                   | 65 |
| algebra::Matrix< T > . . . . .                            | 66 |
| core::MinecraftCoordinates . . . . .                      | 72 |
| core::MinecraftSavefileParser . . . . .                   | 73 |
| core::MinecraftSavefileParser_mcSavefileParsers . . . . . | 74 |
| core::Options . . . . .                                   | 76 |
| core::CommandlineOptions . . . . .                        | 22 |
| core::Polymorphic< T > . . . . .                          | 80 |
| algebra::RowEchelonFormResult< T > . . . . .              | 82 |
| algebra::SkipCorrectnessCheckT . . . . .                  | 83 |
| algebra::SmithFormResult< T > . . . . .                   | 83 |
| core::TextDrawable . . . . .                              | 84 |
| core::Homology . . . . .                                  | 51 |

|                            |    |
|----------------------------|----|
| core::AlgebraHomology< T > | 15 |
| core::LatexWrapper         | 63 |
| algebra::ZModP< P >        | 87 |
| algebra::ZModP< 2 >        | 90 |

# Chapter 3

## Class Index

### 3.1 Class List

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

|   |  |    |
|---|--|----|
| core::AlgebraHomology< T >                  | Concrete subclass of Homology, based on algebra library . . . . .    | 15 |
| complexes::BasicInterval                    | A struct representing an interval of length 0 or 1 . . . . .         | 19 |
| algebra::ChainComplex< T >                  | Class representing a chain complex with coefficients T . . . . .     | 20 |
| core::CommandlineOptions                    | A class for storing user's options from the commandline . . . . .    | 22 |
| core::Complex                               | Interface for complexes . . . . .                                    | 27 |
| complexes::CubicalComplex                   | Class representing a cubical complex . . . . .                       | 29 |
| core::CubicalComplex3D                      | A class representing a cubical complex in 3D space . . . . .         | 32 |
| complexes::CubicalSimplex                   | A class representing a single cubical simplex in a complex . . . . . | 35 |
| algebra::DivResult< T >                     | Result struct for the division operation . . . . .                   | 38 |
| algebra::ExtendedGCDResult                  | Result struct for the extended gcd algorithm . . . . .               | 39 |
| std::formatter< algebra::Integer >          | Formatter for Integer type . . . . .                                 | 40 |
| std::formatter< algebra::Matrix< T > >      | Formatter for the Matrix type . . . . .                              | 41 |
| std::formatter< algebra::ZModP< P > >       | Formatter for ZModP type . . . . .                                   | 43 |
| std::formatter< complexes::BasicInterval >  | Formatter for BasicInterval type . . . . .                           | 44 |
| std::formatter< complexes::CubicalSimplex > | Formatter for CubicalSimplex type . . . . .                          | 46 |
| std::hash< complexes::BasicInterval >       | Std::hash specialization for Interval . . . . .                      | 48 |
| std::hash< complexes::CubicalSimplex >      | Std::hash specialization for CubicalSimplex . . . . .                | 49 |
| algebra::Homology< T >                      | Homology of a chain complex . . . . .                                | 49 |

|  |    |
|--|----|
| core::Homology   |    |
| A homology interface   | 51 |
| core::HomologyLatexPrint   |    |
| Prints a homology group in Latex syntax  | 55 |
| core::HomologyPrintingStrategy   |    |
| A strategy for printing a homology group   | 58 |
| core::HomologyRawPrint   |    |
| A strategy for basic string representation of homology   | 60 |
| algebra::Integer   |    |
| Class of Integers  | 61 |
| core::LatexWrapper   |    |
| A decorator that wraps a text drawable object in a latex document  | 63 |
| core::Manager  |    |
| Main manager for the program   | 65 |
| algebra::Matrix< T >   |    |
| A Matrix class   | 66 |
| core::MinecraftCoordinates   |    |
| A struct containing coordinates in a Minecraft world   | 72 |
| core::MinecraftSavefileParser  |    |
| An interface for Minecraft savefile parser   | 73 |
| core::MinecraftSavefileParser_mcSavefileParsers  |    |
| A Minecraft savefile parser based on <a href="https://github.com/TCA166/mcSavefileParsers.git">https://github.com/TCA166/mcSavefileParsers.git</a>   | 74 |
| core::Options  |    |
| A class for storing user options   | 76 |
| core::Polymorphic< T >   |    |
| A polymorphic class wrapper with value semantics   | 80 |
| algebra::RowEchelonFormResult< T >   |    |
| Result struct for the row echelon algorithm  | 82 |
| algebra::SkipCorrectnessCheck  |    |
| 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 | 83 |
| algebra::SmithFormResult< T >  |    |
| Result struct for the smith algorithm  | 83 |
| core::TextDrawable   |    |
| Marks a class text drawable - it can be represented as a string  | 84 |
| algebra::ZModP< P >  |    |
| Field of integers modulo P   | 87 |
| algebra::ZModP< 2 >  |    |
| Template specialization for P == 2   | 90 |

# 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>              | 93  |
| A file defining multiple concepts for algebraic structures . . . . .                   |     |
| mc-homology/algebra/include/algebra/ <a href="#">chain_complex.h</a>                   | 95  |
| A file containing chain complex and homology implementations . . . . .                 |     |
| mc-homology/algebra/include/algebra/ <a href="#">integer.h</a>                         | 99  |
| A file containing Integer implementation satisfying algebraic concepts . . . . .       |     |
| mc-homology/algebra/include/algebra/ <a href="#">matrix.h</a>                          | 102 |
| A file containing matrix implementation . . . . .                                      |     |
| mc-homology/algebra/include/algebra/ <a href="#">matrix_algorithms.h</a>               | 108 |
| A file containing selected matrix algorithms . . . . .                                 |     |
| mc-homology/algebra/include/algebra/ <a href="#">modulo_fields.h</a>                   | 113 |
| A file containing implementation of fields of integers mod P . . . . .                 |     |
| mc-homology/algebra/include/algebra/ <a href="#">number_theory.h</a>                   | 116 |
| A file containing implementations of selected number theoretic algorithms . . . . .    |     |
| mc-homology/algebra/include/algebra/ <a href="#">z2_field.h</a>                        | 122 |
| A file containing optimized implementation of Z2 field . . . . .                       |     |
| mc-homology/complexes/include/complexes/ <a href="#">compute_chain_complex.h</a>       | 124 |
| A file containing algorithms for transforming complexes into chain complexes . . . . . |     |
| mc-homology/complexes/include/complexes/ <a href="#">cubical_complex.h</a>             | 126 |
| A file containing the CubicalComplex class . . . . .                                   |     |
| mc-homology/complexes/include/complexes/ <a href="#">utils.h</a>                       | 129 |
| Contains auxillary classes and functions . . . . .                                     |     |
| mc-homology/core/include/core/ <a href="#">algebra_homology.h</a>                      | 131 |
| File containing concrete classes deriving from Homology . . . . .                      |     |
| mc-homology/core/include/core/ <a href="#">complex.h</a>                               | 132 |
| A file containing interface for the Complex class . . . . .                            |     |
| mc-homology/core/include/core/ <a href="#">cubical_complex_3d.h</a>                    | 134 |
| A file containing a concrete class CubicalComplex3D . . . . .                          |     |
| mc-homology/core/include/core/ <a href="#">homology.h</a>                              | 135 |
| A file containing a Homology interface . . . . .                                       |     |
| mc-homology/core/include/core/ <a href="#">homology_printing_strategy.h</a>            | 137 |
| File containing strategies for printing homology . . . . .                             |     |
| mc-homology/core/include/core/ <a href="#">latex_wrapper.h</a>                         | 139 |
| Turns a TextDrawable into a simple latex document . . . . .                            |     |
| mc-homology/core/include/core/ <a href="#">manager.h</a>                               | 140 |
| A file containing main logic of the program . . . . .                                  |     |

|   |     |
|---|-----|
| mc-homology/core/include/core/ <a href="#">options.h</a>  | 142 |
| File containing an Options class for storing user options . . . . .   |     |
| mc-homology/core/include/core/ <a href="#">parser.h</a>   | 144 |
| A file containing a minecraft savefile parser . . . . .   |     |
| mc-homology/core/include/core/ <a href="#">polymorphic.h</a>  | 145 |
| A polymorphic type wrapper with value sematincs . . . . .   |     |
| mc-homology/core/include/core/ <a href="#">text_drawable.h</a>  | 148 |
| A file containing a TextDrawable interface, for classes that can be drawn on screen in the form of text . . . . . |     |

# 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  $\oplus$  ( $+$  lub  $-$ ) we have  $x \oplus y == y \oplus x$ .

### 5.2.3 Semantic requirements

For  $x$  and  $y$  and operation  $\oplus$

1.  $x \oplus y == y \oplus x$

## 5.3 algebra::CommutativeRing Concept Reference

An commutative ring concept.

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

### 5.3.1 Concept definition

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

### 5.3.2 Detailed Description

An commutative ring concept.

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

### 5.3.3 Sematic requirements

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

1.  $x * y == y * x$

## 5.4 algebra::EuclideanDomain Concept Reference

An euclidean domain concept.

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

### 5.4.1 Concept definition

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

### 5.4.2 Detailed Description

An euclidean domain concept.

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

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

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  $y \neq T::zero()$

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

## 5.5 algebra::Field Concept Reference

A field concept.

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

### 5.5.1 Concept definition

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

## 5.5.2 Detailed Description

A field concept.

`Field` concept declares that a type models a mathematical definition of a field, where addition is denoted by `+` and multiplication is denoted by `*`. Additionally, the 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 Semantic 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

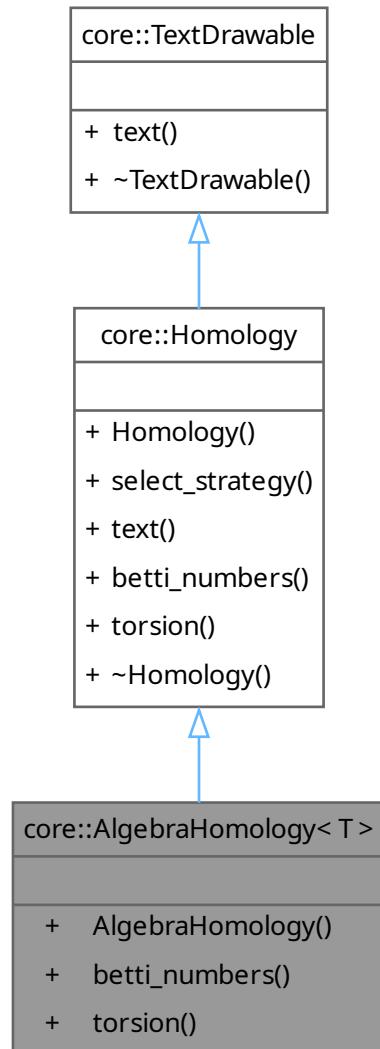
# Class Documentation

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

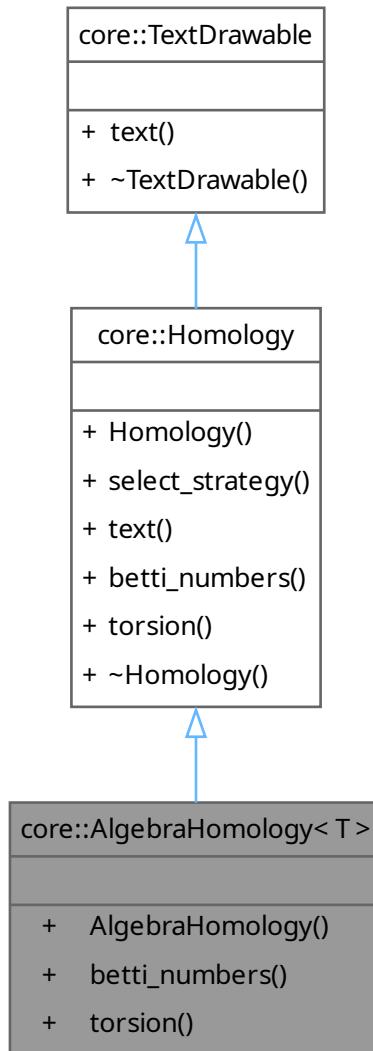
Concrete subclass of [Homology](#), based on algebra library.

```
#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.*

### 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:

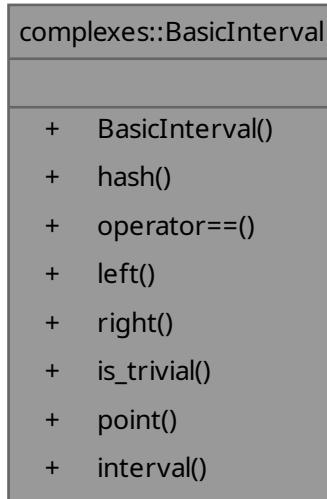
- mc-homology/core/include/core/[algebra\\_homology.h](#)

## 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:



### 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.*

## 6.2.1 Detailed Description

A struct representing an interval of length 0 or 1.

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

## 6.2.2 Member Function Documentation

### 6.2.2.1 `interval()`

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

Creates a interval of length one starting in `left`.

#### Parameters

|                   |                     |
|-------------------|---------------------|
| <code>left</code> | end of the interval |
|-------------------|---------------------|

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

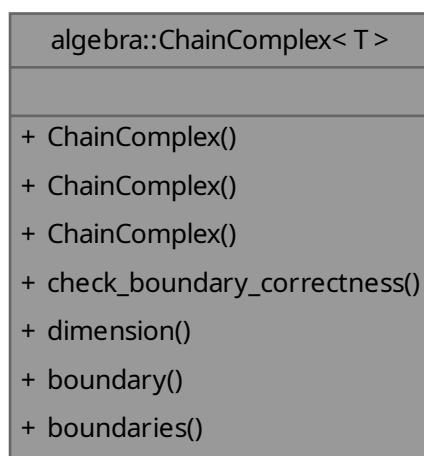
- mc-homology/complexes/include/complexes/[cubical\\_complex.h](#)
- mc-homology/complexes/src/cubical\_complex.cpp

## 6.3 algebra::ChainComplex< T > Class Template Reference

Class representing a chain complex with coefficients `T`.

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

Collaboration diagram for algebra::ChainComplex< `T` >:



## Public Member Functions

- template<std::ranges::sized\_range R>  
**constexpr ChainComplex** (R &&boundaries)  
*Constructs the chain complex from boundary matrices and checks, if they satisfy the chain complex condition.*
- template<std::ranges::sized\_range R>  
**constexpr ChainComplex** (SkipCorrectnessCheckT, R &&boundaries)  
*Constructs the chain complex from the boundary matrices without checking the chain complex condition.*
- **constexpr bool check\_boundary\_correctness () const**  
*Checks, if the boundaries satisfy the chain complex condition.*
- **constexpr std::size\_t dimension () const noexcept**  
*Return the dimension (the number of boundary matrices minus 1) of the chain complex.*
- **constexpr Matrix< T > const & boundary (std::size\_t dim) const**  
*Return the boundary operator at dimension dim.*
- **constexpr std::vector< Matrix< T > > const & boundaries () const noexcept**  
*Return the vector of boundary operators.*

### 6.3.1 Detailed Description

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

Class representing a chain complex with coefficients T.

Chain complex is a free module over T with submodules (C\_d, ..., C\_2, C\_1, C\_0, C\_{-1}) and linear operators B\_n : C\_n → C\_{(n-1)} satisfying B\_n\*B\_{(n+1)} == 0 (the chain complex condidition).

#### Template Parameters

|   |                           |
|---|---------------------------|
| T | Class of the coefficients |
|---|---------------------------|

### 6.3.2 Constructor & Destructor Documentation

#### 6.3.2.1 ChainComplex() [1/2]

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

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

#### Parameters

|            |   |
|------------|---|
| boundaries | The range containing the boundary opeartors |
|------------|---|

### 6.3.2.2 ChainComplex() [2/2]

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

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

Use with caution!

#### Parameters

|                   |   |
|-------------------|---|
| <i>boundaries</i> | The range containing the boundary opeartors |
|-------------------|---|

## 6.3.3 Member Function Documentation

### 6.3.3.1 boundary()

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

Return the boundary operator at dimension *dim*.

#### Parameters

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

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

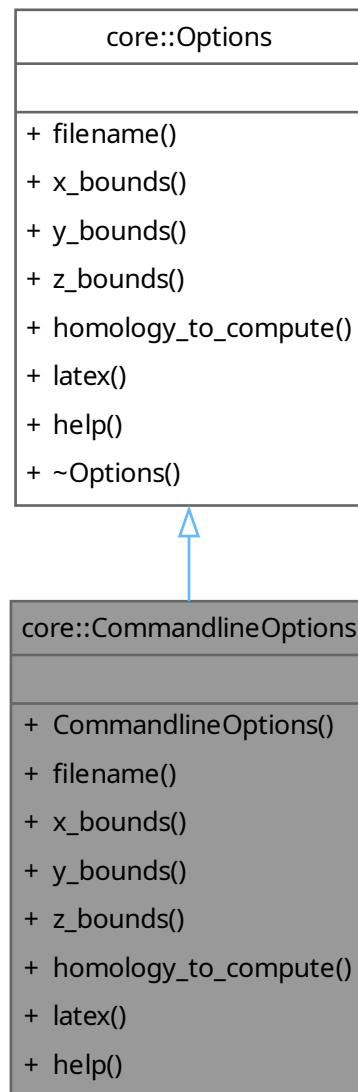
- mc-homology/algebra/include/algebra/[chain\\_complex.h](#)

## 6.4 core::CommandlineOptions Class Reference

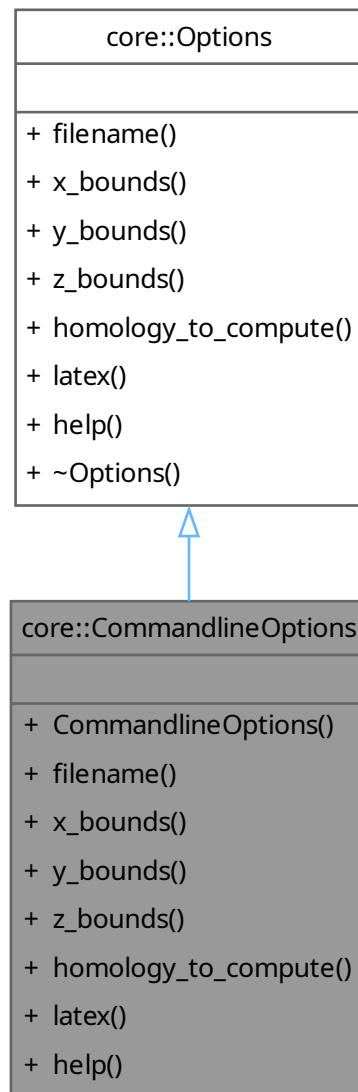
A class for storing user's options from the commandline.

```
#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 y axis.*
  - `HomologyChoice homology_to_compute () const override`  
*Type of homology to compute.*
  - `bool latex () const override`  
*Whether to print latex output.*
  - `bool help () const override`  
*Whether the user requested help.*

## Public Member Functions inherited from `core::Options`

- `virtual ~Options ()`  
*virtual destructor*

### 6.4.1 Detailed Description

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

### 6.4.2 Member Function Documentation

#### 6.4.2.1 `filename()`

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

Path to the region directory.

Implements `core::Options`.

#### 6.4.2.2 `help()`

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

Whether the user requested help.

Implements `core::Options`.

#### 6.4.2.3 `homology_to_compute()`

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

Type of homology to compute.

Implements `core::Options`.

#### 6.4.2.4 `latex()`

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

Whether to print latex output.

Implements [core::Options](#).

#### 6.4.2.5 `x_bounds()`

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

Bounds on x axis.

Implements [core::Options](#).

#### 6.4.2.6 `y_bounds()`

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

Bounds on y axis.

Implements [core::Options](#).

#### 6.4.2.7 `z_bounds()`

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

Bounds on z axis.

Implements [core::Options](#).

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

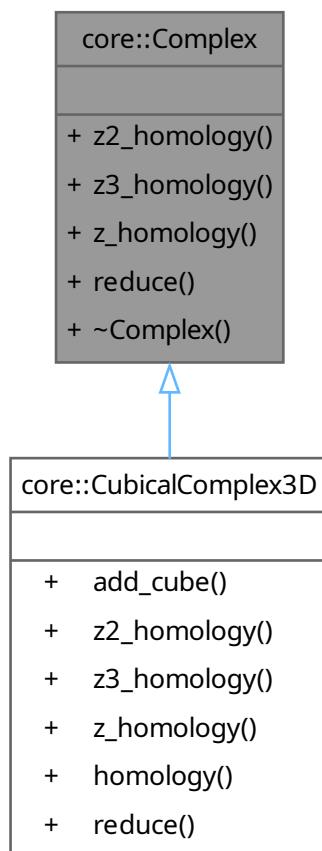
- mc-homology/core/include/core/[options.h](#)
- mc-homology/core/src/options.cpp

## 6.5 core::Complex Class Reference

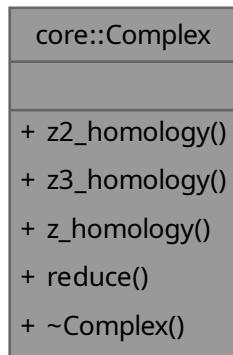
Interface for complexes.

```
#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:

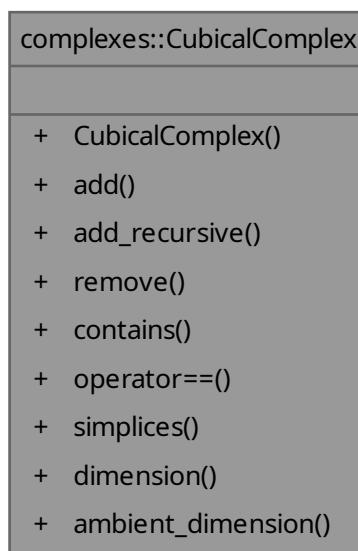
- mc-homology/core/include/core/[complex.h](#)
- mc-homology/core/src/complex.cpp

## 6.6 complexes::CubicalComplex Class Reference

Class representing a cubical complex.

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

Collaboration diagram for complexes::CubicalComplex:



## Public Member Functions

- **CubicalComplex ()**  
*Default constructor for the CubicalComplex.*
- **bool add (CubicalSimplex simplex)**  
*Adds a cubical simplex to the complex.*
- **void add\_recursive (CubicalSimplex simplex)**  
*Adds a cubical simplex to the complex together with its boundary.*
- **bool remove (CubicalSimplex const &simplex)**  
*Removes a cubical simplex from the complex.*
- **bool contains (CubicalSimplex const &simplex) const**  
*Checks, if a cubical complex contains simplex.*
- **bool operator== (CubicalComplex const &) const**  
*Equality comparison operator.*
- **std::vector< std::unordered\_set< CubicalSimplex > > const & simplices () const**  
*Grants access to the simplexes of the complex.*
- **std::size\_t dimension () const**  
*Returns the dimension of the simplex.*
- **std::size\_t ambient\_dimension () const**  
*Returns the ambient dimension of the simplex.*

### 6.6.1 Detailed Description

Class representing a cubical complex.

A class representing a topological space constructed from cubical simplexes.

### 6.6.2 Constructor & Destructor Documentation

#### 6.6.2.1 CubicalComplex()

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

Default constructor for the CubicalComplex.

Creates an empty complex

### 6.6.3 Member Function Documentation

#### 6.6.3.1 add()

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

Adds a cubical simplex to the complex.

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

#### Parameters

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

#### Returns

true, if simplex has been successfully added, false otherwise

### 6.6.3.2 add\_recursive()

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

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

#### Parameters

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

### 6.6.3.3 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

### 6.6.3.4 remove()

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

Removes a cubical simplex from the complex.

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

#### Parameters

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

#### Returns

true, if removal was successfull, false otherwise

### 6.6.3.5 `simplices()`

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

Grants access to the simplexes of the complex.

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

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

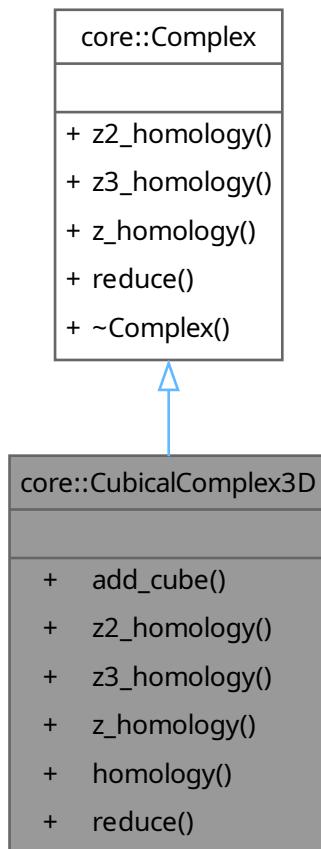
- mc-homology/complexes/include/complexes/cubical\_complex.h
- mc-homology/complexes/src/cubical\_complex.cpp

## 6.7 core::CubicalComplex3D Class Reference

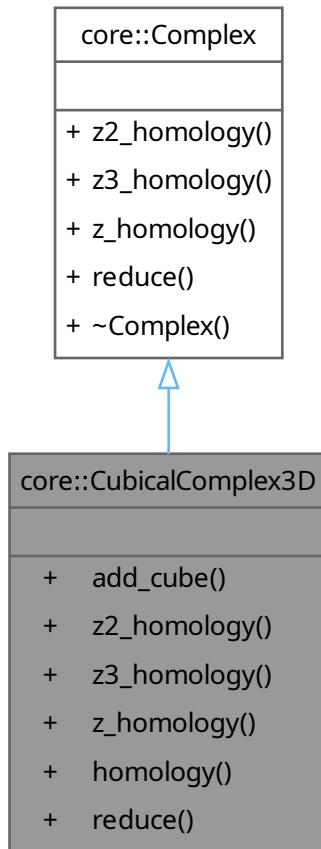
A class representing a cubical complex in 3D space.

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

Inheritance diagram for core::CubicalComplex3D:



Collaboration diagram for core::CubicalComplex3D:



## Public Member Functions

- `void add_cube (int x, int y, int z)`  
*Adds a cube to the complex.*
- `std::unique_ptr< Homology > z2_homology () const override`  
*Computes Z2 homology of the complex.*
- `std::unique_ptr< Homology > z3_homology () const override`  
*Computes Z3 homology of the complex.*
- `std::unique_ptr< Homology > z_homology () const override`  
*Computes Z homology of the complex.*
- `template<class T> std::unique_ptr< AlgebraHomology< T > > homology () const`  
*Computes homology of the complex for coefficients of type T.*
- `void reduce () override`  
*Decreases the complex's size without changing its homology.*

## Public Member Functions inherited from [core::Complex](#)

- virtual ~[Complex](#) ()

*Virtual destructor.*

### 6.7.1 Detailed Description

A class representing a cubical complex in 3D space.

### 6.7.2 Member Function Documentation

#### 6.7.2.1 [reduce\(\)](#)

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

Decreases the complex's size without changing its homology.

Implements [core::Complex](#).

#### 6.7.2.2 [z2\\_homology\(\)](#)

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

Computes Z2 homology of the complex.

Implements [core::Complex](#).

#### 6.7.2.3 [z3\\_homology\(\)](#)

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

Computes Z3 homology of the complex.

Implements [core::Complex](#).

#### 6.7.2.4 [z\\_homology\(\)](#)

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

Computes Z homology of the complex.

Implements [core::Complex](#).

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

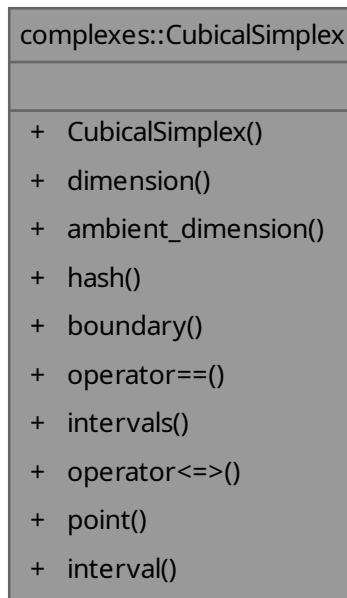
- mc-homology/core/include/core/[cubical\\_complex\\_3d.h](#)
- mc-homology/core/src/cubical\_complex\_3d.cpp

## 6.8 complexes::CubicalSimplex Class Reference

A class representing a single cubical simplex in a complex.

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

Collaboration diagram for complexes::CubicalSimplex:



### Public Member Functions

- [CubicalSimplex](#) (std::vector< [BasicInterval](#) > intervals)  
*Constructs a simplex from a vector of intervals.*
- std::size\_t [dimension](#) () const  
*Returns the dimension of the simplex.*
- std::size\_t [ambient\\_dimension](#) () const  
*Returns the ambient dimension of the simplex.*
- std::size\_t [hash](#) () const  
*Computes a hash of the simplex.*
- std::vector< [CubicalSimplex](#) > [boundary](#) () const  
*Returns the boundary of a simplex.*
- bool [operator==](#) ([CubicalSimplex](#) const &) const  
*C.compares two simplices.*
- std::vector< [BasicInterval](#) > const & [intervals](#) () const  
*Returns the underlying intervals.*
- std::strong\_ordering [operator<=>](#) ([CubicalSimplex](#) const &simplex) const  
*Comparison operator for simplices.*

## Static Public Member Functions

- static [CubicalSimplex point](#) (int p)  
*Creates a cubical simplex out of a single 1d point.*
- static [CubicalSimplex interval](#) (int left)  
*Creates a cubical simplex out of a single 1d interval.*

## Friends

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

## 6.8.1 Detailed Description

A class representing a single cubical simplex in a complex.

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

## 6.8.2 Constructor & Destructor Documentation

### 6.8.2.1 CubicalSimplex()

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

Constructs a simplex from a vector of intervals.

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

#### Parameters

|                           |                                |
|---------------------------|--------------------------------|
| <a href="#">Intervals</a> | A nonempty vector of intervals |
|---------------------------|--------------------------------|

## 6.8.3 Member Function Documentation

### 6.8.3.1 boundary()

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

Returns the boundary of a simplex.

The returned boundary is decresing in the sense of operator $\leq\geq$ .

### 6.8.3.2 interval()

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

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

#### Parameters

|             |                            |
|-------------|----------------------------|
| <i>left</i> | Left bound of the interval |
|-------------|----------------------------|

### 6.8.3.3 operator<=>()

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

Comparison operator for simplices.

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

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

for any  $a < b$

### 6.8.3.4 point()

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

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

#### Parameters

|          |           |
|----------|-----------|
| <i>p</i> | The point |
|----------|-----------|

## 6.8.4 Friends And Related Symbol Documentation

### 6.8.4.1 product

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

Creates a cubical simplex by joining together two simplices.

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

#### Parameters

|           |               |
|-----------|---------------|
| <i>s1</i> | First simplex |
|-----------|---------------|

|    |                |
|----|----------------|
| s2 | Second simplex |
|----|----------------|

**Returns**

The result of concatenation

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

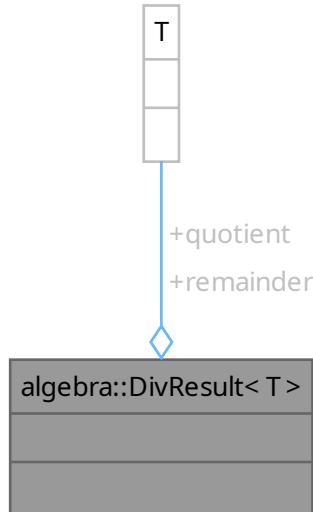
- mc-homology/complexes/include/complexes/[cubical\\_complex.h](#)
- mc-homology/complexes/src/cubical\_complex.cpp

## 6.9 algebra::DivResult< T > Struct Template Reference

Result struct for the division operation.

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

Collaboration diagram for algebra::DivResult< T >:



### Public Attributes

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

### 6.9.1 Detailed Description

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

Result struct for the division operation.

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

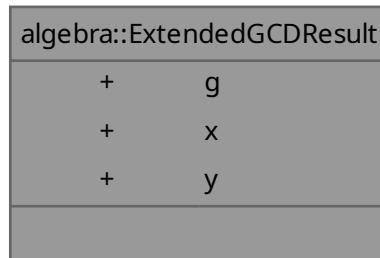
- mc-homology/algebra/include/algebra/number\_theory.h

## 6.10 algebra::ExtendedGCDResult Struct Reference

Result struct for the extended gcd algorithm.

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

Collaboration diagram for algebra::ExtendedGCDResult:



### Public Attributes

- int **g** = 0  
*Greatest common divisor.*
- int **x** = 0  
*First coefficient from the extended algorithm.*
- int **y** = 0  
*Second coefficient from the extended algorithm.*

### 6.10.1 Detailed Description

Result struct for the extended gcd algorithm.

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

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

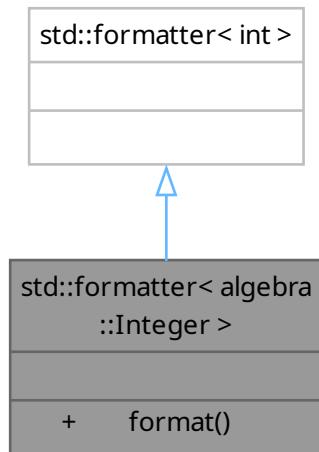
- mc-homology/algebra/include/algebra/number\_theory.h

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

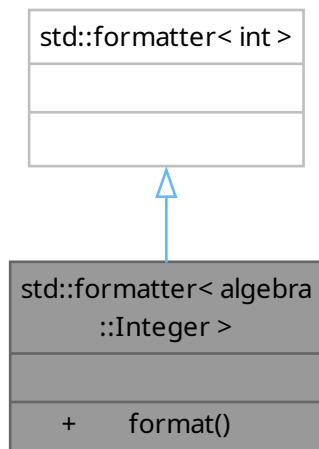
Formatter for Integer type.

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

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



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



## Public Member Functions

- template<class FmtContext>  
FmtContext::iterator **format** (algebra::Integer k, FmtContext &ctx) const  
*Formats an integer.*

### 6.11.1 Detailed Description

Formatter for Integer type.

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

### 6.11.2 Member Function Documentation

#### 6.11.2.1 format()

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

Formats an integer.

Formats an integer using a formatting syntax for `int`.

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

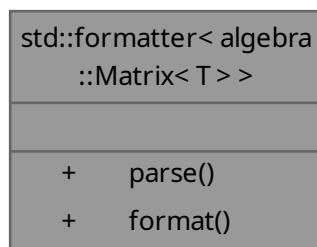
- [mc-homology/algebra/include/algebra/integer.h](#)

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

Formatter for the `Matrix` type.

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

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



## Public Member Functions

- template<class ParseContext>  
constexpr ParseContext::iterator **parse** (ParseContext &ctx)  
*Parses a context string.*
- template<class FmtContext>  
FmtContext::iterator **format** ([algebra::Matrix< T >](#) const &matrix, FmtContext &ctx) const  
*Formats a matrix.*

### 6.12.1 Detailed Description

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

Formatter for the `Matrix` type.

Implementation of the formatter for the `Matrix` type.

### 6.12.2 Format syntax

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

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

### 6.12.3 Example

For `Matrix<Integer> matrix`

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

### 6.12.4 Member Function Documentation

#### 6.12.4.1 format()

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

Formats a matrix.

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

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

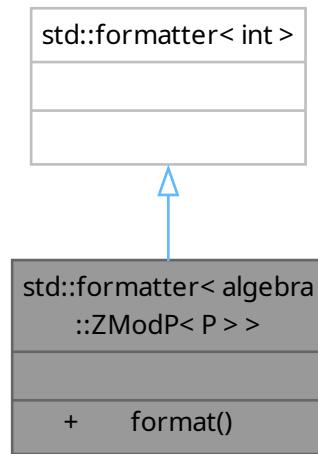
- `mc-homology/algebra/include/algebra/matrix.h`

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

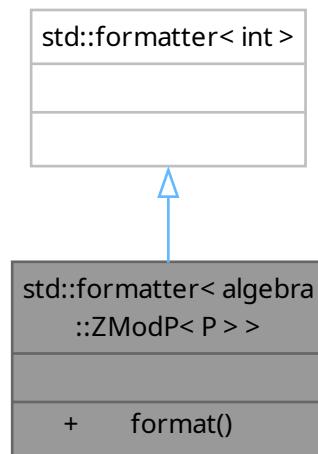
Formatter for ZModP type.

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

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



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



## Public Member Functions

- template<class FmtContext>  
FmtContext::iterator **format** ([algebra::ZModP< P >](#) x, FmtContext &ctx) const  
*Formats an integer mod P.*

### 6.13.1 Detailed Description

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

Formatter for ZModP type.

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

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

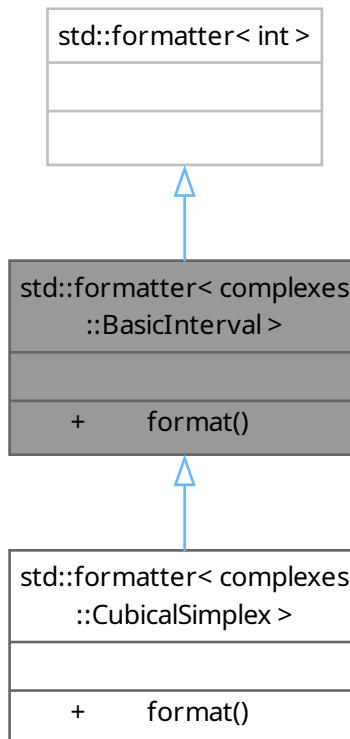
- [mc-homology/algebra/include/algebra/modulo\\_fields.h](#)

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

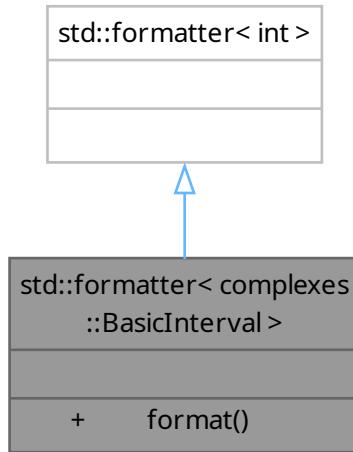
Formatter for BasicInterval type.

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

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



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



## Public Member Functions

- template<class FmtContext>  
FmtContext::iterator **format** (complexes::BasicInterval i, FmtContext &ctx) const  
*Formats an integer.*

### 6.14.1 Detailed Description

Formatter for BasicInterval type.

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

### 6.14.2 Member Function Documentation

#### 6.14.2.1 format()

```

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

Formats an integer.

Formats an interval using a formatting syntax for int.

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

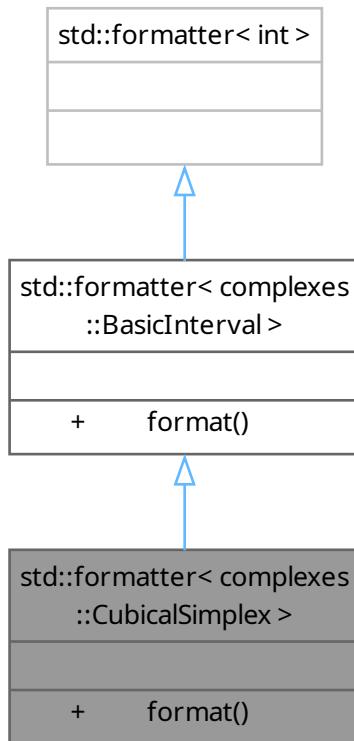
- mc-homology/complexes/include/complexes/cubical\_complex.h

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

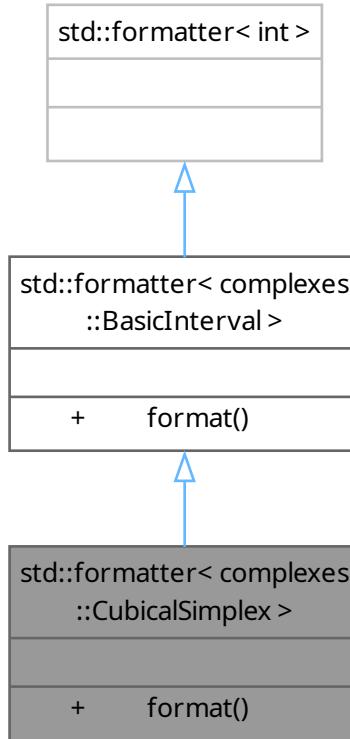
Formatter for CubicalSimplex type.

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

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



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.*

### 6.15.1 Detailed Description

Formatter for CubicalSimplex type.

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

## 6.15.2 Member Function Documentation

### 6.15.2.1 format()

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

Formats a cubical simplex.

Formats an integer using a formatting syntax for an interval.

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

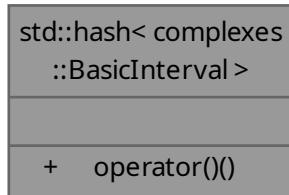
- mc-homology/complexes/include/complexes/cubical\_complex.h

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

std::hash specialization for Interval

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

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



### Public Member Functions

- std::size\_t **operator()** ([complexes::BasicInterval](#) const &i) const  
*Returns a hash of an interval.*

### 6.16.1 Detailed Description

std::hash specialization for Interval

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

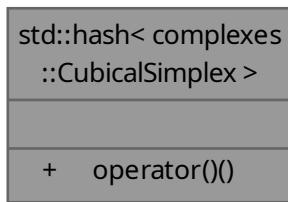
- mc-homology/complexes/include/complexes/cubical\_complex.h
- mc-homology/complexes/src/cubical\_complex.cpp

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

std::hash specialization for CubicalSimplex

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

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



### Public Member Functions

- std::size\_t **operator()** ([complexes::CubicalSimplex const &s](#)) const  
*Returns a hash of a cubical simplex.*

#### 6.17.1 Detailed Description

std::hash specialization for CubicalSimplex

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

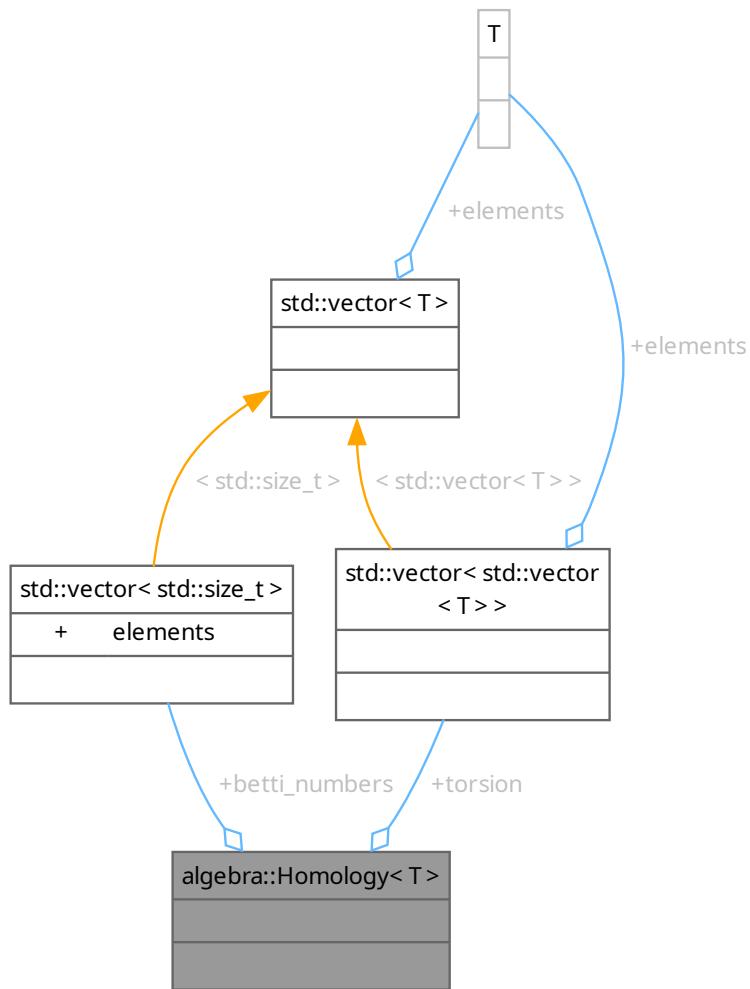
- mc-homology/complexes/include/complexes/[cubical\\_complex.h](#)
- mc-homology/complexes/src/cubical\_complex.cpp

## 6.18 algebra::Homology< T > Struct Template Reference

[Homology](#) of a chain complex.

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

Collaboration diagram for algebra::Homology< T >:



### Public Attributes

- std::vector< std::size\_t > **betti\_numbers** {}  
*Betti numbers of a chain complex.*
- std::vector< std::vector< T > > **torsion** {}  
*Torsion of a chain complex.*

### 6.18.1 Detailed Description

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

[Homology](#) of a chain complex.

A struct containing homology information of a chain complex - its betti numbers and torsion group. The data is stored increasingly in dimension: dim(H\_0) = betti\_numbers[0], dim(H\_1) = betti\_numbers[1] itd.

## 6.18.2 Member Data Documentation

### 6.18.2.1 torsion

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

Torsion of a chain complex.

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

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

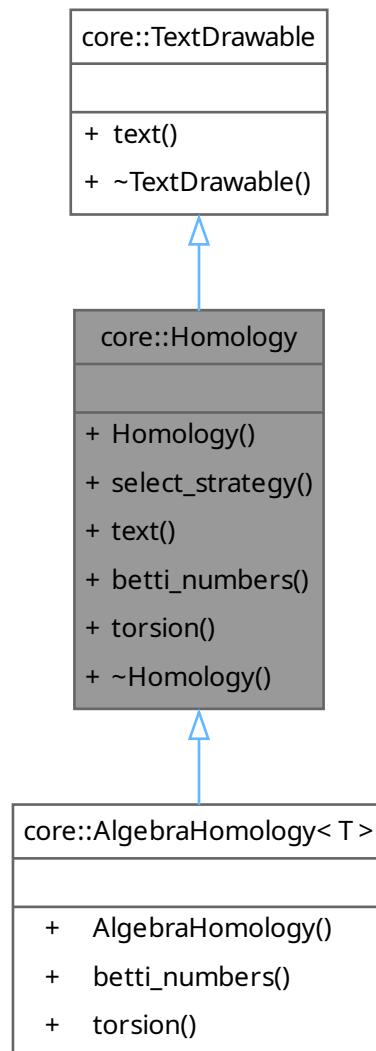
- mc-homology/algebra/include/algebra/[chain\\_complex.h](#)

## 6.19 core::Homology Class Reference

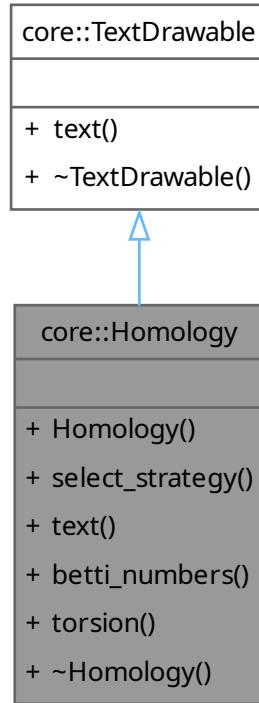
A homology interface.

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

Inheritance diagram for core::Homology:



Collaboration diagram for core::Homology:



## Public Member Functions

- [Homology](#) (std::unique\_ptr< HomologyPrintingStrategy > printing\_strategy=std::make\_unique< HomologyRawPrint >())
 

*Constructs a new homology instance.*
- void [select\\_strategy](#) (std::unique\_ptr< HomologyPrintingStrategy > printing\_strategy)
 

*Select a new printing\_strategy.*
- std::string [text](#) () const override
 

*Outputs the text form of the homology using given strategy.*
- virtual std::vector< std::size\_t > [betti\\_numbers](#) () const =0
 

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

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

## Public Member Functions inherited from [core::TextDrawable](#)

- virtual ~[TextDrawable](#) ()
 

*Virtual destructor.*

### 6.19.1 Detailed Description

A homology interface.

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

### 6.19.2 Constructor & Destructor Documentation

#### 6.19.2.1 Homology()

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

Constructs a new homology instance.

Constructs a new homology object with a specified printing strategy

#### Parameters

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

### 6.19.3 Member Function Documentation

#### 6.19.3.1 betti\_numbers()

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

Returns text representations of stored betti numbers.

Implemented in [core::AlgebraHomology< T >](#).

#### 6.19.3.2 select\_strategy()

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

Select a new printing\_strategy.

#### Parameters

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

#### 6.19.3.3 text()

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

Outputs the test form of the homology using given strategy.

Implements [core::TextDrawable](#).

### 6.19.3.4 torsion()

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

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

Implemented in [core::AlgebraHomology< T >](#).

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

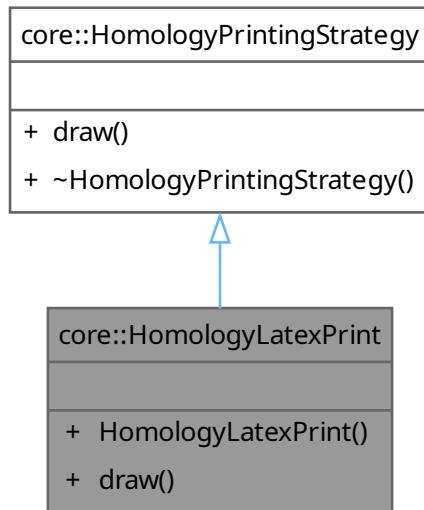
- mc-homology/core/include/core/[homology.h](#)
- mc-homology/core/src/homology.cpp

## 6.20 core::HomologyLatexPrint Class Reference

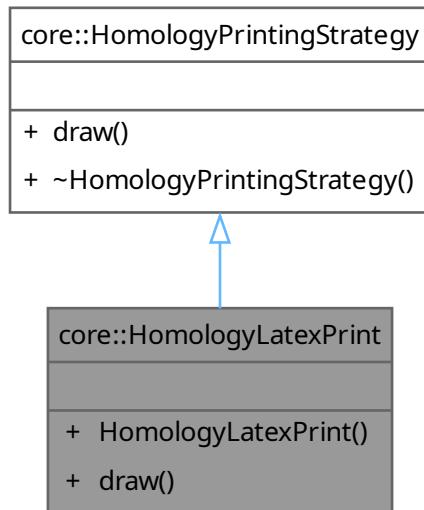
Prints a homology group in Latex syntax.

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

Inheritance diagram for core::HomologyLatexPrint:



Collaboration diagram for core::HomologyLatexPrint:



### Public Member Functions

- [HomologyLatexPrint](#) (std::string ring\_name, std::string homology\_name="H")  
*Constructor for the strategy.*
- std::string [draw](#) ([Homology](#) const &homology) const override  
*Returns the homology description in latex syntax.*

### Public Member Functions inherited from [core::HomologyPrintingStrategy](#)

- virtual ~[HomologyPrintingStrategy](#) ()  
*A virtual destructor.*

#### 6.20.1 Detailed Description

Prints a homology group in Latex syntax.

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

## 6.20.2 Constructor & Destructor Documentation

### 6.20.2.1 HomologyLatexPrint()

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

Constructor for the strategy.

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

#### Parameters

|                  |                               |
|------------------|-------------------------------|
| <i>ring_name</i> | Name for the coefficient ring |
|------------------|-------------------------------|

|                      |                             |
|----------------------|-----------------------------|
| <i>homology_name</i> | Name for the homology group |
|----------------------|-----------------------------|

### 6.20.3 Member Function Documentation

#### 6.20.3.1 draw()

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

Returns the homology description in latex syntax.

Implements [core::HomologyPrintingStrategy](#).

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

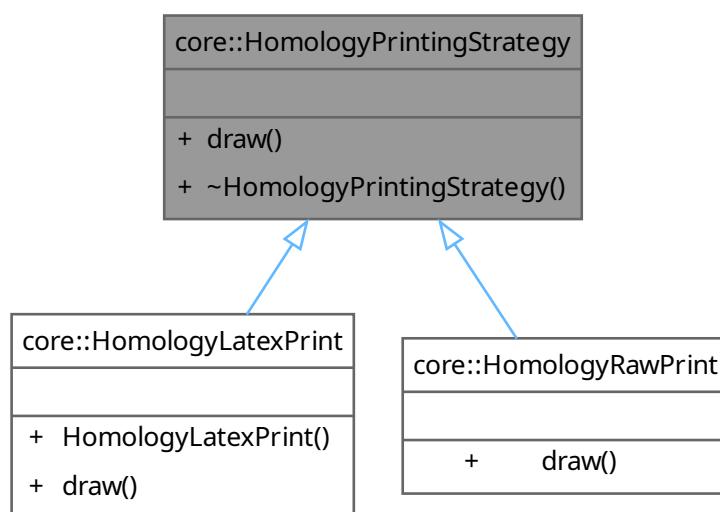
- mc-homology/core/include/core/[homology\\_printing\\_strategy.h](#)
- mc-homology/core/src/homology\_printing\_strategy.cpp

## 6.21 core::HomologyPrintingStrategy Class Reference

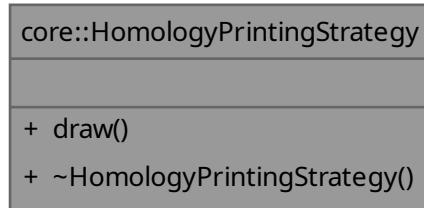
A strategy for printing a homology group.

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

Inheritance diagram for core::HomologyPrintingStrategy:



Collaboration diagram for core::HomologyPrintingStrategy:



## Public Member Functions

- virtual std::string [draw \(Homology const &homology\) const =0](#)  
*Returns the text representation of the homology group.*
- virtual [~HomologyPrintingStrategy \(\)](#)  
*A virtual destructor.*

### 6.21.1 Detailed Description

A strategy for printing a homology group.

### 6.21.2 Member Function Documentation

#### 6.21.2.1 [draw\(\)](#)

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

Returns the text representation of the homology group.

Implemented in [core::HomologyLatexPrint](#), and [core::HomologyRawPrint](#).

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

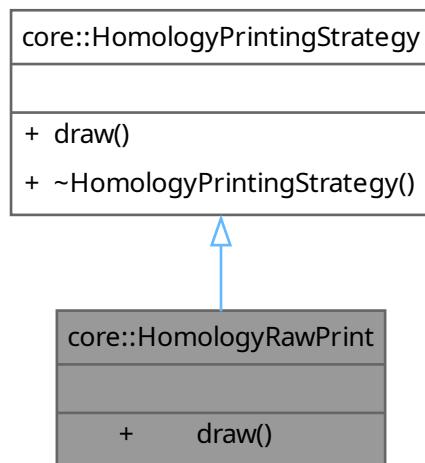
- mc-homology/core/include/core/[homology\\_printing\\_strategy.h](#)
- mc-homology/core/src/homology\_printing\_strategy.cpp

## 6.22 core::HomologyRawPrint Class Reference

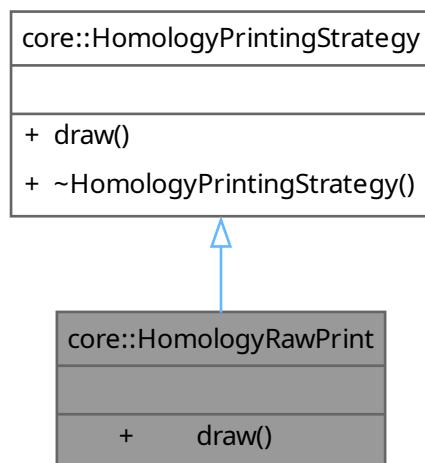
A strategy for basic string representation of homology.

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

Inheritance diagram for core::HomologyRawPrint:



Collaboration diagram for core::HomologyRawPrint:



## Public Member Functions

- std::string [draw \(Homology const &homology\) const override](#)  
*Prints betti numbers and torsion group.*

## Public Member Functions inherited from [core::HomologyPrintingStrategy](#)

- virtual ~[HomologyPrintingStrategy \(\)](#)  
*A virtual destructor.*

### 6.22.1 Detailed Description

A strategy for basic string representation of homology.

### 6.22.2 Member Function Documentation

#### 6.22.2.1 draw()

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

Prints betti numbers and torsion group.

Implements [core::HomologyPrintingStrategy](#).

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

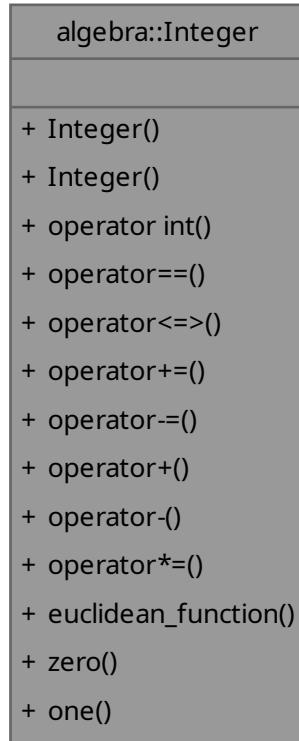
- mc-homology/core/include/core/[homology\\_printing\\_strategy.h](#)
- mc-homology/core/src/homology\_printing\_strategy.cpp

## 6.23 algebra::Integer Class Reference

Class of Integers.

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

Collaboration diagram for algebra::Integer:



## Public Member Functions

- `constexpr Integer ()=default`  
*Returns 0.*
- `constexpr Integer (int k) noexcept`  
*Returns k.*
- `constexpr operator int () noexcept`  
*Returns the underlying integer.*
- `constexpr bool operator== (Integer const &) const =default`  
*Compares two integers for equality.*
- `constexpr std::strong_ordering operator<=> (Integer const &) const =default`  
*Compares two integers and returns their ordering.*
- `constexpr Integer & operator+= (Integer rhs) noexcept`  
*Adds rhs to itself.*
- `constexpr Integer & operator-= (Integer rhs) noexcept`  
*Subtracts rhs from itself.*
- `constexpr Integer operator+ () const noexcept`  
*Returns a copy of self.*
- `constexpr Integer operator- () const noexcept`  
*Returns a negation of self.*

- constexpr `Integer & operator*=(Integer rhs)` noexcept  
*Multiples itself by rhs.*
- constexpr int `euclidean_function()` const noexcept  
*Euclidean function for integers.*

### Static Public Member Functions

- static constexpr `Integer zero()` noexcept  
*Returns 0.*
- static constexpr `Integer one()` noexcept  
*Returns 1.*

### 6.23.1 Detailed Description

Class of Integers.

A class implementing integers, while additionally satisfying `EuclideanDomain` constraint

### 6.23.2 Member Function Documentation

#### 6.23.2.1 `euclidean_function()`

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

Euclidean function for integers.

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

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

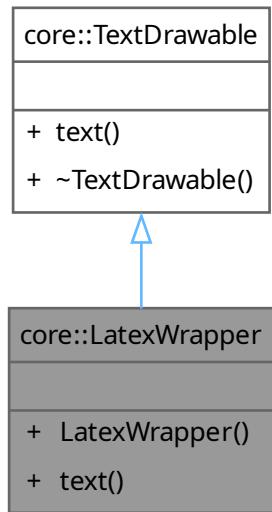
- mc-homology/algebra/include/algebra/`integer.h`

## 6.24 core::LatexWrapper Class Reference

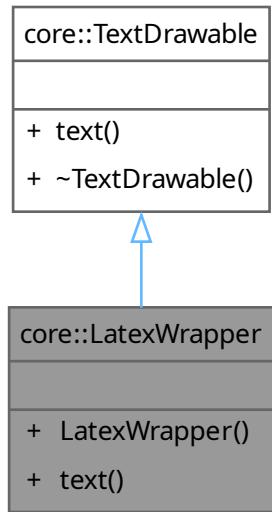
A decorator that wraps a text drawable object in a latex document.

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

Inheritance diagram for core::LatexWrapper:



Collaboration diagram for core::LatexWrapper:



## Public Member Functions

- **LatexWrapper** (std::unique\_ptr<[TextDrawable](#)> inner, std::string documentclass="article")

*Constructs the decorator with the object to wrap.*

- std::string **text () const override**  
*Returns the text representation of the object.*

## Public Member Functions inherited from [core::TextDrawable](#)

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

### 6.24.1 Detailed Description

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

### 6.24.2 Member Function Documentation

#### 6.24.2.1 **text()**

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

Returns the text representation of the object.

Implements [core::TextDrawable](#).

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

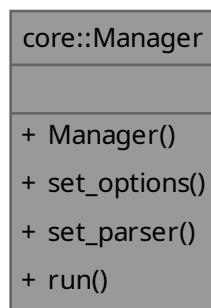
- mc-homology/core/include/core/[latex\\_wrapper.h](#)
- mc-homology/core/src/latex\_wrapper.cpp

## 6.25 core::Manager Class Reference

Main manager for the program.

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

Collaboration diagram for core::Manager:



## Public Member Functions

- **Manager** (std::unique\_ptr< Options > options, std::unique\_ptr< MinecraftSavefileParser > parser)  
*Construct a new manager with selected parser and options.*
- void **set\_options** (std::unique\_ptr< Options > options)  
*Select options.*
- void **set\_parser** (std::unique\_ptr< MinecraftSavefileParser > parser)  
*Select parser.*
- int **run** ()  
*Start the program.*

### 6.25.1 Detailed Description

Main manager for the program.

### 6.25.2 Member Function Documentation

#### 6.25.2.1 run()

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

Start the program.

Returns

program return status

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

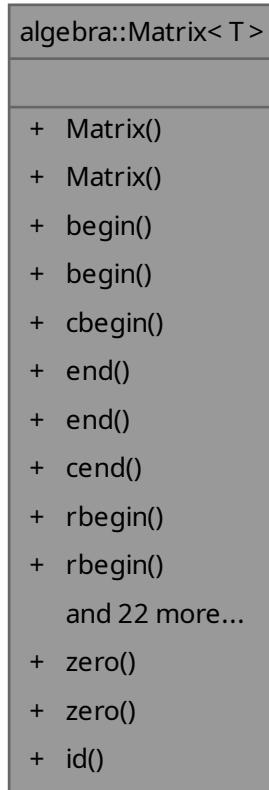
- mc-homology/core/include/core/manager.h
- mc-homology/core/src/manager.cpp

## 6.26 algebra::Matrix< T > Class Template Reference

A [Matrix](#) class.

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

Collaboration diagram for algebra::Matrix< T >:



## Public Types

- using **storage\_type** = `std::vector<T>`  
*Underlying storage type.*
- using **value\_type** = `std::vector<T>::value_type`  
*Type of the stored values.*
- using **allocator\_type** = `std::vector<T>::allocator_type`  
*Type of the used allocator.*
- using **size\_type** = `std::vector<T>::size_type`  
*Size type used by the underlying storage.*
- using **difference\_type** = `std::vector<T>::difference_type`  
*Difference type used by the underlying storage.*
- using **reference** = `std::vector<T>::reference`  
*Reference type to the stored values.*
- using **const\_reference** = `std::vector<T>::const_reference`  
*Const reference type to the stored values.*
- using **pointer** = `std::vector<T>::pointer`  
*Pointer type to the stored values.*
- using **const\_pointer** = `std::vector<T>::const_pointer`

- *Const pointer type to the stored values.*
- using **iterator** = std::vector<T>::iterator  
*Iterator over the underlying storage.*
- using **const\_iterator** = std::vector<T>::const\_iterator  
*Const iterator over the underlying storage.*
- using **reverse\_iterator** = std::vector<T>::reverse\_iterator  
*Reverse iterator over the underlying storage.*
- using **const\_reverse\_iterator** = std::vector<T>::const\_reverse\_iterator  
*Const reverse iterator over the underlying storage.*

## Public Member Functions

- template<std::ranges::sized\_range R>  
**constexpr Matrix** (R &&**data**, **size\_type nrows**, **size\_type ncols**)  
*Construct a matrix from a range.*
- **constexpr iterator begin** () noexcept  
*Iterator over the coefficients.*
- **constexpr const\_iterator begin** () const noexcept  
*Iterator over the coefficients.*
- **constexpr const\_iterator cbegin** () const noexcept  
*Const iterator over the coefficients.*
- **constexpr iterator end** () noexcept  
*Sentinel for the coefficients iterator.*
- **constexpr const\_iterator end** () const noexcept  
*Sentinel for the coefficients iterator.*
- **constexpr const\_iterator cend** () const noexcept  
*Sentinel for the coefficients const iterator.*
- **constexpr reverse\_iterator rbegin** () noexcept  
*Reverse iterator over the coefficients.*
- **constexpr const\_reverse\_iterator rbegin** () const noexcept  
*Reverse iterator over the coefficients.*
- **constexpr const\_reverse\_iterator crbegin** () const noexcept  
*Const reverse iterator over the coefficients.*
- **constexpr reverse\_iterator rend** () noexcept  
*Sentinel for the coefficients reverse iterator.*
- **constexpr const\_reverse\_iterator rend** () const noexcept  
*Sentinel for the coefficients reverse iterator.*
- **constexpr const\_reverse\_iterator crend** () const noexcept  
*Sentinel for the coefficients const reverse iterator.*
- **constexpr bool empty** () const noexcept  
*Test, if the matrix is empty.*
- **constexpr size\_type size** () const noexcept  
*Number of elements in the matrix.*
- **constexpr void swap** (Matrix &other) noexcept(m\_data.swap(std::declval< std::vector< **value\_type** > >()))  
*Specialized swap algorithm for the matrix.*
- **constexpr size\_type nrows** () const noexcept  
*Number of rows.*
- **constexpr size\_type ncols** () const noexcept  
*Number of columns.*
- **constexpr bool operator==** (Matrix const &) const =default

*Equality comparison for matrices.*

- `constexpr reference operator[] (size_type row, size_type col)`

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

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

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

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

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

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

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

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

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

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

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

*Returns true if matrix is zero, false otherwise.*

## Static Public Member Functions

- `static constexpr Matrix zero (size_type n)`

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

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

*Return an identity matrix.*

### 6.26.1 Detailed Description

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

A `Matrix` class.

A two-dimensional array representing a mathematical matrix.

## 6.26.2 Constructor & Destructor Documentation

### 6.26.2.1 Matrix()

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

Construct a matrix from a range.

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

#### Parameters

|              |                             |
|--------------|-----------------------------|
| <i>data</i>  | Range with the coefficients |
| <i>nrows</i> | Number of rows              |
| <i>ncols</i> | Number of columns           |

## 6.26.3 Member Function Documentation

### 6.26.3.1 at() [1/2]

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

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

#### Parameters

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

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

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

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

#### Parameters

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

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

### 6.26.3.3 operator\*=( )

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

Multiplies itself by rhs.

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

### 6.26.3.4 operator[]() [1/2]

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

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

#### Parameters

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

### 6.26.3.5 operator[]() [2/2]

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

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

#### Parameters

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

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

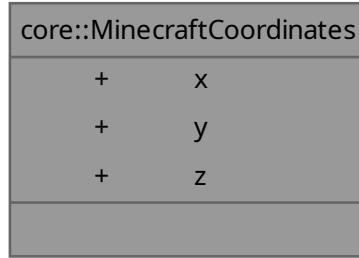
- mc-homology/algebra/include/algebra/[matrix.h](#)

## 6.27 core::MinecraftCoordinates Struct Reference

A struct containing coordinates in a Minecraft world.

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

Collaboration diagram for core::MinecraftCoordinates:



### Public Attributes

- int **x** = 0  
*x coordinate*
- int **y** = 0  
*y coordinate*
- int **z** = 0  
*z coordinate*

### 6.27.1 Detailed Description

A struct containing coordinates in a Minecraft world.

Minecraft world coordinates. Note, that y defines height.

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

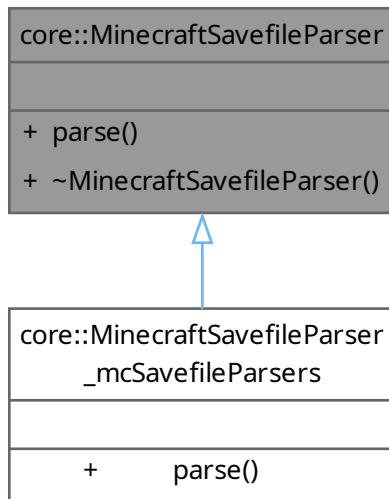
- mc-homology/core/include/core/parser.h

## 6.28 core::MinecraftSavefileParser Class Reference

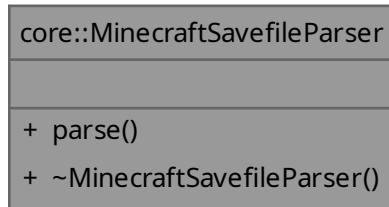
An interface for Minecraft savefile parser.

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

Inheritance diagram for core::MinecraftSavefileParser:



Collaboration diagram for core::MinecraftSavefileParser:



### Public Member Functions

- `virtual std::unique_ptr<Complex> parse (std::filesystem::path const &path, MinecraftCoordinates lower_corner, MinecraftCoordinates upper_corner)=0`  
*Parses a Minecraft savefile.*

### 6.28.1 Detailed Description

An interface for Minecraft savefile parser.

### 6.28.2 Member Function Documentation

#### 6.28.2.1 parse()

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

Parses a Minecraft savefile.

Parses a Minecraft savefile within given bounds.  $\text{lower\_corner.x} \leq \text{upper\_corner.x}$   $\text{lower\_corner.y} \leq \text{upper\_corner.y}$   $\text{lower\_corner.z} \leq \text{upper\_corner.z}$

#### 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:

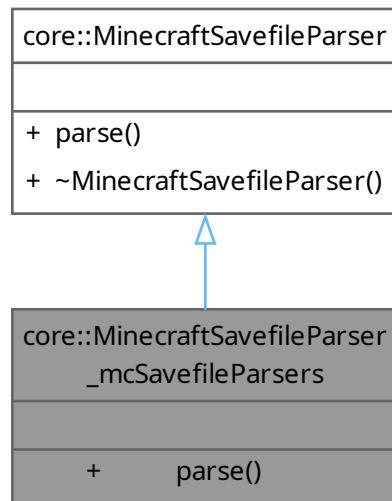
- mc-homology/core/include/core/parser.h
- mc-homology/core/src/parser.cpp

## 6.29 core::MinecraftSavefileParser\_mcSavefileParsers Class Reference

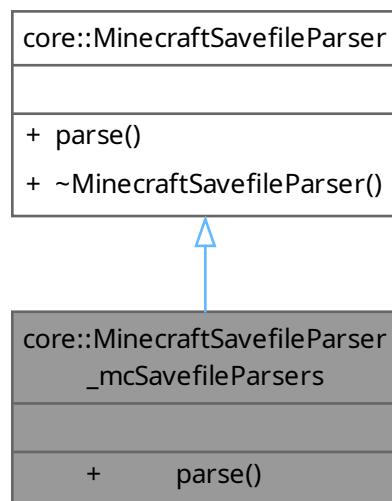
A Minecraft savefile parser based on <https://github.com/TCA166/mcSavefileParsers.git>.

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

Inheritance diagram for core::MinecraftSavefileParser\_mcSavefileParsers:



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.*

### 6.29.1 Detailed Description

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

### 6.29.2 Member Function Documentation

#### 6.29.2.1 parse()

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

Parses a Minecraft savefile.

Parses a Minecraft savefile within given bounds.  $\text{lower\_corner.x} \leq \text{upper\_corner.x}$   $\text{lower\_corner.y} \leq \text{upper\_corner.y}$   $\text{lower\_corner.z} \leq \text{upper\_corner.z}$

#### 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       |

Implements [core::MinecraftSavefileParser](#).

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

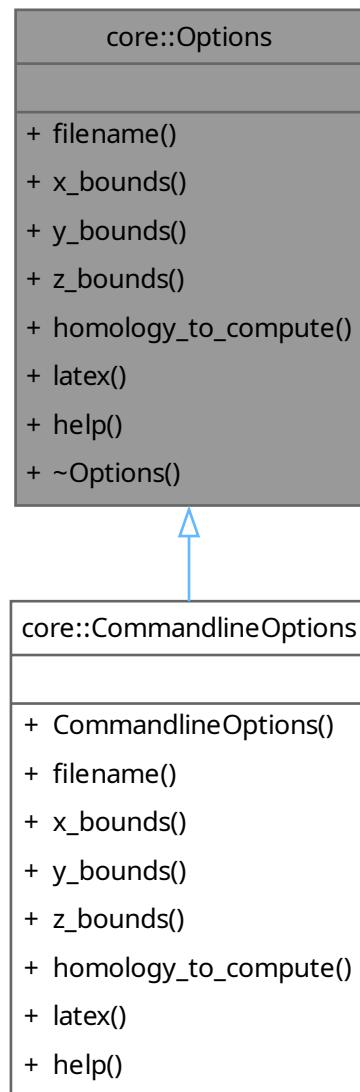
- mc-homology/core/include/core/parser.h
- mc-homology/core/src/parser.cpp

## 6.30 core::Options Class Reference

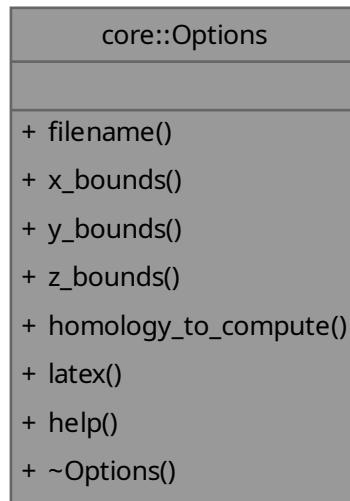
A class for storing user options.

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

Inheritance diagram for core::Options:



Collaboration diagram for core::Options:



### Public Member Functions

- virtual std::filesystem::path [filename](#) () const =0  
*Path to the region directory.*
- virtual std::pair< int, int > [x\\_bounds](#) () const =0  
*Bounds on x axis.*
- virtual std::pair< int, int > [y\\_bounds](#) () const =0  
*Bounds on y axis.*
- virtual std::pair< int, int > [z\\_bounds](#) () const =0  
*Bounds on 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*

### 6.30.1 Detailed Description

A class for storing user options.

## 6.30.2 Member Function Documentation

### 6.30.2.1 filename()

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

Path to the region directory.

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

### 6.30.2.2 help()

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

Whether the user requested help.

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

### 6.30.2.3 homology\_to\_compute()

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

Type of homology to compute.

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

### 6.30.2.4 latex()

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

Whether to print latex output.

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

### 6.30.2.5 x\_bounds()

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

Bounds on x axis.

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

### 6.30.2.6 y\_bounds()

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

Bounds on y axis.

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

### 6.30.2.7 z\_bounds()

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

Bounds on y axis.

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

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

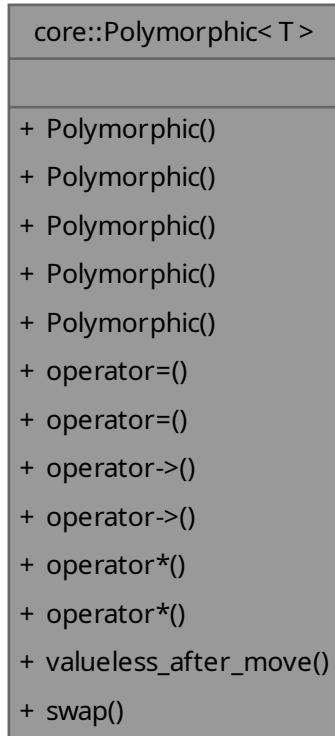
- mc-homology/core/include/core/[options.h](#)
- mc-homology/core/src/options.cpp

## 6.31 core::Polymorphic< T > Class Template Reference

A polymorphic class wrapper with value semantics.

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

Collaboration diagram for core::Polymorphic< T >:



## Public Member Functions

- `constexpr Polymorphic ()`  
*Constructs a default initialized object.*
- `template<class U = T>`  
`constexpr Polymorphic (U &&u)`  
*Constructs the object from the object u.*
- `template<class U, class... Args>`  
`constexpr Polymorphic (std::in_place_type_t< U >, Args &&... args)`  
*Constructs the object in place.*
- `constexpr Polymorphic (Polymorphic const &other)`  
*Copy constructor.*
- `constexpr Polymorphic (Polymorphic &&other) noexcept`  
*Move constructor.*
- `constexpr Polymorphic & operator= (Polymorphic const &other)`  
*Copy assignment.*
- `constexpr Polymorphic & operator= (Polymorphic &&other) noexcept`  
*Move assignment.*
- `constexpr T const * operator-> () const noexcept`  
*Arrow operator.*
- `constexpr T * operator-> () noexcept`  
*Arrow operator.*
- `constexpr T const & operator* () const noexcept`  
*Dereference operator.*
- `constexpr T & operator* () noexcept`  
*Dereference operator.*
- `constexpr bool valueless_after_move () const noexcept`  
*Returns true, if the object has been moved out of.*
- `constexpr void swap (Polymorphic &other) noexcept`  
*Swaps two objects together.*

### 6.31.1 Detailed Description

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

A polymorphic class wrapper with value semantics.

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

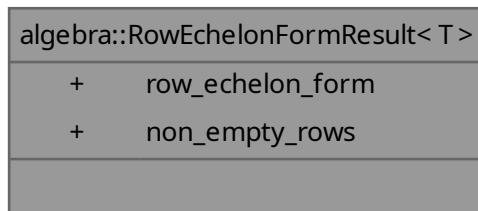
- `mc-homology/core/include/core/polymorphic.h`

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

Result struct for the row echelon algorithm.

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

Collaboration diagram for algebra::RowEchelonFormResult< T >:



### Public Attributes

- `Matrix< T > row_echelon_form = {}`  
*Row echelon form of a matrix.*
- `std::size_t non_empty_rows = 0`  
*Number of non-empty rows of the matrix in row echelon form.*

### 6.32.1 Detailed Description

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

Result struct for the row echelon algorithm.

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

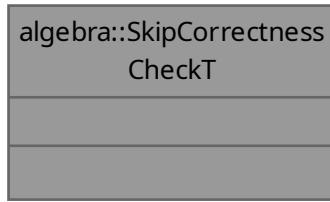
- [mc-homology/algebra/include/algebra/matrix\\_algorithms.h](#)

## 6.33 algebra::SkipCorrectnessCheckT Struct Reference

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

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

Collaboration diagram for algebra::SkipCorrectnessCheckT:



### 6.33.1 Detailed Description

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

Use with caution!

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

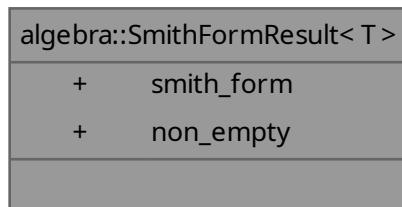
- mc-homology/algebra/include/algebra/[chain\\_complex.h](#)

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

Result struct for the smith algorithm.

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

Collaboration diagram for algebra::SmithFormResult< T >:



### Public Attributes

- `Matrix< T > smith_form = {}`  
*Smith form of a matrix.*
- `std::size_t non_empty = 0`  
*Number of non zero rows or columns.*

### 6.34.1 Detailed Description

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

Result struct for the smith algorithm.

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

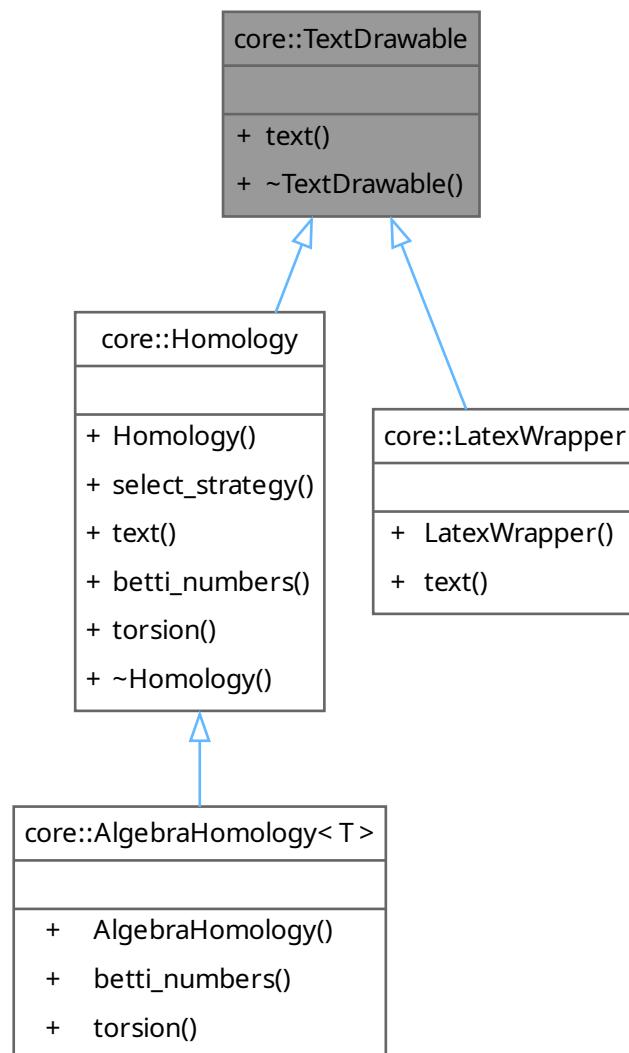
- mc-homology/algebra/include/algebra/[matrix\\_algorithms.h](#)

## 6.35 core::TextDrawable Class Reference

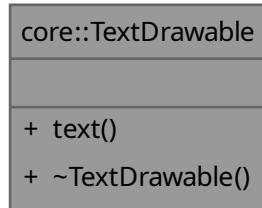
Marks a class text drawable - it can be represented as a string.

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

Inheritance diagram for core::TextDrawable:



Collaboration diagram for core::TextDrawable:



### Public Member Functions

- virtual std::string [text \(\) const =0](#)  
*Returns the text representation of the object.*
- virtual [~TextDrawable \(\)](#)  
*Virtual destructor.*

### 6.35.1 Detailed Description

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

### 6.35.2 Member Function Documentation

#### 6.35.2.1 [text\(\)](#)

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

Returns the text representation of the object.

Implemented in [core::Homology](#), and [core::LatexWrapper](#).

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

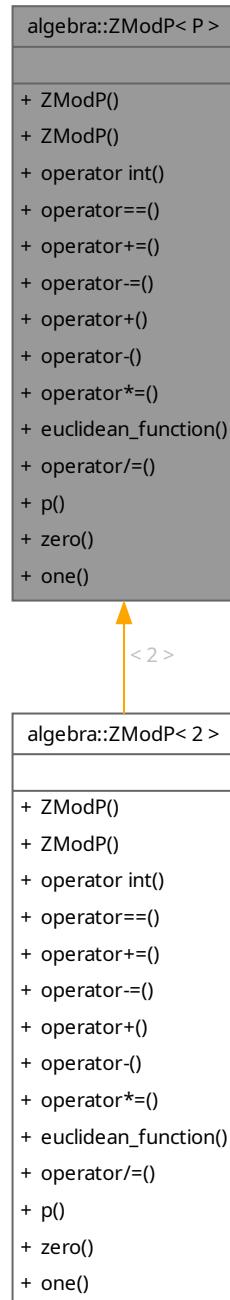
- mc-homology/core/include/core/[text\\_drawable.h](#)
- mc-homology/core/src/text\_drawable.cpp

## 6.36 algebra::ZModP< P > Class Template Reference

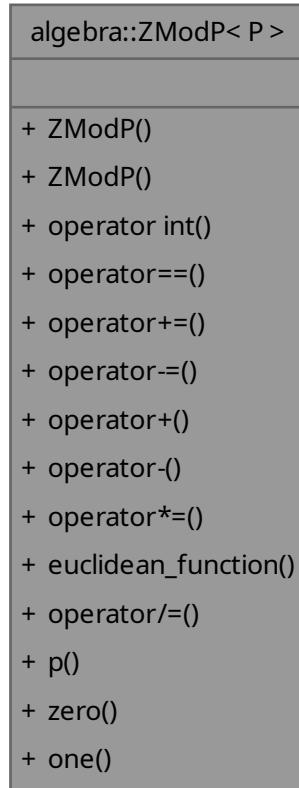
Field of integers modulo P.

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

Inheritance diagram for algebra::ZModP< P >:



Collaboration diagram for algebra::ZModP< P >:



## Public Member Functions

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

*Multiples itself by rhs.*

- constexpr int **euclidean\_function** () const noexcept
 

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

*Divides itself by rhs.*

## Static Public Member Functions

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

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

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

*Returns 1.*

## 6.36.1 Detailed Description

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

**Field** of integers modulo P.

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

### Parameters

|   |                |
|---|----------------|
| P | A prime number |
|---|----------------|

## 6.36.2 Member Function Documentation

### 6.36.2.1 euclidean\_function()

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

Euclidean function for a field.

Euclidean function for fields is constantly equal to 1.

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

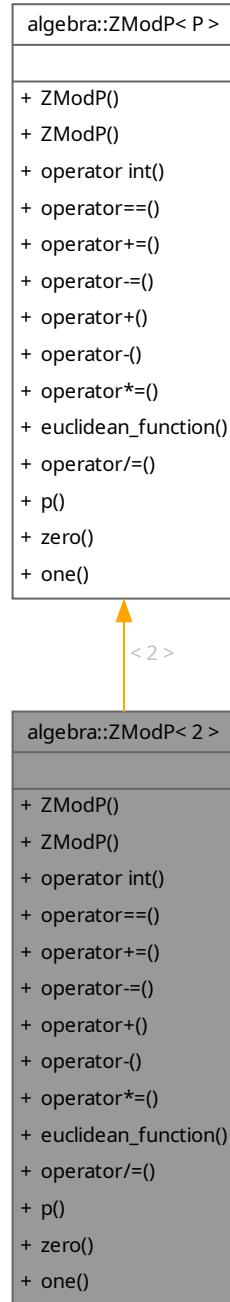
- mc-homology/algebra/include/algebra/modulo\_fields.h

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

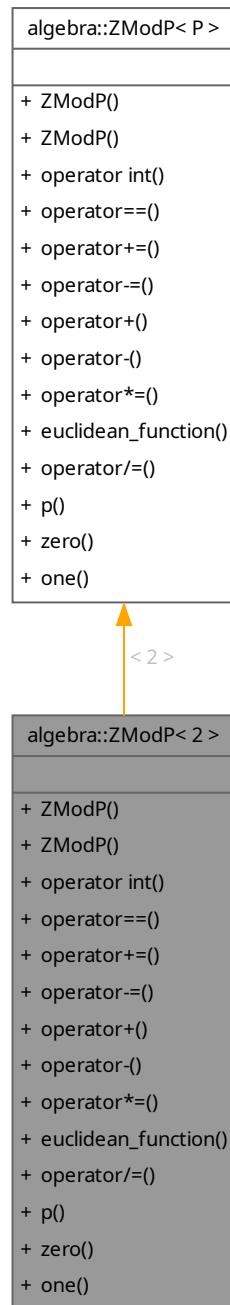
Template specialization for P == 2.

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

Inheritance diagram for algebra::ZModP< 2 >:



Collaboration diagram for algebra::ZModP< 2 >:



## Public Member Functions

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

- `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`  
*Multiples itself by rhs.*
- `constexpr int euclidean_function() const noexcept`  
*Euclidean function for fields.*
- `constexpr ZModP & operator/=(ZModP rhs)`  
*Divides itself by rhs.*

## Static Public Member Functions

- `static constexpr int p() noexcept`  
*Returns the modulus P.*
- `static constexpr ZModP zero() noexcept`  
*Returns 0.*
- `static constexpr ZModP one() noexcept`  
*Returns 1.*

### 6.37.1 Detailed Description

Template specialization for  $P == 2$ .

### 6.37.2 Member Function Documentation

#### 6.37.2.1 `euclidean_function()`

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

Euclidean function for fields.

Euclidean function for fields is constantly equal to 1.

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

- mc-homology/algebra/include/algebra/z2\_field.h

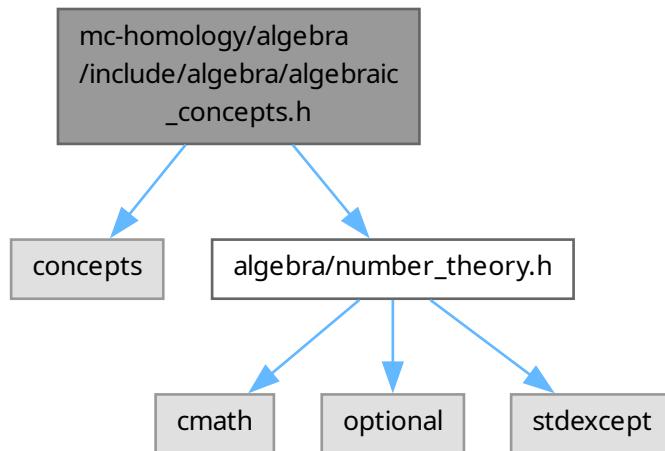
# Chapter 7

## File Documentation

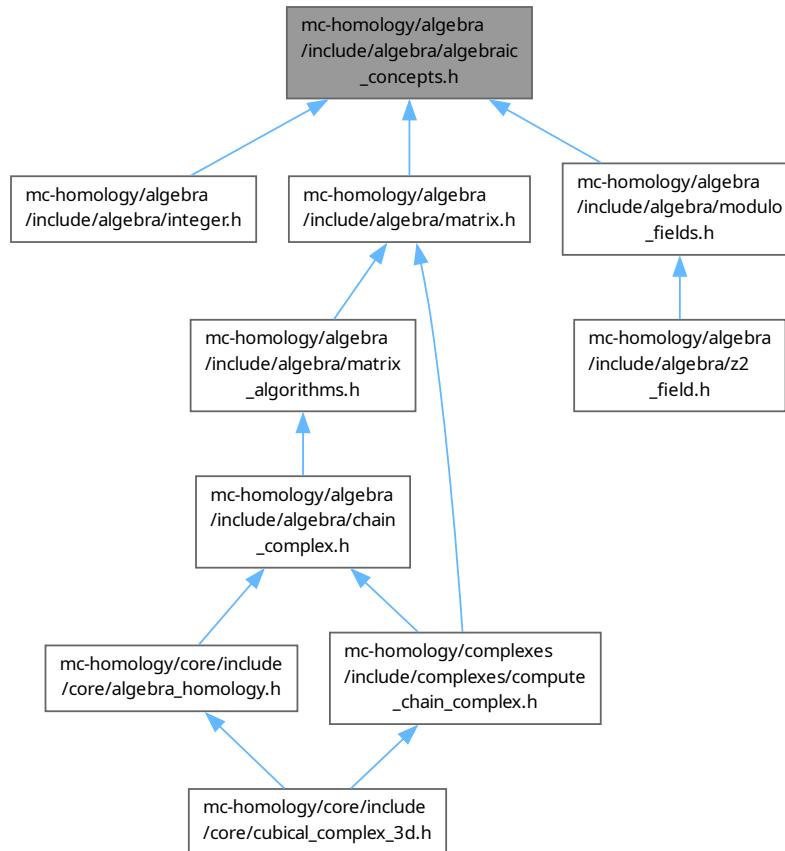
### 7.1 mc-homology/algebra/include/algebra/algebraic\_concepts.h File Reference

A file defining multiple concepts for algebraic structures.

```
#include <concepts>
#include "algebra/number_theory.h"
Include dependency graph for algebraic_concepts.h:
```



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



## 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.](#)

```

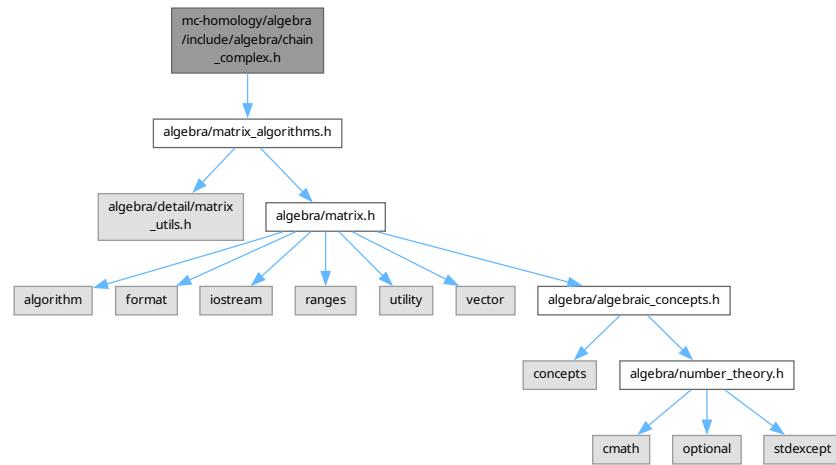
00001
00003
00004 #pragma once
00005
00006 #include <concepts>
00007
00008 #include "algebra/number_theory.h"
00009
00010 namespace algebra {
00011
00014 template<class T>
00015 constexpr inline bool is_commutative_v = false;
00016
00027 template<class T>
00028 concept Commutative = is_commutative_v<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

```

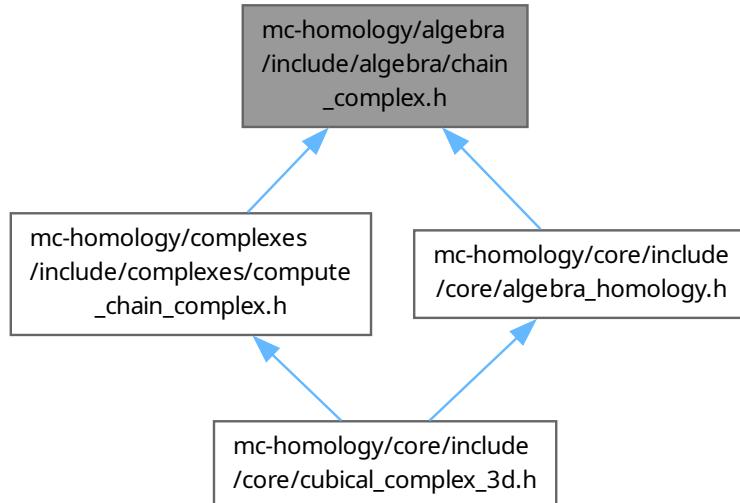
## 7.3 mc-homology/algebra/include/algebra/chain\_complex.h File Reference

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
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
00064     constexpr bool check_boundary_correctness() const {
00065         if (m_boundaries.size() < 2) {
00066             return true;
00067         }
00068         namespace rs = std::ranges;
00069         namespace vs = std::views;
00070         auto b_n_plus_1_view = m_boundaries | vs::drop(1);
00071         auto b_n_view = m_boundaries | vs::take(m_boundaries.size() - 1);
00072         return rs::all_of(
00073             vs::zip(b_n_plus_1_view, b_n_view),
00074             [](auto const& boundaries) {
00075                 try {
00076                     auto const& [b_n_plus_1, b_n] = boundaries;
00077                     return (b_n * b_n_plus_1).is_zero();
00078                 } catch (std::domain_error&) {
00079                     return false;
00080                 }
00081             });
00082     );
00083 }
00084
00087     constexpr std::size_t dimension() const noexcept {
00088         return m_boundaries.size();
00089     }
00090
00094     constexpr Matrix<T> const& boundary(std::size_t dim) const {
00095         return m_boundaries.at(dim);
00096     }
00097
00099     constexpr std::vector<Matrix<T>> const& boundaries() const noexcept {
00100         return m_boundaries;
00101     }
00102
00103 private:
00104     std::vector<Matrix<T>> m_boundaries;
00105 };
00106
00113 template<class T>
00114 struct Homology {
00116     std::vector<std::size_t> betti_numbers {};
00117
00123     std::vector<std::vector<T>> torsion {};
00124 };
00125
00128 template<EuclideanDomain T>
00129 Homology<T> homology(ChainComplex<T> const& chain_complex) {
00130     namespace rs = std::ranges;
00131     namespace vs = std::views;
00132     auto const& boundaries = chain_complex.boundaries();
00133     Homology<T> homology;
00134     homology.betti_numbers.resize(boundaries.size());
00135     homology.torsion.resize(boundaries.size());
00136     std::size_t prev_smith_units_count = 0;
00137     std::vector<T> prev_smith_diagonal_without_units {};
00138     for (std::size_t k = boundaries.size(); k > 0; --k) {
00139         auto const n = k - 1;
00140         auto const& boundary = boundaries[n];
00141         auto [smith, rank] = smith_form(boundary);
00142         auto nullity = boundary.ncols() - rank;
00143         auto smith_diagonal_without_units = rs::to<std::vector>(
00144             vs::iota(0u, rank)
00145             | vs::transform([&smith](std::size_t n) { return smith[n, n]; })
00146             | vs::drop_while([](T const& x) {
00147                 return x.euclidean_function() == 1;
00148             })
00149         );
00150         auto smith_units = rank - smith_diagonal_without_units.size();
00151         homology.betti_numbers[n] = nullity - prev_smith_units_count
00152             - prev_smith_diagonal_without_units.size();
00153         homology.torsion[n] = std::move(prev_smith_diagonal_without_units);
00154         prev_smith_units_count = smith_units;
00155         prev_smith_diagonal_without_units =
00156             std::move(smith_diagonal_without_units);
00157     }
00158     return homology;
00159 }
00160
00163 template<Field T>
00164 Homology<T> homology(ChainComplex<T> const& chain_complex) {
00165     auto const& boundaries = chain_complex.boundaries();
00166     Homology<T> homology;
00167     homology.betti_numbers.resize(boundaries.size());
00168     homology.torsion.resize(boundaries.size());
00169     std::size_t prev_rank = 0;
00170     for (std::size_t k = boundaries.size(); k > 0; --k) {

```

```

00171     auto const n = k - 1;
00172     auto const& boundary = boundaries[n];
00173     auto [_, rank] = row_echelon_form(boundary);
00174     auto nullity = boundary.ncols() - rank;
00175     homology.betti_numbers[n] = nullity - prev_rank;
00176     prev_rank = rank;
00177 }
00178 return homology;
00179 }
00180
00182 template<std::ranges::sized_range R>
00183 ChainComplex<SkipCorrectnessCheckT, R&&>
00184     -> ChainComplex<typename std::ranges::range_value_t<R>::value_type>;
00185
00187 template<std::ranges::sized_range R>
00188 ChainComplex<R&&>
00189     -> ChainComplex<typename std::ranges::range_value_t<R>::value_type>;
00190
00191 } // namespace algebra

```

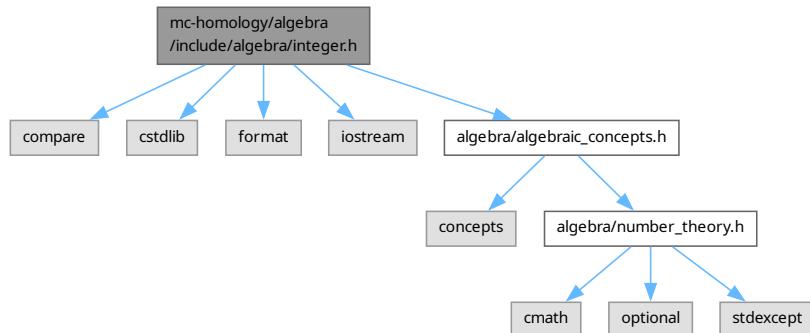
## 7.5 mc-homology/algebra/include/algebra/integer.h File Reference

A file containing Integer implementation satisfying algebraic concepts.

```

#include <compare>
#include <cstdlib>
#include <format>
#include <iostream>
#include "algebra/algebraic_concepts.h"
Include dependency graph for integer.h:

```



### Classes

- class [algebra::Integer](#)  
*Class of Integers.*
- struct [std::formatter< algebra::Integer >](#)  
*Formatter for Integer type.*

## Functions

- `constexpr Integer algebra::operator+ (Integer lhs, Integer rhs) noexcept`  
*Adds two integers.*
- `constexpr Integer algebra::operator- (Integer lhs, Integer rhs) noexcept`  
*Subtracts two integers.*
- `constexpr Integer algebra::operator* (Integer lhs, Integer rhs) noexcept`  
*Multiples 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:
00087     int m_inner_representation = 0;
00088 };
00089
00091 constexpr Integer operator+(Integer lhs, Integer rhs) noexcept {
00092     return lhs += rhs;
00093 }
00094
00096 constexpr Integer operator-(Integer lhs, Integer rhs) noexcept {
00097     return lhs -= rhs;
00098 }
00099
00101 constexpr Integer operator*(Integer lhs, Integer rhs) noexcept {
00102     return lhs *= rhs;
00103 }
00104
00106 constexpr Integer abs(Integer k) noexcept {
00107     return std::abs(static_cast<int>(k));
00108 }
```

```

00109
00115 constexpr DivResult<Integer> divide(Integer a, Integer b) {
00116     auto div_result = divide(static_cast<int>(a), static_cast<int>(b));
00117     .quotient = div_result.quotient,
00118     .remainder = div_result.remainder
00119 };
00120 }
00121 }
00122
00127 constexpr Integer modulo(Integer a, Integer n) {
00128     return divide(a, n).remainder;
00129 }
00130
00132 template<>
00133 constexpr inline bool is_commutative_v<Integer> = true;
00134
00136 inline std::ostream& operator<<(std::ostream& output, Integer k) {
00137     return output << static_cast<int>(k);
00138 }
00139
00140 } // namespace algebra
00141
00146 template<>
00147 struct std::formatter<algebra::Integer>: public std::formatter<int> {
00151     template<class FmtContext>
00152         FmtContext::iterator format(algebra::Integer k, FmtContext& ctx) const {
00153             return std::formatter<int>::format(static_cast<int>(k), ctx);
00154         }
00155 };

```

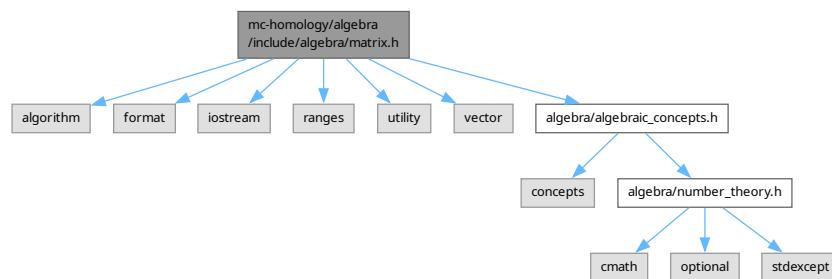
## 7.7 mc-homology/algebra/include/algebra/matrix.h File Reference

A file containing matrix implementation.

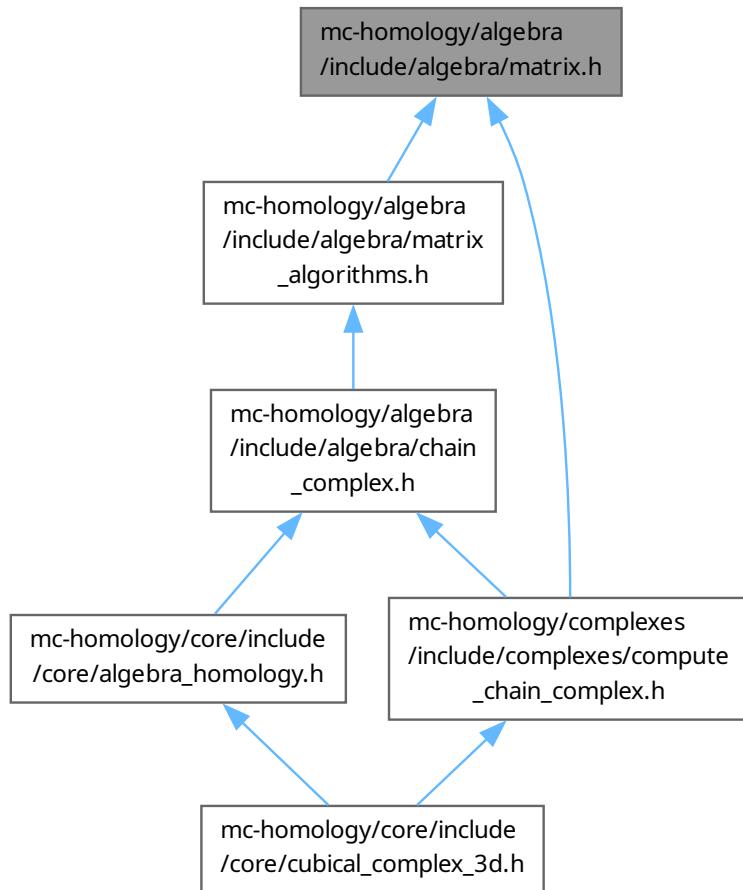
```

#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)`

- Multiples 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]
```

*Multiples two matrices.*

*Multiples two matrices using the usual matrix multiplication.*

## 7.8 matrix.h

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

```
00001
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
00076     constexpr iterator begin() noexcept {
00077         return m_data.begin();
00078     }
00079
00080     constexpr const_iterator begin() const noexcept {
00081         return m_data.begin();
00082     }
00083
00084     constexpr const_iterator cbegin() const noexcept {
00085         return m_data.cbegin();
00086     }
00087
00088     constexpr iterator end() noexcept {
00089         return m_data.end();
00090     }
00091
00092     constexpr const_iterator end() const noexcept {
00093         return m_data.end();
00094     }
00095
00096     constexpr const_iterator cend() const noexcept {
00097         return m_data.cend();
00098     }
00099
00100     constexpr reverse_iterator rbegin() noexcept {
00101         return m_data.rbegin();
00102     }
00103
00104     constexpr const_reverse_iterator rbegin() const noexcept {
00105         return m_data.rbegin();
00106     }
00107
00108     constexpr const_reverse_iterator crbegin() const noexcept {
00109         return m_data.crbegin();
00110     }
00111
00112     constexpr reverse_iterator rend() noexcept {
00113         return m_data.rend();
00114     }
00115
00116     constexpr const_reverse_iterator rend() const noexcept {
00117         return m_data.rend();
00118     }
00119
00120     constexpr const_reverse_iterator crend() const noexcept {
00121         return m_data.crend();
00122     }
00123
00124     constexpr bool empty() const noexcept {
00125         return m_data.empty();
00126     }
00127
00128     constexpr size_type size() const noexcept {
00129         return m_data.size();
00130     }
00131
00132     constexpr void swap(Matrix& other) noexcept(
00133         m_data.swap(std::declval<std::vector<value_type>>());
00134     ) {
00135         namespace rs = std::ranges;
00136         m_data.swap(other.m_data);
00137         rs::swap(m_nrows, other.m_nrows);
00138         rs::swap(m_ncols, other.m_ncols);
00139     }
00140
00141     constexpr size_type nrows() const noexcept {
00142         return m_nrows;
00143     }
00144
00145     constexpr size_type ncols() const noexcept {
00146         return m_ncols;
00147     }
```

```

00165
00166     constexpr bool operator==(Matrix const&) const = default;
00167
00168     constexpr reference operator[](size_type row, size_type col) {
00169         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00170             throw std::out_of_range("Indices out of matrix range");
00171         }
00172         return m_data[to_underlying_index(row, col)];
00173     }
00174
00175     constexpr const_reference operator[](size_type row, size_type col) const {
00176         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00177             throw std::out_of_range("Indices out of matrix range");
00178         }
00179         return m_data[to_underlying_index(row, col)];
00180     }
00181
00182     constexpr reference at(size_type row, size_type col) {
00183         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00184             throw std::out_of_range("Indices out of matrix range");
00185         }
00186         return m_data.at(to_underlying_index(row, col));
00187     }
00188
00189     constexpr const_reference at(size_type row, size_type col) const {
00190         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00191             throw std::out_of_range("Indices out of matrix range");
00192         }
00193         return m_data.at(to_underlying_index(row, col));
00194     }
00195
00196     constexpr const_reference at(size_type row, size_type col) const {
00197         if (row >= m_nrows || col >= m_ncols) [[unlikely]] {
00198             throw std::out_of_range("Indices out of matrix range");
00199         }
00200         return m_data.at(to_underlying_index(row, col));
00201     }
00202
00203     constexpr storage_type const& data() const noexcept {
00204         return m_data;
00205     }
00206
00207     constexpr Matrix transpose() const {
00208         auto transposed = zero(m_ncols, m_nrows);
00209         for (size_type i = 0; i < m_nrows; ++i) {
00210             for (size_type j = 0; j < m_ncols; ++j) {
00211                 transposed[j, i] = at(i, j);
00212             }
00213         }
00214         return transposed;
00215     }
00216
00217     constexpr Matrix& operator+=(Matrix const& rhs)
00218         requires AdditiveGroup<T>
00219     {
00220         if (m_nrows != rhs.m_nrows || m_ncols != rhs.m_ncols) {
00221             throw std::domain_error("Adding matrices of different dimensions");
00222         }
00223         namespace rs = std::ranges;
00224         rs::transform(m_data, rhs.m_data, rs::begin(m_data), std::plus {});
00225         return *this;
00226     }
00227
00228     constexpr Matrix& operator-=(Matrix const& rhs)
00229         requires AdditiveGroup<T>
00230     {
00231         if (m_nrows != rhs.m_nrows || m_ncols != rhs.m_ncols) {
00232             throw std::domain_error(
00233                 "Subtracting matrices of different dimensions"
00234             );
00235         }
00236         namespace rs = std::ranges;
00237         rs::transform(m_data, rhs.m_data, rs::begin(m_data), std::minus {});
00238         return *this;
00239     }
00240
00241     constexpr Matrix operator+() const
00242         requires AdditiveGroup<T>
00243     {
00244         return *this;
00245     }
00246
00247     constexpr Matrix operator-() const
00248         requires AdditiveGroup<T>
00249     {
00250         namespace rs = std::ranges;
00251         rs::transform(m_data, rs::begin(m_data), std::minus {});
00252     }
00253
00254     constexpr Matrix operator+() const
00255         requires AdditiveGroup<T>
00256     {
00257         return *this;
00258     }
00259
00260     constexpr Matrix operator-() const
00261         requires AdditiveGroup<T>
00262     {
00263         namespace rs = std::ranges;
00264         std::vector<T> negated;
00265         negated.reserve(m_data.capacity());
00266         rs::transform(m_data, rs::begin(negated), std::negate {});
00267         return Matrix(negated, m_nrows, m_ncols);
00268     }
00269
00270     constexpr Matrix operator*=(Matrix const& rhs)
00271         requires Ring<T>
00272     {

```

```
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
00289     constexpr static Matrix zero(size_type n, size_type m)
00290         requires AdditiveGroup<T>
00291     {
00292         std::vector data(n * m, value_type::zero());
00293         return Matrix(std::move(data), n, m);
00294     }
00295
00296
00297     constexpr bool is_zero() const noexcept {
00298         return std::ranges::all_of(m_data, [](T const& x) {
00299             return x == T::zero();
00300         });
00301     }
00302
00303
00304     constexpr static Matrix id(size_type n)
00305         requires CommutativeRing<T>
00306     {
00307         auto identity = zero(n);
00308         for (size_type i = 0; i < n; ++i) {
00309             identity[i, i] = value_type::one();
00310         }
00311         return identity;
00312     }
00313
00314
00315 private:
00316     constexpr size_type
00317         to_underlying_index(size_type row, size_type col) const noexcept {
00318             return row * m_ncols + col;
00319         }
00320
00321     storage_type m_data;
00322     size_type m_nrows;
00323     size_type m_ncols;
00324 };
00325
00326 template<AdditiveGroup T>
00327 constexpr Matrix<T> operator+(Matrix<T> lhs, Matrix<T> const& rhs) {
00328     return lhs += rhs;
00329 }
00330
00331
00332 template<AdditiveGroup T>
00333 constexpr Matrix<T> operator-(Matrix<T> lhs, Matrix<T> const& rhs) {
00334     return lhs -= rhs;
00335 }
00336 }
00337
00338 template<Ring T>
00339 constexpr Matrix<T> operator*(Matrix<T> const& lhs, Matrix<T> const& rhs) {
00340     using Matrix = Matrix<T>;
00341     using size_type = Matrix::size_type;
00342     if (lhs.ncols() != rhs.nrows()) {
00343         throw std::domain_error(
00344             "The number of columns of lhs is different "
00345             "than the number of rows of rhs"
00346         );
00347     }
00348     auto const inner_dim = lhs.ncols();
00349     Matrix product = Matrix::zero(lhs.nrows(), rhs.ncols());
00350     for (size_type i = 0; i < product.nrows(); ++i) {
00351         for (size_type j = 0; j < product.ncols(); ++j) {
00352             for (size_type k = 0; k < inner_dim; ++k) {
00353                 product[i, j] += lhs[i, k] * rhs[k, j];
00354             }
00355         }
00356     }
00357     return product;
00358 }
00359
00360
00361
00362 template<std::ranges::sized_range R>
00363 Matrix(R&&, std::size_t, std::size_t) -> Matrix<std::ranges::range_value_t<R>>;
00364
00365
00366
00367 template<class T>
00368 std::ostream& operator<<(std::ostream& output, Matrix<T> const& matrix) {
00369     return output << std::format("{}", matrix);
00370 }
00371
00372
00373 } // namespace algebra
00374
00375 template<class T>
```

```

00399     requires std::formattable<T, char>
00400 struct std::formatter<algebra::Matrix<T>> {
00401     template<class ParseContext>
00402     constexpr ParseContext::iterator parse(ParseContext& ctx) {
00403         auto it = ctx.begin();
00404         if (it == ctx.end() || *it == '}') {
00405             return it;
00406         } else if (*it == '-') {
00407             multi_line = false;
00408             ++it;
00409         } else if (*it == '#') {
00410             multi_line = true;
00411             ++it;
00412         }
00413     }
00414
00415     if (it == ctx.end() || *it == '}') {
00416         return it;
00417     } else if (*it == ':') {
00418         ctx.advance_to(++it);
00419         return coefficient_formatter.parse(ctx);
00420     } else {
00421         throw std::format_error("Invalid formatting option for Matrix");
00422     }
00423 }
00424
00425 template<class FmtContext>
00426 FmtContext::iterator
00427 format(algebra::Matrix<T> const& matrix, FmtContext& ctx) const {
00428     auto const maybe_new_line = multi_line ? "\n" : "";
00429     if (matrix.empty()) {
00430         std::format_to(ctx.out(), "[]");
00431     } else {
00432         std::format_to(ctx.out(), "[");
00433         for (std::size_t i = 0; i < matrix.nrows(); ++i) {
00434             std::format_to(ctx.out(), "[");
00435             for (std::size_t j = 0; j < matrix.ncols(); ++j) {
00436                 coefficient_formatter.format(matrix[i, j], ctx);
00437                 if (j + 1 < matrix.ncols()) {
00438                     std::format_to(ctx.out(), ", ");
00439                 }
00440             }
00441             std::format_to(ctx.out(), "]");
00442             if (i + 1 < matrix.nrows()) {
00443                 std::format_to(ctx.out(), ", {}, ", maybe_new_line);
00444             }
00445         }
00446         std::format_to(ctx.out(), "]");
00447     }
00448 }
00449
00450     if (multi_line) {
00451         std::format_to(
00452             ctx.out(),
00453             "\nMatrix {} x {}\n",
00454             matrix.nrows(),
00455             matrix.ncols()
00456         );
00457     }
00458     return ctx.out();
00459 }
00460
00461 }
00462
00463 private:
00464     bool multi_line = false;
00465     std::formatter<T> coefficient_formatter;
00466 };

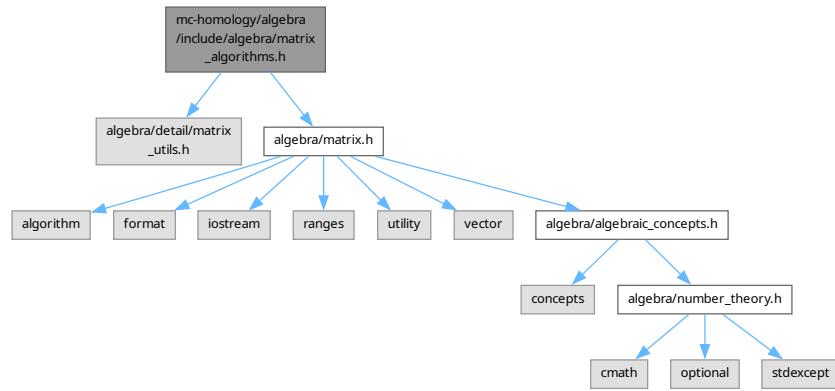
```

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

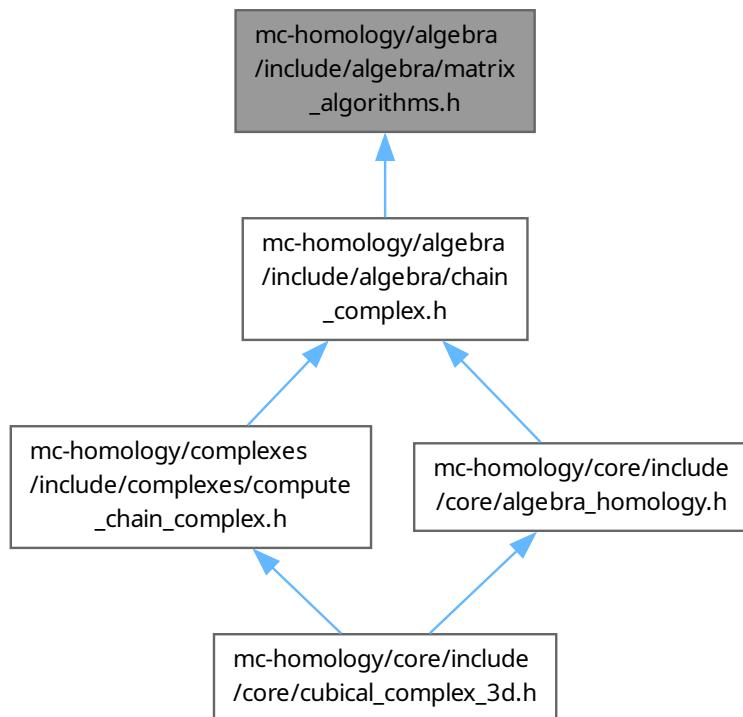
A file containing selected matrix algorithms.

```
#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 echelon 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

|                     |                          |
|---------------------|--------------------------|
| <code>matrix</code> | 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 echelon form in place.

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

#### Parameters

|         |        |                          |
|---------|--------|--------------------------|
| in, out | matrix | Matrix to be transformed |
|---------|--------|--------------------------|

#### Returns

The number of non-zero rows

### 7.9.2.3 smith\_form() [1/2]

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

Transforms a matrix into a Smith form.

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

#### Parameters

|        |                          |
|--------|--------------------------|
| matrix | Matrix to be transformed |
|--------|--------------------------|

#### Returns

A struct containing two fields

1. smith\_form The transformed matrix
2. non\_empty The number of non-zero rows or columns

### 7.9.2.4 smith\_form() [2/2]

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

Transforms a matrix into a Smith form in place.

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

#### Parameters

|         |        |                          |
|---------|--------|--------------------------|
| in, out | matrix | Matrix to be transformed |
|---------|--------|--------------------------|

#### Returns

The number of non-zero rows or columns

## 7.10 matrix\_algorithms.h

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

```

00001
00003
00004 #pragma once
00005
00006 #include "algebra/detail/matrix_utils.h"
00007 #include "algebra/matrix.h"
00008
00009 namespace algebra {
00010
00019 template<Field T>
00020 constexpr std::size_t row_echelon_form(std::in_place_t, Matrix<T>& matrix) {
00021     std::size_t i = 0;
00022     for (std::size_t j = 0; j < matrix.ncols(); ++j) {
00023         auto maybe_i =
00024             detail::first_nonzero_submatrix_column_coefficient(matrix, i, j);
00025         if (!maybe_i) {
00026             continue;
00027         }
00028         if (*maybe_i != i) {
00029             detail::submatrix_swap_rows(matrix, i, *maybe_i, j);
00030         }
00031         for (auto k = i + 1; k < matrix.nrows(); ++k) {
00032             auto mult = -matrix[k, j] / matrix[i, j];
00033             detail::submatrix_add_row(matrix, mult, i, k, j);
00034         }
00035         ++i;
00036     }
00037     return i;
00038 }
00039
00041 template<class T>
00042 struct RowEchelonFormResult {
00044     Matrix<T> row_echelon_form = {};
00045
00048     std::size_t non_empty_rows = 0;
00049 };
00050
00061 template<Field T>
00062 constexpr RowEchelonFormResult<T> row_echelon_form(Matrix<T> matrix) {
00063     auto i = row_echelon_form(std::in_place, matrix);
00064     return RowEchelonFormResult {
00065         .row_echelon_form = matrix,
00066         .non_empty_rows = i
00067     };
00068 }
00069
00078 template<EuclideanDomain T>
00079 constexpr std::size_t smith_form(std::in_place_t, Matrix<T>& matrix) {
00080     std::size_t k = 0;
00081     auto move_min_to_corner = [&] {
00082         auto min_element =
00083             detail::minimal_nonzero_submatrix_element(matrix, k, k);
00084         if (matrix[min_element.first, min_element.second] == T::zero()) {
00085             return;
00086         }
00087         if (min_element.first != k) {
00088             detail::submatrix_swap_rows(matrix, k, min_element.first, k);
00089         }
00090         if (min_element.second != k) {
00091             detail::submatrix_swap_cols(matrix, k, min_element.second, k);
00092         }
00093     };
00094     for (; k < std::min(matrix.nrows(), matrix.ncols()); ++k) {
00095         bool col_all_zeros = false;
00096         while (!col_all_zeros) {
00097             move_min_to_corner();
00098             if (matrix[k, k] == T::zero()) {
00099                 return k;
00100             }
00101             col_all_zeros = true;
00102             for (std::size_t i = k + 1; i < matrix.nrows(); ++i) {
00103                 auto [q, r] = divide(matrix[i, k], matrix[k, k]);
00104                 detail::submatrix_add_row(matrix, -q, k, i, k);
00105                 if (r != T::zero()) {
00106                     col_all_zeros = false;
00107                 }
00108             }
00109         }
00110         bool row_all_zeros = false;
00111         while (!row_all_zeros) {
00112             move_min_to_corner();
00113             if (matrix[k, k] == T::zero()) {

```

```

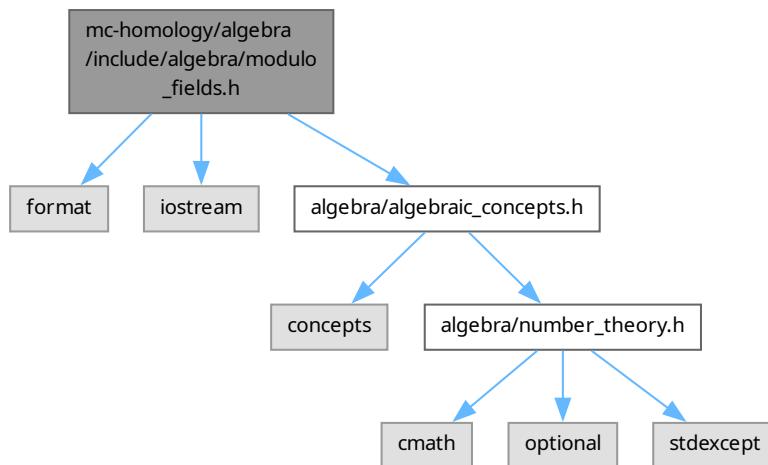
00114         return k;
00115     }
00116     row_all_zeros = true;
00117     for (std::size_t j = k + 1; j < matrix.ncols(); ++j) {
00118         auto [q, r] = divide(matrix[k, j], matrix[k, k]);
00119         detail::submatrix_add_col(matrix, -q, k, j, k);
00120         if (r != T::zero()) {
00121             row_all_zeros = false;
00122         }
00123     }
00124 }
00125 if constexpr (std::totally_ordered<T>) {
00126     for (std::size_t i = 0; i < k; ++i) {
00127         if (matrix[i, i] < T::zero()) {
00128             matrix[i, i] = -matrix[i, i];
00129         }
00130     }
00131 }
00132 }
00133 return k;
00134 }
00135
00137 template<class T>
00138 struct SmithFormResult {
00139     Matrix<T> smith_form = {};
00140     std::size_t non_empty = 0;
00141 };
00142
00144 template<EuclideanDomain T>
00145 constexpr SmithFormResult<T> smith_form(Matrix<T> matrix) {
00146     auto k = smith_form(std::in_place, matrix);
00147     return SmithFormResult {.smith_form = matrix, .non_empty = k};
00148 }
00149
00150 } // namespace algebra

```

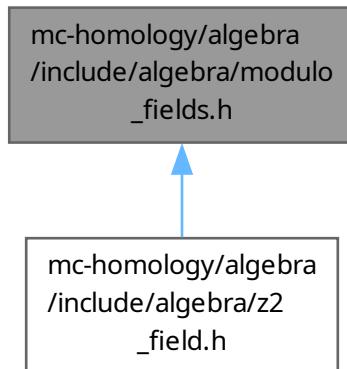
## 7.11 mc-homology/algebra/include/algebra/modulo\_fields.h File Reference

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
00019 template<int P>
00020 class ZModP {
00021     static_assert(is_prime(P));
00022
00023 public:
00025     constexpr ZModP() = default;
00026
00028     constexpr ZModP(int n) noexcept : m_inner_representation(modulo(n, P)) {}
00029
00031     constexpr static int p() noexcept {
00032         return P;
00033     }
00034
00036     constexpr static ZModP zero() noexcept {
00037         return ZModP(0);
00038     }
00039
00041     constexpr static ZModP one() noexcept {
00042         return ZModP(1);
00043     }
00044
00046     constexpr explicit operator int() const noexcept {
00047         return m_inner_representation;
00048     }
00049
00051     constexpr bool operator==(ZModP const&) const = default;
00052
00054     constexpr ZModP& operator+=(ZModP rhs) noexcept {
00055         m_inner_representation =
00056             modulo(m_inner_representation + rhs.m_inner_representation, P);
00057         return *this;
00058     }
00059
00061     constexpr ZModP& operator-=(ZModP rhs) noexcept {
00062         m_inner_representation =
00063             modulo(m_inner_representation - rhs.m_inner_representation, P);
00064         return *this;
00065     }
00066
00068     constexpr ZModP operator+() const noexcept {
00069         return *this;
00070     }
00071
00073     constexpr ZModP operator-() const noexcept {
00074         return ZModP(modulo(-m_inner_representation, P));
00075     }
00076
00078     constexpr ZModP& operator*=(ZModP rhs) noexcept {
00079         m_inner_representation =
00080             modulo(m_inner_representation * rhs.m_inner_representation, P);
00081         return *this;
00082     }
00083
00087     constexpr int euclidean_function() const noexcept {
00088         return 1;
00089     }
00090
00092     constexpr ZModP& operator/=(ZModP rhs) {
00093         auto inverse = inverse_mod(rhs.m_inner_representation, P);
00094         if (!inverse) [[unlikely]] {
00095             throw std::domain_error("Division by 0");
00095 }
```

```

00096         }
00097         return *this *= *inverse;
00098     }
00099
00100 private:
00101     int m_inner_representation = 0;
00102 };
00103
00105 template<int P>
00106 constexpr ZModP<P> operator+(ZModP<P> lhs, ZModP<P> rhs) noexcept {
00107     return lhs += rhs;
00108 }
00109
00111 template<int P>
00112 constexpr ZModP<P> operator-(ZModP<P> lhs, ZModP<P> rhs) noexcept {
00113     return lhs -= rhs;
00114 }
00115
00117 template<int P>
00118 constexpr ZModP<P> operator*(ZModP<P> lhs, ZModP<P> rhs) noexcept {
00119     return lhs *= rhs;
00120 }
00121
00123 template<int P>
00124 constexpr ZModP<P> operator/(ZModP<P> lhs, ZModP<P> rhs) {
00125     return lhs /= rhs;
00126 }
00127
00129 template<int P>
00130 std::ostream& operator<<(std::ostream& output, ZModP<P> x) {
00131     return output << static_cast<int>(x);
00132 }
00133
00135 template<int P>
00136 constexpr inline bool is_commutative_v<ZModP<P>> = true;
00137
00138 } // namespace algebra
00139
00144 template<int P>
00145 struct std::formatter<algebra::ZModP<P>> : public std::formatter<int> {
00146     template<class FmtContext>
00147     FmtContext::iterator format(algebra::ZModP<P> x, FmtContext& ctx) const {
00148         return std::formatter<int>::format(static_cast<int>(x), ctx);
00149     }
00150 }
00151 };

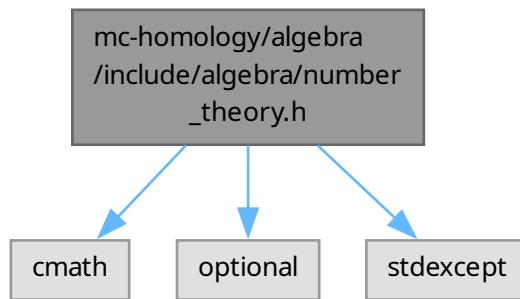
```

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

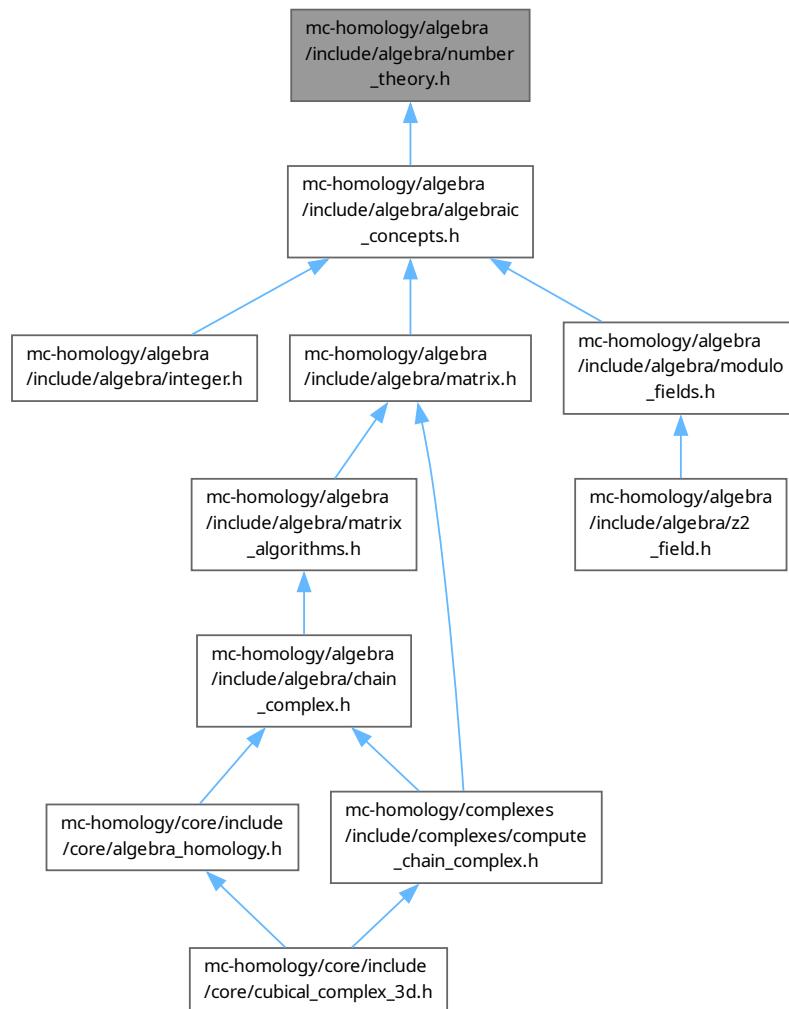
A file containing implementations of selected number theoretic algorithms.

```
#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  $a == \text{quotient} * n + \text{remainder}$  and  $0 \leq \text{remainder} < \text{abs}(n)$

#### Parameters

|   |            |
|---|------------|
| a | An integer |
| b | An integer |

#### Returns

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

#### 7.13.2.2 extended\_gcd()

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

Extended Euclidean algorithm.

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

#### Parameters

|   |            |
|---|------------|
| a | An integer |
| b | An integer |

#### Returns

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

### 7.13.2.3 inverse\_mod()

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

Inverse modulo n.

Calculates inverse of a mod n.

#### Parameters

|   |                 |
|---|-----------------|
| a | Inverted number |
| n | The modulus     |

#### Returns

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

|   |                |
|---|----------------|
| n | Number to test |
|---|----------------|

#### Returns

true, if the number is prime, otherwise false

### 7.13.2.5 modulo()

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

Calculates the remainder of a by n.

#### Parameters

|   |            |
|---|------------|
| a | An integer |
|---|------------|

|   |            |
|---|------------|
| n | an integer |
|---|------------|

**Returns**

The remainder

## 7.14 number\_theory.h

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

```

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 {
00036     T quotient = {};
00037     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 xl = 1;
00105     int x2 = 0;
00106     int yl = 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(xl, x2) = std::pair{x2, xl - q * x2};
00112         std::tie(yl, y2) = std::pair{y2, yl - q * y2};
00113     }
00114     return {.g = a, .x = xl, .y = yl};
00115 }
00116

```

```

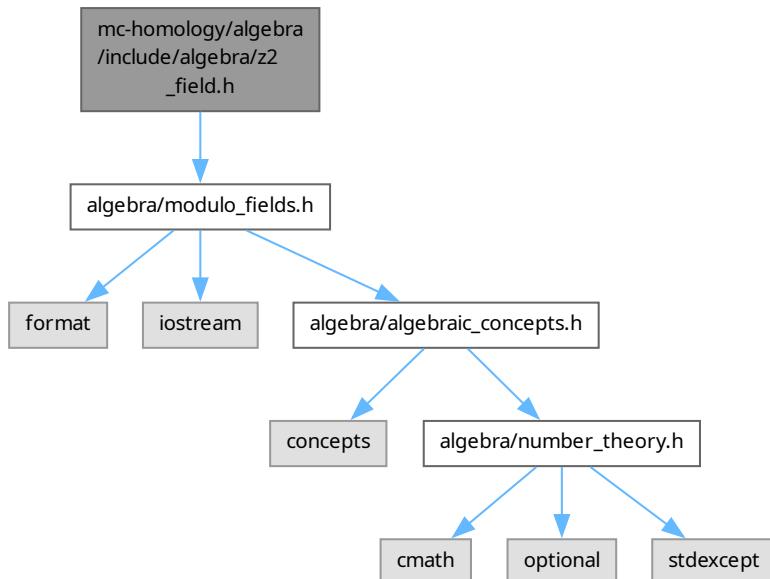
00125 constexpr std::optional<int> inverse_mod(int a, int n) noexcept {
00126     auto [g, x, __] = extended_gcd(a, n);
00127     if (g == 1) [likely] {
00128         return x;
00129     } else [unlikely] {
00130         return std::nullopt;
00131     }
00132 }
00133 } // namespace algebra

```

## 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

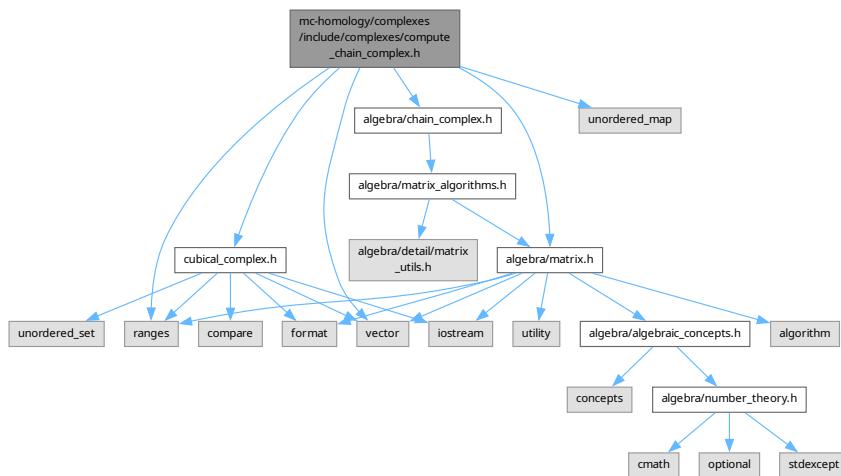
[Go to the documentation of this file.](#)

```
00001
00003
00004 #pragma once
00005
00006 #include "algebra/modulo_fields.h"
00007
00008 namespace algebra {
00009
00011 template<>
00012 class ZModP<2> {
00013 public:
00015     constexpr ZModP() = default;
00016
00018     constexpr ZModP(int n) noexcept : m_inner_representation(modulo(n, 2)) {}
00019
00021     constexpr static int p() noexcept {
00022         return 2;
00023     }
00024
00026     constexpr static ZModP zero() noexcept {
00027         return ZModP(0);
00028     }
00029
00031     constexpr static ZModP one() noexcept {
00032         return ZModP(1);
00033     }
00034
00036     constexpr explicit operator int() const noexcept {
00037         return m_inner_representation;
00038     }
00039
00041     constexpr bool operator==(ZModP const&) const = default;
00042
00044     constexpr ZModP& operator+=(ZModP rhs) noexcept {
00045         m_inner_representation ^= rhs.m_inner_representation;
00046         return *this;
00047     }
00048
00050     constexpr ZModP& operator-=(ZModP rhs) noexcept {
00051         m_inner_representation ^= rhs.m_inner_representation;
00052         return *this;
00053     }
00054
00056     constexpr ZModP operator+() const noexcept {
00057         return *this;
00058     }
00059
00061     constexpr ZModP operator-() const noexcept {
00062         return *this;
00063     }
00064
00066     constexpr ZModP& operator*=(ZModP rhs) noexcept {
00067         m_inner_representation &= rhs.m_inner_representation;
00068         return *this;
00069     }
00070
00074     constexpr int euclidean_function() const noexcept {
00075         return 1;
00076     }
00077
00079     constexpr ZModP& operator/=(ZModP rhs) {
00080         if (!rhs.m_inner_representation) [[unlikely]] {
00081             throw std::domain_error("Division by 0");
00082         }
00083         return *this;
00084     }
00085
00086 private:
00087     bool m_inner_representation = false;
00088 };
00089
00091 using Z2 = ZModP<2>;
00092
00093 } // namespace algebra
```

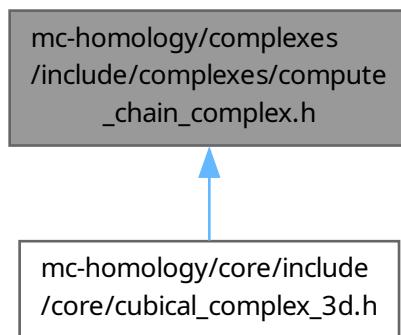
## 7.17 mc-homology/complexes/include/complexes/compute\_chain\_complex.h File Reference

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

```

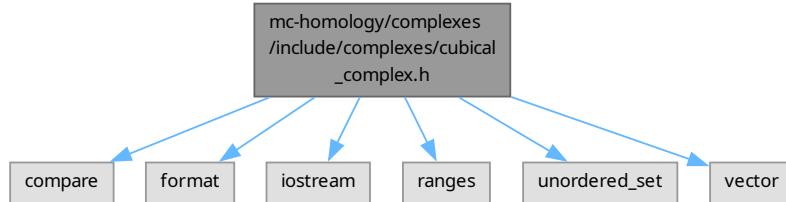
## 7.19 mc-homology/complexes/include/complexes/cubical\_complex.h File Reference

A file containing the CubicalComplex class.

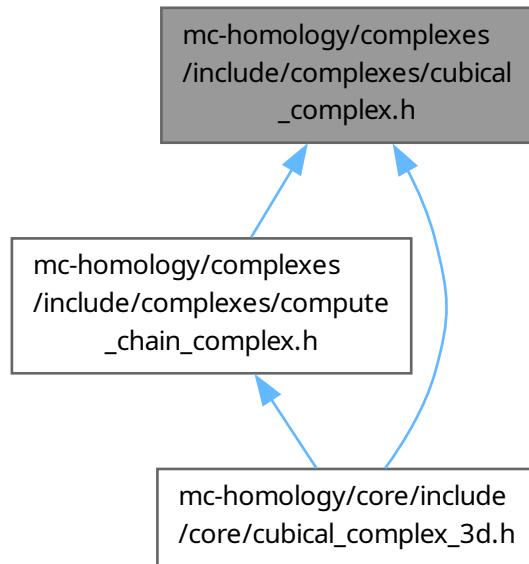
```

#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
00019 class BasicInterval {
00020 public:
00022     BasicInterval();
00023
00025     std::size_t hash() const;
00026
00028     bool operator==(BasicInterval const&) const;
00029
00031     int left() const;
00032
00034     int right() const;
00035
00038     bool is_trivial() const;
00039
00041     static BasicInterval point(int p);
00042
00046     static BasicInterval interval(int left);
00047
00048 private:
00049     int m_left = 0;
00050     bool m_full = false;
00051 };
00052
00054 std::ostream& operator<<(std::ostream& output, BasicInterval i);
00055
00061 class CubicalSimplex {
00062 public:
00069     CubicalSimplex(std::vector<BasicInterval> intervals);
00070
00072     std::size_t dimension() const;
00073
00075     std::size_t ambient_dimension() const;
00076
00078     std::size_t hash() const;
00079
00083     std::vector<CubicalSimplex> boundary() const;
00084
00086     bool operator==(CubicalSimplex const&) const;
00087
00089     std::vector<BasicInterval> const& intervals() const;
00090
00103     std::strong_ordering operator<=(CubicalSimplex const& simplex) const;
00104
00108     static CubicalSimplex point(int p);
00109
00113     static CubicalSimplex interval(int left);
00114
00124     friend CubicalSimplex
00125     product(CubicalSimplex const& s1, CubicalSimplex const& s2);
00126
00127 private:
00128     CubicalSimplex();
00129
00130     std::vector<BasicInterval> m_intervals = {};
00131     std::size_t m_dimension = 0;
00132 };
00133
00135 std::ostream& operator<<(std::ostream& output, CubicalSimplex const& s);
00136

```

```

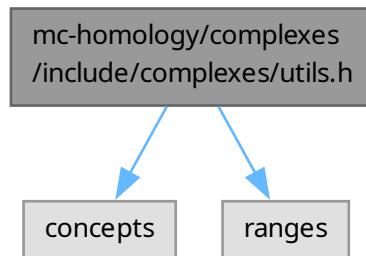
00141 class CubicalComplex {
00142 public:
00146     CubicalComplex();
00147
00158     bool add(CubicalSimplex simplex);
00159
00163     void add_recursive(CubicalSimplex simplex);
00164
00174     bool remove(CubicalSimplex const& simplex);
00175
00182     bool contains(CubicalSimplex const& simplex) const;
00183
00185     bool operator==(CubicalComplex const&) const;
00186
00191     std::vector<std::unordered_set<CubicalSimplex>> const& simplices() const;
00192
00194     std::size_t dimension() const;
00195
00197     std::size_t ambient_dimension() const;
00198
00199 private:
00200     void add_recursive_impl(CubicalSimplex simplex);
00201
00202     std::vector<std::unordered_set<CubicalSimplex>> m_simplices;
00203 };
00204
00205 } // namespace complexes
00206
00208 template<>
00209 struct std::hash<complexes::BasicInterval> {
00211     std::size_t operator()(complexes::BasicInterval const& i) const;
00212 };
00213
00215 template<>
00216 struct std::hash<complexes::CubicalSimplex> {
00218     std::size_t operator()(complexes::CubicalSimplex const& s) const;
00219 };
00220
00225 template<>
00226 struct std::formatter<complexes::BasicInterval>: public std::formatter<int> {
00227     template<class FmtContext>
00228     FmtContext::iterator
00229     format(complexes::BasicInterval i, FmtContext& ctx) const {
00230         std::format_to(ctx.out(), "[");
00231         std::formatter<int>::format(i.left(), ctx);
00232         if (!i.is_trivial()) {
00233             std::format_to(ctx.out(), ", ");
00234             std::formatter<int>::format(i.right(), ctx);
00235         }
00236         std::format_to(ctx.out(), "]");
00237         return ctx.out();
00238     }
00239 }
00240
00241 };
00242
00243
00244 template<>
00245 struct std::formatter<complexes::CubicalSimplex>:
00246     public std::formatter<complexes::BasicInterval> {
00247     template<class FmtContext>
00248     FmtContext::iterator
00249     format(complexes::CubicalSimplex const& s, FmtContext& ctx) const {
00250         namespaces vs = std::views;
00251         std::formatter<complexes::BasicInterval>::format(
00252             s.intervals().front(),
00253             ctx
00254         );
00255         for (auto i : s.intervals() | vs::drop(1)) {
00256             std::format_to(ctx.out(), "x");
00257             std::formatter<complexes::BasicInterval>::format(i, ctx);
00258         }
00259         return ctx.out();
00260     }
00261 };
00262
00263 }
```

## 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.](#)

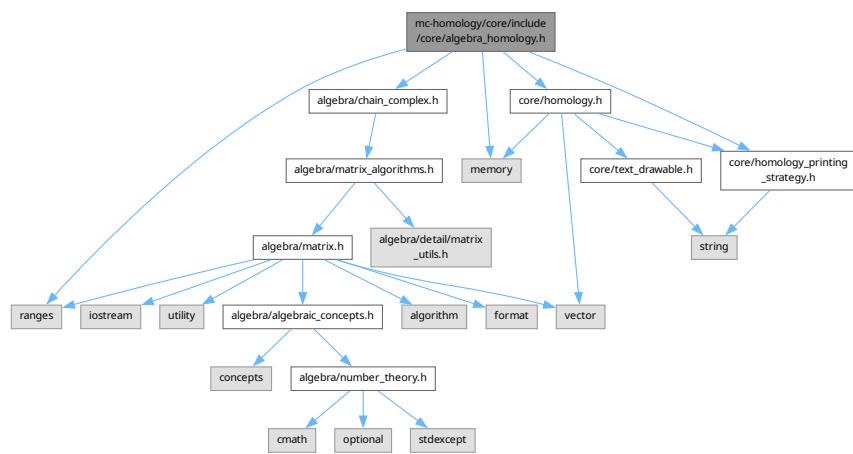
```

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

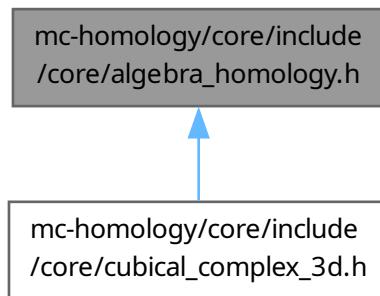
## 7.23 mc-homology/core/include/core/algebra\_homology.h File Reference

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:
00046     algebra::Homology<T> m_homology;
00047 };
00048
00049 } // namespace core

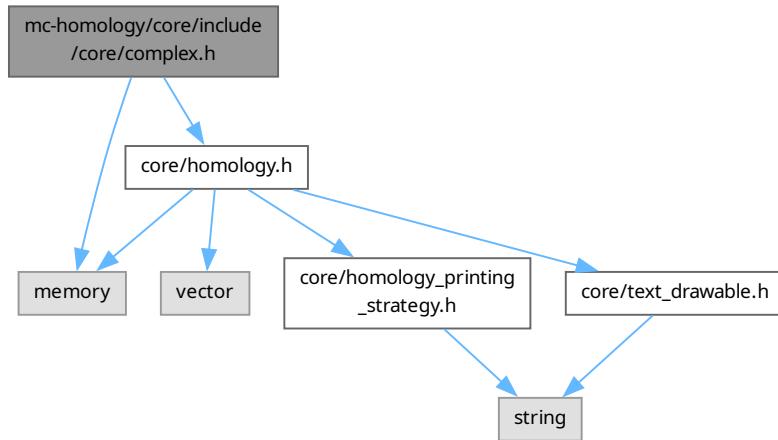
```

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

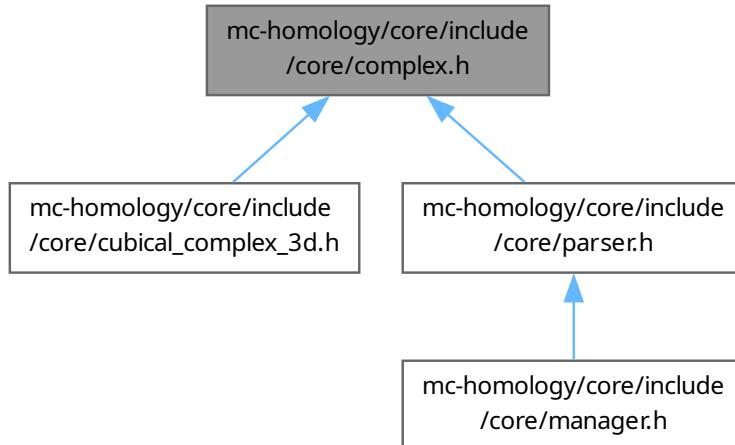
A file containing interface for the Complex class.

```
#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

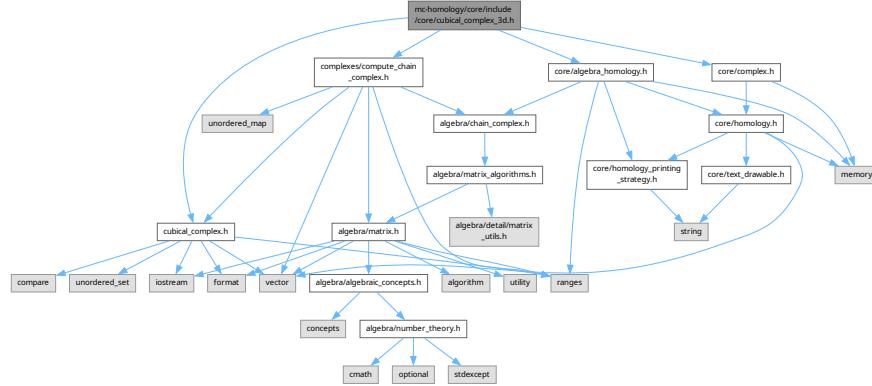
```

## 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.](#)

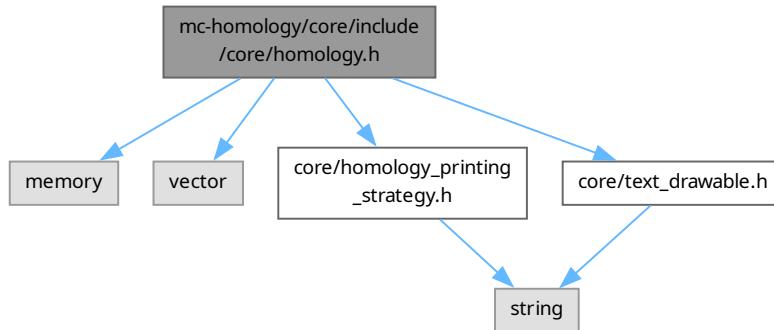
```
00001 #pragma once
00003 #include "complexes/compute_chain_complex.h"
00004 #include "complexes/cubical_complex.h"
00005 #include "core/algebra_homology.h"
00006 #include "core/complex.h"
00007
00008 namespace core {
00009
00010     class CubicalComplex3D: public Complex {
00011         public:
00012             void add_cube(int x, int y, int z);
00013             std::unique_ptr<Homology> z2_homology() const override;
00014             std::unique_ptr<Homology> z3_homology() const override;
00015             std::unique_ptr<Homology> z_homology() const override;
00016
00017             template<class T>
00018             std::unique_ptr<AlgebraHomology<T>> homology() const {
00019                 return std::make_unique<AlgebraHomology<T>>(
00020                     algebra::homology(complexes::compute_chain_complex<T>(m_inner))
00021             );
00022         }
00023
00024         void reduce() override;
00025
00026     private:
00027         complexes::CubicalComplex m_inner;
00028     };
00029 } // namespace core
```

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

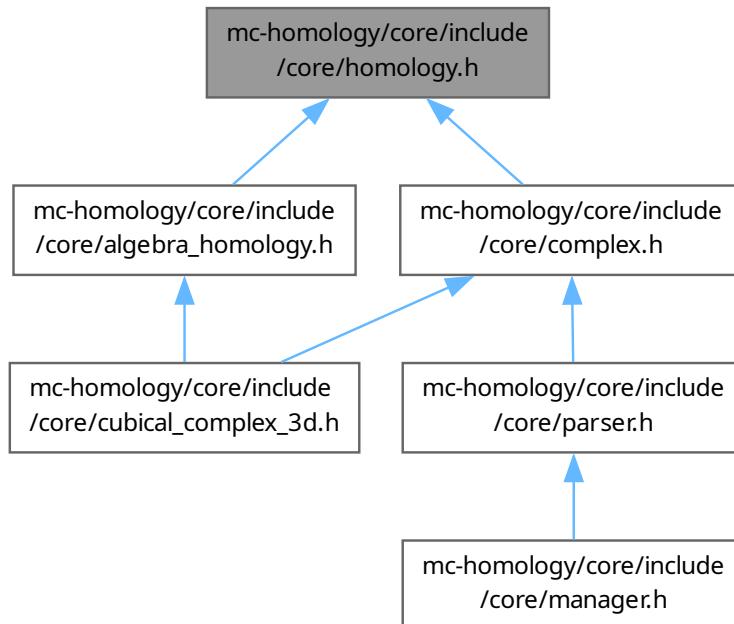
A file containing a Homology interface.

```
#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

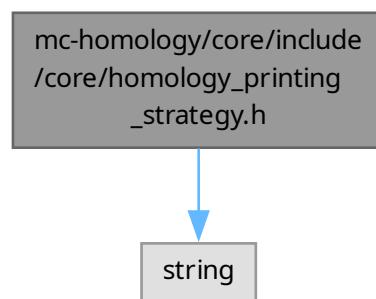
[Go to the documentation of this file.](#)

```
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:
00052     std::unique_ptr<HomologyPrintingStrategy> m_printing_strategy;
00053 };
00054
00055 } // namespace core
```

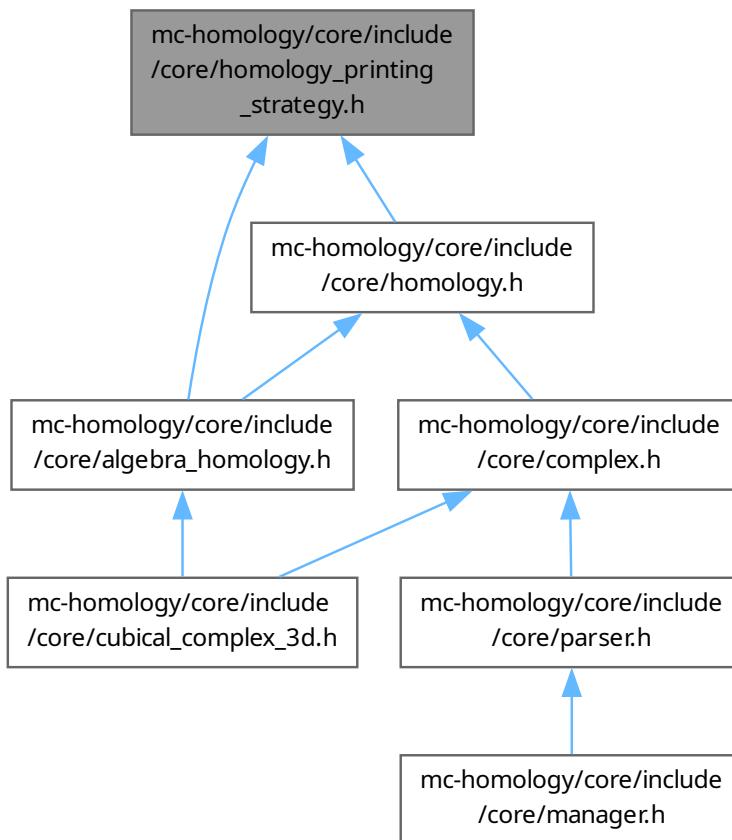
## 7.31 mc-homology/core/include/core/homology\_printing\_strategy.h File Reference

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.](#)

```

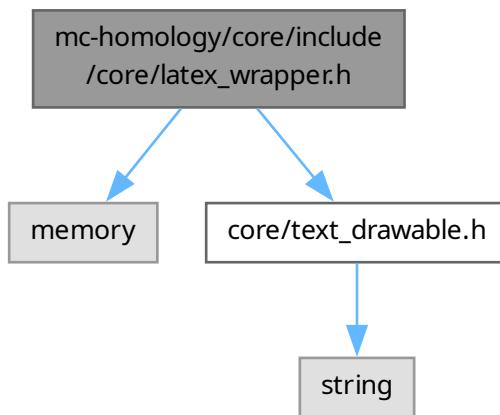
00001
00003 #pragma once
00004
00005 #include <string>
00006
00007 namespace core {
00008
00009 class Homology;
0010
0012 class HomologyPrintingStrategy {
0013 public:
0015     virtual std::string draw(Homology const& homology) const = 0;
0016
0018     virtual ~HomologyPrintingStrategy();
0019 };
0020
0022 class HomologyRawPrint: public HomologyPrintingStrategy {
0023 public:
0025     std::string draw(Homology const& homology) const override;
0026 };
0027
0032 class HomologyLatexPrint: public HomologyPrintingStrategy {
0033 public:
0041     HomologyLatexPrint(std::string ring_name, std::string homology_name = "H");
0042
0044     std::string draw(Homology const& homology) const override;
0045
0046 private:
0047     std::string m_ring_name;
0048     std::string m_homology_name;
0049 };
0050
0051 } // 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.](#)

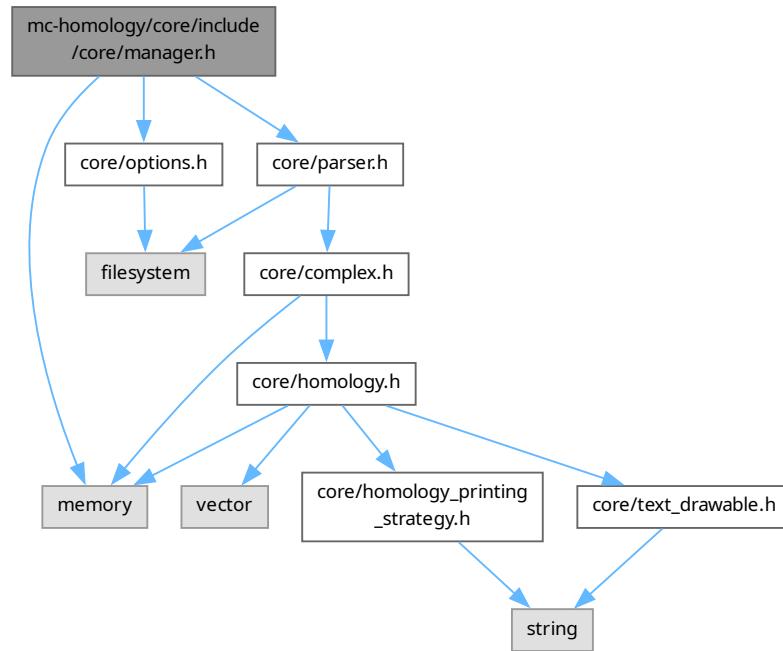
```
00001
00003 #pragma once
00004
00005 #include <memory>
00006
00007 #include "core/text_drawable.h"
00008
00009 namespace core {
00010
00013 class LatexWrapper: public TextDrawable {
00014 public:
00016     LatexWrapper(
00017         std::unique_ptr<TextDrawable> inner,
00018         std::string documentclass = "article"
00019     );
00020
00022     std::string text() const override;
00023
00024 private:
00025     std::unique_ptr<TextDrawable> m_inner;
00026     std::string m_documentclass;
00027 };
00028
00029 } // 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.](#)

```

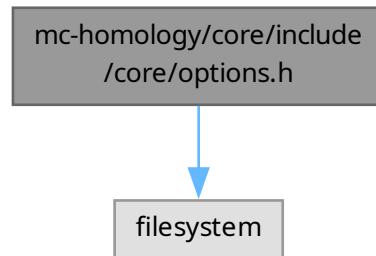
00001
00003 #pragma once
00004
00005 #include <memory>
00006
00007 #include "core/options.h"
00008 #include "core/parser.h"
00009
00010 namespace core {
00011
00013 class Manager {
00014 public:
00016     Manager(
00017         std::unique_ptr<Options> options,
00018         std::unique_ptr<MinecraftSavefileParser>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

```
00022     void set_options(std::unique_ptr<Options> options);
00023
00025     void set_parser(std::unique_ptr<MinecraftSavefileParser> parser);
00026
00028     int run();
00029
00030     private:
00031         std::unique_ptr<Options> m_options;
00032         std::unique_ptr<MinecraftSavefileParser> m_parser;
00033     };
00034
00035 }
00036
00037 } // namespace core
```

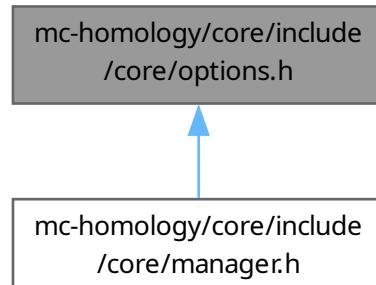
## 7.37 mc-homology/core/include/core/options.h File Reference

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:
00019     virtual std::filesystem::path filename() const = 0;
00020
00021     virtual std::pair<int, int> x_bounds() const = 0;
00022
00023     virtual std::pair<int, int> y_bounds() const = 0;
00024
00025     virtual std::pair<int, int> z_bounds() const = 0;
00026
00027     virtual HomologyChoice homology_to_compute() const = 0;
00028
00029     virtual bool latex() const = 0;
00030
00031     virtual bool help() const = 0;
00032
00033     virtual ~Options();
00034
00035 };
00036
00037
00038
00039
00040
00041
00042 };
00043
00044
00045
00046 class CommandlineOptions: public Options {
00047 public:
00048     CommandlineOptions(int argc, char** argv);
00049
00050     std::filesystem::path filename() const override;
00051
00052     std::pair<int, int> x_bounds() const override;
00053
00054     std::pair<int, int> y_bounds() const override;
00055
00056     std::pair<int, int> z_bounds() const override;
00057
00058     HomologyChoice homology_to_compute() const override;
00059
00060     bool latex() const override;
00061
00062     bool help() const override;
00063
00064
00065
00066
00067
00068
00069
00070

```

```

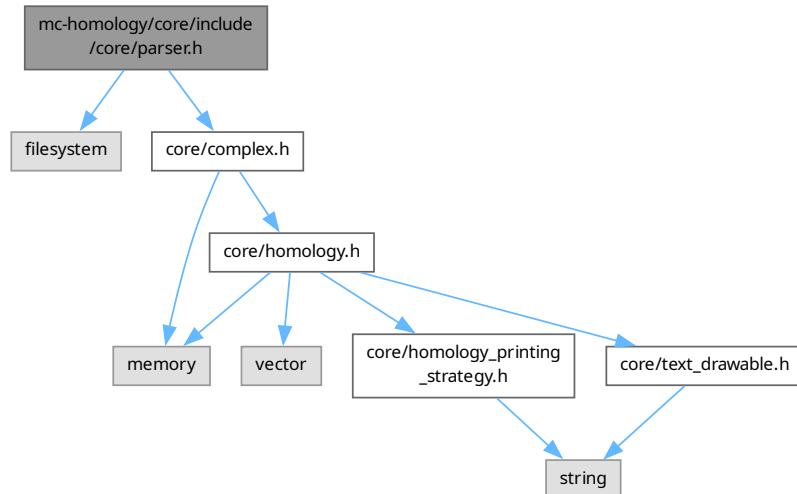
00071
00072 private:
00073     std::filesystem::path m_filename = "";
00074     std::pair<int, int> m_x_bounds = {0, 0};
00075     std::pair<int, int> m_y_bounds = {0, 0};
00076     std::pair<int, int> m_z_bounds = {0, 0};
00077     HomologyChoice m_homology_to_compute = HomologyChoice::Z2;
00078     bool m_latex = false;
00079     bool m_help = false;
00080 };
00081
00082 } // namespace core

```

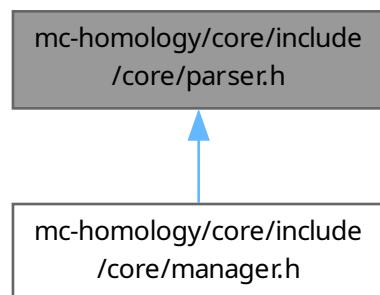
## 7.39 mc-homology/core/include/core/parser.h File Reference

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:
00026     virtual std::unique_ptr<Complex> parse(
00027         std::filesystem::path const& path,
00028         MinecraftCoordinates lower_corner,
00029         MinecraftCoordinates upper_corner
00030     ) = 0;
00031     virtual ~MinecraftSavefileParser();
00032 };
00033
00034 class MinecraftSavefileParser_mcSavefileParsers:
00035 public MinecraftSavefileParser {
00036 public:
00037     std::unique_ptr<Complex> parse(
00038         std::filesystem::path const& path,
00039         MinecraftCoordinates lower_corner,
00040         MinecraftCoordinates upper_corner
00041     ) override;
00042 };
00043
00044
00045 } // namespace core

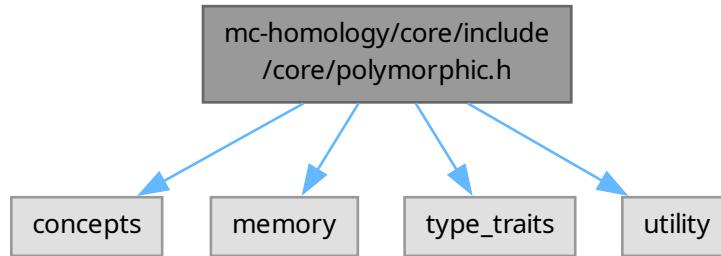
```

## 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.*

### 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
00040     template<class U, class... Args>
00041         requires std::derived_from<std::remove_cvref_t<U>, T>
00042             && std::copy_constructible<std::remove_cvref_t<U>>
00043             && std::constructible_from<std::remove_cvref_t<U>, Args...>
00044             && std::same_as<std::remove_cvref_t<U>, U>
00045     constexpr explicit Polymorphic(std::in_place_type_t<U>, Args&&... args) :
00046         m_object(
00047             std::make_unique<Model<std::remove_cvref_t<U>>(
00048                 std::forward<Args>(args)...
00049             )
00050         ) {}
00051
00052
00053     constexpr Polymorphic(Polymorphic const& other) :
00054         m_object(other.m_object->clone()) {}
00055
00056
00057     constexpr Polymorphic(Polymorphic&& other) noexcept :
00058         m_object(std::move(other.m_object)) {}
00059
00060
00061     constexpr Polymorphic& operator=(Polymorphic const& other) {
00062         m_object = other.m_object->clone();
00063         return *this;
00064     }
00065
00066
00067     constexpr Polymorphic& operator=(Polymorphic&& other) noexcept {
00068         m_object = std::move(other.m_object);
00069         return *this;
00070     }
00071
00072
00073     constexpr T const* operator->() const noexcept {
00074         return m_object->get_address();
00075     }
00076
00077
00078     constexpr T* operator->() noexcept {
00079         return m_object->get_address();
00080     }
00081
00082
00083     constexpr T const& operator*() const noexcept {
00084         return *m_object->get_address();
00085     }
00086
00087
00088     constexpr T& operator*() noexcept {
00089         return *m_object->get_address();
00090     }
00091
00092
00093     constexpr bool valueless_after_move() const noexcept {
00094         return m_object == nullptr;
00095     }
00096
00097
00098     constexpr void swap(Polymorphic& other) noexcept {
00099         using std::swap;
00100         swap(m_object, other.m_object);
00101     }
00102
00103
00104 private:
00105     struct Concept {
00106         constexpr virtual T const* get_address() const noexcept = 0;
00107         constexpr virtual T* get_address() noexcept = 0;
00108         constexpr virtual std::unique_ptr<Concept> clone() const = 0;
00109         constexpr virtual ~Concept() = default;
00110     };
00111
00112     template<class Object>
00113     struct Model: public Concept {
00114         template<class U = Object>
00115             constexpr Model(U&& value) : m_value(std::forward<U>(value)) {}
00116
00117         constexpr T const* get_address() const noexcept override {
00118             return std::addressof(m_value);
00119         }
00120
00121         constexpr T* get_address() noexcept override {
00122             return std::addressof(m_value);
00123         }
00124
00125         constexpr std::unique_ptr<Concept> clone() const override {
00126             return std::make_unique<Model<Object>>(m_value);
00127         }
00128
00129     private:
00130         Object m_value;
00131     };
00132
00133     std::unique_ptr<Concept> m_object;

```

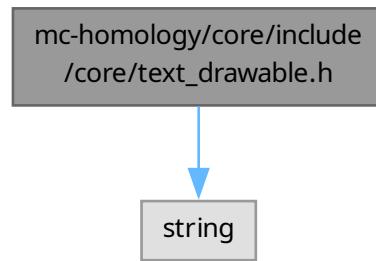
```
00134 };
00135
00136 #endif
00137
00138 } // namespace core
```

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

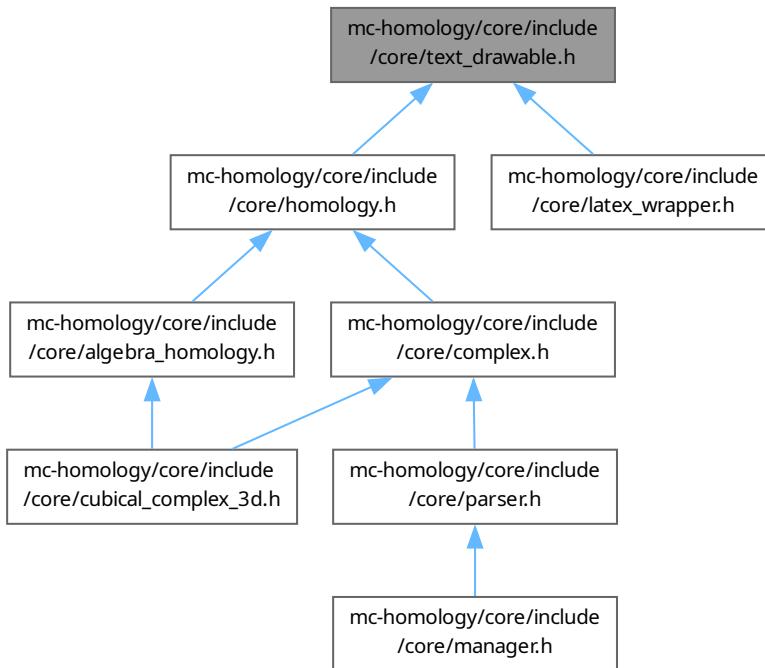
A file containing a `TextDrawable` interface, for classes that can be drawn on screen in the form of text.

```
#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, 30

add\_recursive  
complexes::CubicalComplex, 30

algebra::AdditiveGroup, 9

algebra::ChainComplex< T >, 20  
boundary, 22  
ChainComplex, 21

algebra::Commutative, 10

algebra::CommutativeRing, 10

algebra::DivResult< T >, 38

algebra::EuclideanDomain, 11

algebra::ExtendedGCDResult, 39

algebra::Field, 11

algebra::Group, 12

algebra::Homology< T >, 49  
torsion, 51

algebra::Integer, 61  
euclidean\_function, 63

algebra::Matrix< T >, 66  
at, 70  
Matrix, 70  
operator[], 71  
operators\*, 71

algebra::Ring, 13

algebra::RowEchelonFormResult< T >, 82

algebra::SkipCorrectnessCheckT, 83

algebra::SmithFormResult< T >, 83

algebra::ZModP< 2 >, 90  
euclidean\_function, 92

algebra::ZModP< P >, 87  
euclidean\_function, 89

at  
algebra::Matrix< T >, 70

betti\_numbers  
core::AlgebraHomology< T >, 18  
core::Homology, 54

boundary  
algebra::ChainComplex< T >, 22  
complexes::CubicalSimplex, 36

ChainComplex  
algebra::ChainComplex< T >, 21

complexes::BasicInterval, 19  
interval, 20

complexes::CubicalComplex, 29  
add, 30  
add\_recursive, 30  
contains, 31

CubicalComplex, 30  
remove, 31  
simplices, 31

complexes::CubicalSimplex, 35  
boundary, 36  
CubicalSimplex, 36  
interval, 36  
operator<=>, 37  
point, 37  
product, 37

compute\_chain\_complex  
compute\_chain\_complex.h, 125

compute\_chain\_complex.h  
compute\_chain\_complex, 125

contains  
complexes::CubicalComplex, 31

core::AlgebraHomology< T >, 15  
betti\_numbers, 18  
torsion, 18

core::CommandLineOptions, 22  
filename, 25  
help, 25  
homology\_to\_compute, 25  
latex, 25  
x\_bounds, 26  
y\_bounds, 26  
z\_bounds, 26

core::Complex, 27  
reduce, 28  
z2\_homology, 28  
z3\_homology, 29  
z\_homology, 29

core::CubicalComplex3D, 32  
reduce, 34  
z2\_homology, 34  
z3\_homology, 34  
z\_homology, 34

core::Homology, 51  
betti\_numbers, 54  
Homology, 54  
select\_strategy, 54  
text, 54  
torsion, 54

core::HomologyLatexPrint, 55  
draw, 58  
HomologyLatexPrint, 57

core::HomologyPrintingStrategy, 58  
draw, 59

core::HomologyRawPrint, 60

draw, 61  
 core::LatexWrapper, 63  
   text, 65  
 core::Manager, 65  
   run, 66  
 core::MinecraftCoordinates, 72  
 core::MinecraftSavefileParser, 73  
   parse, 74  
 core::MinecraftSavefileParser\_mcSavefileParsers, 74  
   parse, 76  
 core::Options, 76  
   filename, 79  
   help, 79  
   homology\_to\_compute, 79  
   latex, 79  
   x\_bounds, 79  
   y\_bounds, 79  
   z\_bounds, 79  
 core::Polymorphic< T >, 80  
 core::TextDrawable, 84  
   text, 86  
 CubicalComplex  
   complexes::CubicalComplex, 30  
 CubicalSimplex  
   complexes::CubicalSimplex, 36  
  
 divide  
   integer.h, 100  
   number\_theory.h, 119  
 draw  
   core::HomologyLatexPrint, 58  
   core::HomologyPrintingStrategy, 59  
   core::HomologyRawPrint, 61  
  
 euclidean\_function  
   algebra::Integer, 63  
   algebra::ZModP< 2 >, 92  
   algebra::ZModP< P >, 89  
 extended\_gcd  
   number\_theory.h, 119  
  
 filename  
   core::CommandlineOptions, 25  
   core::Options, 79  
 format  
   std::formatter< algebra::Integer >, 41  
   std::formatter< algebra::Matrix< T > >, 42  
   std::formatter< complexes::BasicInterval >, 45  
   std::formatter< complexes::CubicalSimplex >, 48  
  
 help  
   core::CommandlineOptions, 25  
   core::Options, 79  
 Homology  
   core::Homology, 54  
 homology\_to\_compute  
   core::CommandlineOptions, 25  
   core::Options, 79  
 HomologyLatexPrint  
  
 core::HomologyLatexPrint, 57  
  
 integer.h  
   divide, 100  
   modulo, 100  
 interval  
   complexes::BasicInterval, 20  
   complexes::CubicalSimplex, 36  
 inverse\_mod  
   number\_theory.h, 119  
 is\_prime  
   number\_theory.h, 120  
  
 latex  
   core::CommandlineOptions, 25  
   core::Options, 79  
  
 Matrix  
   algebra::Matrix< T >, 70  
 matrix.h  
   operator\*, 104  
 matrix\_algorithms.h  
   row\_echelon\_form, 110  
   smith\_form, 111  
 mc-homology/algebra/include/algebra/algebraic\_concepts.h, 93, 95  
 mc-homology/algebra/include/algebra/chain\_complex.h, 95, 97  
 mc-homology/algebra/include/algebra/integer.h, 99, 101  
 mc-homology/algebra/include/algebra/matrix.h, 102, 104  
 mc-homology/algebra/include/algebra/matrix\_algorithms.h, 108, 112  
 mc-homology/algebra/include/algebra/modulo\_fields.h, 113, 115  
 mc-homology/algebra/include/algebra/number\_theory.h, 116, 121  
 mc-homology/algebra/include/algebra/z2\_field.h, 122, 123  
 mc-homology/complexes/include/complexes/compute\_chain\_complex.h, 124, 125  
 mc-homology/complexes/include/complexes/cubical\_complex.h, 126, 128  
 mc-homology/complexes/include/complexes/utils.h, 129, 130  
 mc-homology/core/include/core/algebra\_homology.h, 131, 132  
 mc-homology/core/include/core/complex.h, 132, 134  
 mc-homology/core/include/core/cubical\_complex\_3d.h, 134, 135  
 mc-homology/core/include/core/homology.h, 135, 137  
 mc-homology/core/include/core/homology\_printing\_strategy.h, 137, 139  
 mc-homology/core/include/core/latex\_wrapper.h, 139, 140  
 mc-homology/core/include/core/manager.h, 140, 141  
 mc-homology/core/include/core/options.h, 142, 143  
 mc-homology/core/include/core/parser.h, 144, 145

mc-homology/core/include/core/polymorphic.h, 145, std::hash< complexes::CubicalSimplex >, 49  
146

mc-homology/core/include/core/text\_drawable.h, 148, text  
149

modulo  
integer.h, 100  
number\_theory.h, 120

number\_theory.h  
divide, 119  
extended\_gcd, 119  
inverse\_mod, 119  
is\_prime, 120  
modulo, 120

operator<=>  
complexes::CubicalSimplex, 37

operator[]  
algebra::Matrix< T >, 71

operator\*  
matrix.h, 104

operator\*=  
algebra::Matrix< T >, 71

parse  
core::MinecraftSavefileParser, 74  
core::MinecraftSavefileParser\_mcSavefileParsers,  
76

point  
complexes::CubicalSimplex, 37

product  
complexes::CubicalSimplex, 37

reduce  
core::Complex, 28  
core::CubicalComplex3D, 34

remove  
complexes::CubicalComplex, 31

row\_echelon\_form  
matrix\_algorithms.h, 110

run  
core::Manager, 66

select\_strategy  
core::Homology, 54

simplices  
complexes::CubicalComplex, 31

smith\_form  
matrix\_algorithms.h, 111

std::formatter< algebra::Integer >, 40  
format, 41

std::formatter< algebra::Matrix< T > >, 41  
format, 42

std::formatter< algebra::ZModP< P > >, 43

std::formatter< complexes::BasicInterval >, 44  
format, 45

std::formatter< complexes::CubicalSimplex >, 46  
format, 48

std::hash< complexes::BasicInterval >, 48