OOP Pažymio skaičiavimas - Vector klasė

Generated by Doxygen 1.10.0

1 Hierarchical Index **1**

1.1 Class Hierarchy 1

2 Class Index **3**

2.1 Class List 3

3 File Index **5**

3.1 File List 5

4 Class Documentation **7**

4.1 Stud Class Reference 7

4.2 Vector< T > Class Template Reference 8

4.3 Zmogus Class Reference 10

5 File Documentation **11**

5.1 funkcijos.h 11

5.2 student.h 11

5.3 vector.h 13

5.4 zmogus.h 17

Index **19**

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Vector< T > 8
Vector< int > 8
Zmogus 10
Stud 7

Chapter 2

Class Index

2.1 Class List

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

Stud 7
Vector< T > 8
Zmogus 10

Chapter 3

File Index

3.1 File List

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

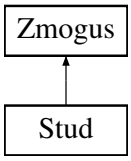
vector/funkcijos.h 11
vector/student.h 11
vector/vector.h 13
vector/zmogus.h 17

Chapter 4

Class Documentation

4.1 Stud Class Reference

Inheritance diagram for Stud:



Public Member Functions

- **Stud** (Vector< int > &namuDarbai, string &vardas, string &pavarde, int egzaminas, double gal, int ndcount)
- **Stud** (const Stud &other)
- Stud & **operator=** (const Stud &other)
- **Stud** (Stud &&other) noexcept
- Stud & **operator=** (Stud &&other) noexcept
- Vector< int > **getNamuDarbai** () const
- string **getVardas** () const
- string **getPavarde** () const
- int **getEgzaminas** () const
- double **getGal** () const
- int **getNdcount** () const
- void **setNamuDarbai** (const Vector< int > &namuDarbai)
- void **setVardas** (const string &vardas)
- void **setPavarde** (const string &pavarde)
- void **setEgzaminas** (int egzaminas)
- void **setGal** (double gal)
- void **setNdcount** (int ndcount)
- void **addND** (int namuDarbai)
- void **clearND** ()

Public Member Functions inherited from Zmogus

- **Zmogus** (const std::string &vardas, const std::string &pavarde)

Private Attributes

- int **egzaminas_**
- int **ndcount_**
- double **gal_**
- Vector< int > **namuDarbai_**

Friends

- std::istream & **operator>>** (std::istream &is, Stud &stud)
- std::ostream & **operator<<** (std::ostream &os, const Stud &stud)

Additional Inherited Members

Public Attributes inherited from Zmogus

- std::string **vardas_**
- std::string **pavarde_**

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

- vector/student.h

4.2 Vector< T > Class Template Reference

Public Types

- using **size_type** = size_t
- using **value_type** = T
- using **reference** = T&
- using **const_reference** = const T&
- using **iterator** = T*
- using **const_iterator** = const T*
- using **pointer** = T*
- using **const_pointer** = const T*
- using **reverse_iterator** = std::reverse_iterator<iterator>
- using **const_reverse_iterator** = std::reverse_iterator<const_iterator>

Public Member Functions

- **Vector** (size_type count, const value_type &value=value_type())
- **Vector** (const Vector &other)
- **Vector** (Vector &&other) noexcept
- Vector & **operator=** (const Vector &other)
- Vector & **operator=** (Vector &&other) noexcept
- **Vector** (std::initializer_list< T > init)
- reference **operator[ ]** (size_type pos)
- const_reference **operator[ ]** (size_type pos) const
- reference **at** (size_type pos)
- const_reference **at** (size_type pos) const
- reference **front** ()
- const_reference **front** () const
- reference **back** ()
- const_reference **back** () const
- pointer **data** () noexcept
- const_pointer **data** () const noexcept
- size_type **size** () const noexcept
- size_type **capacity** () const noexcept
- bool **empty** () const noexcept
- size_type **max_size** () const noexcept
- void **reserve** (size_type new_cap)
- void **shrink_to_fit** ()
- void **resize** (size_type count, value_type value=value_type())
- void **push_back** (const value_type &value)
- void **pop_back** ()
- void **clear** () noexcept
- iterator **insert** (const_iterator pos, const value_type &value)
- iterator **erase** (const_iterator pos)
- iterator **erase** (iterator pos)
- iterator **erase** (iterator first, iterator last)
- void **swap** (Vector &other) noexcept
- iterator **begin** () noexcept
- const_iterator **begin** () const noexcept
- iterator **end** () noexcept
- const_iterator **end** () const noexcept
- reverse_iterator **rbegin** () noexcept
- const_reverse_iterator **rbegin** () const noexcept
- reverse_iterator **rend** () noexcept
- const_reverse_iterator **rend** () const noexcept

Private Attributes

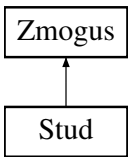
- size_type **_size**
- size_type **_capacity**
- value_type * **_data**

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

- vector/vector.h

4.3 Zmogus Class Reference

Inheritance diagram for Zmogus:



Public Member Functions

- **Zmogus** (const std::string &vardas, const std::string &pavarde)

Public Attributes

- std::string **vardas_**
- std::string **pavarde_**

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

- vector/zmogus.h

Chapter 5

File Documentation

5.1 funkcijos.h

```
00001 #ifndef FUNKCIJOS_H
00002 #define FUNKCIJOS_H
00003 #include "vector.h"
00004 #include "student.h"
00005
00006 bool rusiavimas(const Stud& a, const Stud& b, char metPas);
00007 double vidurkis(Stud&);
00008 double mediana(Stud&);
00009 void isFailo(const std::string& failPav, Vector<Stud>& studentai, int dyd);
00010 void failuGen(const std::string& failPav, int numRecords);
00011 void rusiavimasGen(const std::string& failPav, Vector<Stud>& studentai, Vector<Stud>& luzeriukai);
00012 void ranka(Vector<Stud>& studentai);
00013 bool rusiavimasV(const Stud& a, const Stud& b);
00014 bool rusiavimasP(const Stud& a, const Stud& b);
00015 bool rusiavimasG(const Stud& a, const Stud& b);
00016 void testai();
00017 void isvedimas_i_ekrana(const Vector<Stud>& luzeriukai, const Vector<Stud>& studentai);
00018 void isvedimas_i_faila(const Vector<Stud>& luzeriukai, const Vector<Stud>& studentai, const string&
      failPav);
00019 void VectorTest();
00020 void insertTest();
00021 void vectorVsVector();
00022
00023
00024
00025 #endif
```

5.2 student.h

```
00001 #ifndef STUDENT_H
00002 #define STUDENT_H
00003 #include "vector.h"
00004 #include <iostream>
00005 #include <string>
00006 #include <iomanip>
00007 #include "zmogus.h"
00008
00009 using namespace std;
00010
00011 class Stud : public Zmogus{
00012 private:
00013   //  string vardas_, pavarde_;
00014     int egzaminas_, ndcount_;
00015     double gal_;
00016     Vector<int> namuDarbai_;
00017 public:
00018     Stud() : egzaminas_(0), gal_(0), ndcount_(0) { }  // default konstruktorius
00019     Stud(Vector <int>& namuDarbai, string& vardas, string& pavarde, int egzaminas, double gal, int
      ndcount)
00020         : namuDarbai_(namuDarbai), Zmogus(vardas, pavarde), egzaminas_(egzaminas), gal_(gal),
      ndcount_(ndcount) {}
00021
00022     ~Stud() {namuDarbai_.clear(); vardas_.clear(), pavarde_.clear();}
00023
```

```
00024     // Copy constructor
00025     Stud(const Stud& other)
00026         : Zmogus(other.vardas_, other.pavarde_),  egzaminas_(other.egzaminas_),  gal_(other.gal_),
      namuDarbai_(other.namuDarbai_), ndcount_(other.ndcount_) {}
00027
00028     // Copy assignment operator
00029     Stud& operator=(const Stud& other) {
00030         if (this != &other) {
00031             vardas_ = other.vardas_;
00032             pavarde_ = other.pavarde_;
00033             egzaminas_ = other.egzaminas_;
00034             gal_ = other.gal_;
00035             namuDarbai_ = other.namuDarbai_;
00036             ndcount_ = other.ndcount_;
00037         }
00038         return *this;
00039     }
00040
00041     // Move constructor
00042     Stud(Stud&& other) noexcept
00043         : Zmogus(move(other.vardas_), move(other.pavarde_)), egzaminas_(other.egzaminas_),
      gal_(other.gal_), namuDarbai_(move(other.namuDarbai_)), ndcount_(move(other.ndcount_)){
00044         other.vardas_.clear(); other.pavarde_.clear(); other.ndcount_ = 0;  other.egzaminas_ = 0;
      other.gal_ = 0; other.namuDarbai_.clear();}// clearint
00045
00046
00047     // Move assignment operator
00048     Stud& operator=(Stud&& other) noexcept {
00049         if (this != &other) {
00050             vardas_ = move(other.vardas_);
00051             pavarde_ = move(other.pavarde_);
00052             egzaminas_ = move(other.egzaminas_);
00053             gal_ = move(other.gal_);
00054             namuDarbai_ = move(other.namuDarbai_);
00055             ndcount_ = move(other.ndcount_);
00056         }
00057         return *this;
00058     }
00059
00060
00061
00062     // getteriai
00063     Vector <int> getNamuDarbai() const {return namuDarbai_;}
00064     string getVardas() const {return vardas_;}
00065     string getPavarde() const {return pavarde_;}
00066     int getEgzaminas() const {return egzaminas_;}
00067     double getGal() const {return gal_;}
00068     int getNdcount() const {return ndcount_;}
00069
00070     //setteriai
00071     void setNamuDarbai(const Vector <int>& namuDarbai) {namuDarbai_ = namuDarbai;}
00072     void setVardas(const string& vardas) {vardas_ = vardas;}
00073     void setPavarde(const string& pavarde) {pavarde_ = pavarde;}
00074     void setEgzaminas (int egzaminas) {egzaminas_ = egzaminas;}
00075     void setGal (double gal) {gal_ = gal;}
00076     void setNdcount (int ndcount) {ndcount_ = ndcount;}
00077
00078     //kiti
00079     void addND(int namuDarbai) { namuDarbai_.push_back(namuDarbai); }
00080     void clearND() { namuDarbai_.clear(); }
00081
00082   // Input Operator
00083 friend std::istream& operator»(std::istream& is, Stud& stud) {
00084     is » stud.vardas_ » stud.pavarde_;
00085     stud.namuDarbai_.clear();
00086     int balas;
00087     for (int i = 0; i < stud.getNdcount(); ++i) {
00088         is » balas;
00089         stud.namuDarbai_.push_back(balas);
00090     }
00091
00092     is » stud.egzaminas_;
00093     return is;
00094 }
00095
00096 // Output Operator
00097 friend std::ostream& operator«(std::ostream& os, const Stud& stud) {
00098     os « stud.vardas_ « setw(20) « stud.pavarde_ « setw(20) « stud.gal_ « setw(20) « "\n";
00099     return os;
00100 }
00101
00102
00103
00104 };
00105
00106
00107 #endif
```

5.3 vector.h

```
00001 #pragma once
00002
00003
00004 #include <iostream>
00005 #include <stdexcept>
00006 #include <algorithm>
00007 #include <iterator>
00008 #include <limits>
00009 #include <initializer_list>
00010
00011
00012 template <typename T>
00013 class Vector {
00014     public:
00015     using size_type = size_t;
00016     using value_type = T;
00017     using reference  = T&;
00018     using const_reference = const T&;
00019     using iterator = T*;
00020     using const_iterator = const T*;
00021     using pointer = T*;
00022     using const_pointer = const T*;
00023     using reverse_iterator = std::reverse_iterator<iterator>;
00024     using const_reverse_iterator = std::reverse_iterator<const_iterator>;
00025
00026
00027      //member functions:
00028
00029     //default konstruktorius
00030     Vector() : _size(0), _capacity(0), _data(nullptr) {}
00031
00032     //fill konstruktorius
00033     Vector(size_type count, const value_type& value = value_type())
00034     : _size(count), _capacity(count), _data(new value_type[count]) {
00035     std::fill(_data, _data + _size, value);
00036     }
00037
00038     //copy konstruktorius
00039     Vector(const Vector& other)
00040     : _size(other._size), _capacity(other._capacity), _data(new value_type[other._capacity]) {
00041     std::copy(other._data, other._data + _size, _data);
00042     }
00043
00044     //move konstruktorius
00045     Vector(Vector&& other) noexcept
00046         : _size(other._size), _capacity(other._capacity), _data(other._data) {
00047         other._size = 0;
00048         other._capacity = 0;
00049         other._data = nullptr;
00050     }
00051
00052     //copy assignment operatorius
00053     Vector& operator=(const Vector& other) {
00054         if (this != &other) {
00055             value_type* new_data = new value_type[other._capacity];
00056             std::copy(other._data, other._data + other._size, new_data);
00057             delete[] _data;
00058             _data = new_data;
00059             _size = other._size;
00060             _capacity = other._capacity;
00061         }
00062         return *this;
00063     }
00064
00065     //move assignment operatorius
00066     Vector& operator=(Vector&& other) noexcept {
00067         if (this != &other) {
00068             delete[] _data;
00069             _data = other._data;
00070             _size = other._size;
00071             _capacity = other._capacity;
00072             other._data = nullptr;
00073             other._size = 0;
00074             other._capacity = 0;
00075         }
00076         return *this;
00077     }
00078
00079
00080     //destruktorius
00081         ~Vector() { delete[] _data;}
00082
00083
00084     // Initializer list constructor
00085     Vector(std::initializer_list<T> init)
```

```
00086     : _size(init.size()), _capacity(init.size()), _data(new T[init.size()]) {
00087         std::copy(init.begin(), init.end(), _data);
00088     }
00089 
00090 
00091     // Element access - grazina elemento reference nurodytoj lokacijoj
00092     reference operator[](size_type pos) {
00093         return _data[pos];
00094     }
00095 
00096     //const element access
00097     const_reference operator[](size_type pos) const {
00098         return _data[pos];
00099     }
00100     //su bounds check
00101     reference at(size_type pos) {
00102         if (pos >= _size) {
00103             throw std::out_of_range("Vector::at: index out of range");
00104         }
00105         return _data[pos];
00106     }
00107 
00108     const_reference at(size_type pos) const {
00109         if (pos >= _size) {
00110             throw std::out_of_range("Vector::at: index out of range");
00111         }
00112         return _data[pos];
00113     }
00114 
00115      // pirmo elemento reference
00116     reference front() {
00117         return _data[0];
00118     }
00119 
00120      // const pirmo elemento reference
00121     const_reference front() const {
00122         return _data[0];
00123     }
00124 
00125     // paskutinio elemento reference
00126     reference back() {
00127         return _data[_size - 1];
00128     }
00129 
00130     // const paskutinio elemento reference
00131     const_reference back() const {
00132         return _data[_size - 1];
00133     }
00134 
00135     // pointeris i pirma vektoriaus internal array elementa
00136     pointer data() noexcept {
00137         return _data;
00138     }
00139 
00140     // const pointeris i pirma vektoriaus internal array elementa
00141     const_pointer data() const noexcept {
00142         return _data;
00143     }
00144 
00145     //capacity
00146     // dydis - grazina elementu sk
00147     size_type size() const noexcept {
00148         return _size;
00149     }
00150 
00151     //talpa - grazina kiek elementu telpa dabar
00152     size_type capacity() const noexcept {
00153         return _capacity;
00154     }
00155 
00156     //ar empty
00157     bool empty() const noexcept {
00158         return _size == 0;
00159     }
00160 
00161     //max sk elementu
00162     size_type max_size() const noexcept {
00163     return std::numeric_limits<size_type>::max();
00164     }
00165 
00166     //rezervuoja didesne talpa
00167     void reserve(size_type new_cap) {
00168         if (new_cap > _capacity) {
00169             value_type* new_data = new value_type[new_cap];
00170             std::copy(_data, _data + _size, new_data);
00171             delete[] _data;
00172             _data = new_data;
```

```
00173             _capacity = new_cap;
00174         }
00175     }
00176
00177     //freeina nenaudojama atminti
00178      void shrink_to_fit() {
00179         if (_capacity > _size) {
00180             value_type* new_data = new value_type[_size];
00181             std::copy(_data, _data + _size, new_data);
00182             delete[] _data;
00183             _data = new_data;
00184             _capacity = _size;
00185         }
00186     }
00187
00188     //padidina talpa
00189     void resize(size_type count, value_type value = value_type()) {
00190         if (count > _capacity) {
00191             reserve(count);
00192         }
00193         if (count > _size) {
00194             std::fill(_data + _size, _data + count, value);
00195         }
00196         _size = count;
00197     }
00198
00199     //modifiers:
00200
00201     //push
00202     void push_back(const value_type& value) {
00203         if (_size == _capacity) {
00204             reserve(_capacity == 0 ? 1 : 2 * _capacity);
00205         }
00206         _data[_size++] = value;
00207     }
00208
00209     //pop
00210     void pop_back() {
00211         if (_size > 0) {
00212             --_size;
00213         }
00214     }
00215
00216     //clear
00217      void clear() noexcept {
00218         _size = 0;
00219     }
00220
00221     //insertina i norima vieta
00222      iterator insert(const_iterator pos, const value_type& value) {
00223         size_type index = pos - begin();
00224         if (_size == _capacity) {
00225             reserve(_capacity == 0 ? 1 : 2 * _capacity);
00226         }
00227         std::copy_backward(begin() + index, end(), end() + 1);
00228         _data[index] = value;
00229         ++_size;
00230         return begin() + index;
00231     }
00232
00233     //erasina specifinej vietoj
00234       iterator erase(const_iterator pos) {
00235         size_type index = pos - begin();
00236         std::copy(begin() + index + 1, end(), begin() + index);
00237         --_size;
00238         return begin() + index;
00239     }
00240
00241      iterator erase(iterator pos) {
00242         if (pos < _data || pos >= _data + size()) return end();
00243         std::move(pos + 1, _data + size(), pos);
00244         --_size;
00245         return pos;
00246     }
00247
00248     iterator erase(iterator first, iterator last) {
00249         if (first < _data || last > _data + size() || first > last) return end();
00250         auto new_end = std::move(last, _data + size(), first);
00251         _size -= std::distance(first, last);
00252         return first;
00253     }
00254
00255     //swapina su kitu vectorium
00256      void swap(Vector& other) noexcept {
00257         std::swap(_data, other._data);
00258         std::swap(_size, other._size);
00259         std::swap(_capacity, other._capacity);
```

```
00260     }
00261
00262      // Iteratoriai:
00263
00264      //grazina pradzios it
00265     iterator begin() noexcept {
00266         return _data;
00267     }
00268
00269     //grazina const pradzios it
00270     const_iterator begin() const noexcept {
00271         return _data;
00272     }
00273
00274     //grazina pabaigos it
00275     iterator end() noexcept {
00276         return _data + _size;
00277     }
00278
00279     //grazina pabaigos const it
00280     const_iterator end() const noexcept {
00281         return _data + _size;
00282     }
00283
00284     // reverse iteratorius pradzios
00285     reverse_iterator rbegin() noexcept {
00286         return reverse_iterator(end());
00287     }
00288     // const reverse iteratorius pradzios
00289      const_reverse_iterator rbegin() const noexcept {
00290         return const_reverse_iterator(end());
00291     }
00292
00293     // reverse iteratorius pabaigos
00294      reverse_iterator rend() noexcept {
00295         return reverse_iterator(begin());
00296     }
00297
00298     // const reverse iteratorius pabaigos
00299     const_reverse_iterator rend() const noexcept {
00300         return const_reverse_iterator(begin());
00301     }
00302
00303
00304
00305
00306     private:
00307     size_type _size;
00308     size_type _capacity;
00309     value_type* _data;
00310
00311 };
00312
00313 // Non-member
00314
00315 // tikrina ar lygus
00316 template <typename T>
00317 bool operator==(const Vector<T>& lhs, const Vector<T>& rhs) {
00318     return lhs.size() == rhs.size() && std::equal(lhs.begin(), lhs.end(), rhs.begin());
00319 }
00320
00321 //tikrina ar nelygus
00322 template <typename T>
00323 bool operator!=(const Vector<T>& lhs, const Vector<T>& rhs) {
00324     return !(lhs == rhs);
00325 }
00326
00327 // std::swap sitam vektorius
00328 template <typename T>
00329 void swap(Vector<T>& lhs, Vector<T>& rhs) noexcept {
00330     lhs.swap(rhs);
00331 }
00332
00333 // erase
00334 template <typename T, typename UnaryPredicate>
00335 typename Vector<T>::iterator erase_if(Vector<T>& vec, UnaryPredicate p) {
00336     auto it = std::remove_if(vec.begin(), vec.end(), p);
00337     auto res = vec.end();
00338     if (it != vec.end()) {
00339         res = vec.erase(it, vec.end());
00340     }
00341     return res;
00342 }
00343
00344 // output
00345 template <typename T>
00346 std::ostream& operator<<(std::ostream& os, const Vector<T>& vec) {
```

```
00347     os « "[";
00348     for (size_t i = 0; i < vec.size(); ++i) {
00349         os « vec[i];
00350         if (i != vec.size() - 1) {
00351             os « ", ";
00352         }
00353     }
00354     os « "]";
00355     return os;
00356 }
```

5.4 zmogus.h

```
00001 #ifndef ZMOGUS_H
00002 #define ZMOGUS_H
00003 #include <string>
00004
00005 class Zmogus {
00006
00007     public:
00008     std::string vardas_;
00009     std::string pavarde_;
00010
00011     Zmogus() : vardas_(""), pavarde_("") {}
00012     Zmogus(const std::string& vardas, const std::string& pavarde)
00013     : vardas_(vardas), pavarde_(pavarde) {}
00014
00015 };
00016
00017 #endif
```

Index

Stud, 7

Vector< T >, 8
vector/funkcijos.h, 11
vector/student.h, 11
vector/vector.h, 13
vector/zmogus.h, 17

Zmogus, 10