Dokumentacija

v3.0

Generated by Doxygen 1.13.2

Chapter 1

README

v3.0

Realokacijų skaičius vienodas std::vector ir Vector klasei:

Atlikti std::vector vs Vector testai, užpildant vektorius 10000, 100000, 1000000, 10000000, 100000000 elementų push_back() funkcija. Pagal visus testus ir jų vidurkį, greičiau užpildomas naujai sukurtas Vector klasės vektorius.
Išteštuoti Vector klasės metodai:
5 Vector klasės metodų pavyzdžiai: //1 void push_back(const T& value) { if (size_ == capacity_) { reserve(capacity↩
_ == 0 ? 1 : capacity_ ∗ 2); } data_[size_++] = value; }
//2 void push_back(T&& value) { if (size_ == capacity_) { reserve(capacity_ == 0 ? 1 : capacity_ ∗ 2); } data_[size↩
_++] = std::move(value); }
//3 void pop_back() { if (size_ > 0) { –size_; } }
//4 void clear() { size_ = 0; }
//5 void assign(T∗ first, T∗ last) { clear(); size_t new_size = last - first; if (new_size > capacity_) { reallocate(new↩
_size); } for (size_t i = 0; i < new_size; ++i) { data_[i] = ∗(first + i); } size_ = new_size; }
main.cpp:
Meniu su galimais pasirinkimais:
1 - Konstruktorius
2 - Destruktorius
3 - Copy constructor
4 - Copy assignment operator
5 - Move constructor
6 - Move operator
7 - Input operator
8 - Output operator
9 - studentų grupavimas ir išvedimas į failus

```
9.1 - įvesti duomenis ranka

9.2 - generuoti duomenis

9.3 - nuskaityti duomenis iš failo
```

Kopiuterio parametrai: CPU - Apple M3 RAM - 16 GB SSD - 494,38 GB

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

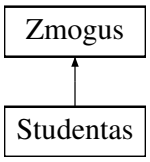
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Studentas Class Reference

Inheritance diagram for Studentas:



Public Member Functions

- **Studentas** (const Studentas &other)
- Studentas & **operator=** (const Studentas &other)
- **Studentas** (Studentas &&other) noexcept
- Studentas & **operator=** (Studentas &&other) noexcept
- string vardas () const override
- string pavarde () const override
- double **galutinisVid** () const
- double **galutinisMed** () const
- vector< int > **getNamuDarbai** () const
- int **getEgzaminas** () const
- size_t **getNamuDarbaiSize** () const
- void setVardas (const string &vardas) override
- void setPavarde (const string &pavarde) override
- void **setEgzaminas** (int egzas)
- void **setNamuDarbai** (const vector< int > &nd)
- void **setGalutinisVid** (double vid)
- void **setGalutinisMed** (double med)
- istream & **readStudent** (istream &is)
- void **skaiciuotiGalutinius** ()

Public Member Functions inherited from Zmogus

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

Static Public Member Functions

- static double **skaiciuotiVidurki** (const vector< int > &pazymiai)
- static double **skaiciuotiMediana** (vector< int > pazymiai)

Friends

- ostream & **operator**<< (ostream &os, const Studentas &s)
- istream & **operator**>> (istream &is, Studentas &s)

Additional Inherited Members

Protected Attributes inherited from Zmogus

- string **vardas_**
- string **pavarde_**

5.1.1 Member Function Documentation

5.1.1.1 pavarde()

```
string Studentas::pavarde () const  [inline], [override], [virtual]
```

Implements Zmogus.

5.1.1.2 setPavarde()

```
void Studentas::setPavarde (
            const string & pavarde)  [inline], [override], [virtual]
```

Implements Zmogus.

5.1.1.3 setVardas()

```
void Studentas::setVardas (
            const string & vardas)  [inline], [override], [virtual]
```

Implements Zmogus.

5.1.1.4 vardas()

```
string Studentas::vardas () const  [inline], [override], [virtual]
```

Implements Zmogus.

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

- student.h
- student.cpp

5.2 Vector< T > Class Template Reference

Public Member Functions

- **Vector** (size_t size)
- **Vector** (std::initializer_list< T > init)
- **Vector** (const Vector &other)
- **Vector** (Vector &&other) noexcept
- Vector & **operator=** (const Vector &other)
- Vector & **operator=** (Vector &&other) noexcept
- T * **erase** (T *pos)
- T * **erase** (T *first, T *last)
- void **shrink_to_fit** ()
- void **swap** (Vector &other) noexcept
- T & **operator[ ]** (size_t index)
- const T & **operator[ ]** (size_t index) const
- T & **at** (size_t index)
- const T & **at** (size_t index) const
- T & **front** ()

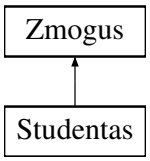
- const T & **front** () const
- T & **back** ()
- const T & **back** () const
- size_t **size** () const
- size_t **capacity** () const
- bool **empty** () const
- void **reserve** (size_t new_capacity)
- void **resize** (size_t new_size)
- void **push_back** (const T &value)
- void **push_back** (T &&value)
- void **pop_back** ()
- void **clear** ()
- void **assign** (T *first, T *last)
- bool **operator==** (const Vector &other) const
- bool **operator!=** (const Vector &other) const
- T * **begin** ()
- T * **end** ()
- const T * **begin** () const
- const T * **end** () const

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

- vector.h

5.3 Zmogus Class Reference

Inheritance diagram for Zmogus:



Public Member Functions

- **Zmogus** (const string &vardas, const string &pavarde)
- virtual string **vardas** () const =0
- virtual string **pavarde** () const =0
- virtual void **setVardas** (const string &vardas)=0
- virtual void **setPavarde** (const string &pavarde)=0

Protected Attributes

- string **vardas_**
- string **pavarde_**

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

- student.h

Chapter 6

File Documentation

6.1 student.h

```
00001 #ifndef STUDENT_H
00002 #define STUDENT_H
00003
00004 #include <iostream>
00005 #include <vector>
00006 #include <string>
00007 #include <iomanip>
00008 #include <algorithm>
00009 #include <numeric>
00010 #include <stdexcept>
00011 #include <fstream>
00012 #include <sstream>
00013
00014 using namespace std;
00015 using namespace std::chrono;
00016
00017 class Zmogus {
00018     protected:
00019         string vardas_;
00020         string pavarde_;
00021     public:
00022         Zmogus() = default;
00023         Zmogus(const string& vardas, const string& pavarde) : vardas_(vardas), pavarde_(pavarde) {}
00024         virtual ~Zmogus() = default;
00025
00026         virtual string vardas() const = 0;
00027         virtual string pavarde() const = 0;
00028
00029         virtual void setVardas(const string& vardas) = 0;
00030         virtual void setPavarde(const string& pavarde) = 0;
00031 };
00032
00033 class Studentas : public Zmogus {
00034     private:
00035         vector<int> namuDarbai_;
00036         int egzaminas_;
00037         double galutinisVid_;
00038         double galutinisMed_;
00039
00040     public:
00041         // Konstruktoriai ir destruktoriai
00042         Studentas();
00043         ~Studentas();
00044
00045         Studentas(const Studentas& other);
00046         Studentas& operator=(const Studentas& other);
00047         Studentas(Studentas&& other) noexcept;
00048         Studentas& operator=(Studentas&& other) noexcept;
00049
00050         // Getteriai
00051         string vardas() const override { return vardas_; }
00052         string pavarde() const override { return pavarde_; }
00053         double galutinisVid() const { return galutinisVid_; }
00054         double galutinisMed() const { return galutinisMed_; }
00055         vector<int> getNamuDarbai() const { return namuDarbai_; }
00056         int getEgzaminas() const { return egzaminas_; }
00057         size_t getNamuDarbaiSize() const { return namuDarbai_.size(); }
00058
00059         // Setteriai
00060         void setVardas(const string& vardas) override { vardas_ = vardas; }
00061         void setPavarde(const string& pavarde) override { pavarde_ = pavarde; }
```

```
00062         void setEgzaminas(int egzas) { egzaminas_ = egzas; }
00063         void setNamuDarbai(const vector<int>& nd) { namuDarbai_ = nd; }
00064         void setGalutinisVid(double vid) { galutinisVid_ = vid; }
00065         void setGalutinisMed(double med) { galutinisMed_ = med; }
00066 
00067         // Funkcijos
00068         istream& readStudent(istream& is);
00069 
00070         void skaiciuotiGalutinius();
00071         static double skaiciuotiVidurki(const vector<int>& pazymiai);
00072         static double skaiciuotiMediana(vector<int> pazymiai);
00073 
00074         // Operatoriai
00075         friend ostream& operator«(ostream& os, const Studentas& s);
00076         friend istream& operator»(istream& is, Studentas& s);
00077     };
00078 
00079 #endif
```

6.2 utils.h

```
00001 #ifndef PROGRAMA_UTILS_H
00002 #define PROGRAMA_UTILS_H
00003 
00004 #include "student.h"
00005 #include "vector.h"
00006 
00007 void testDestructor();
00008 void testConstructor();
00009 void testCopyConstructor();
00010 void testCopyAssignment();
00011 void testMoveConstructor();
00012 void testMoveAssignment();
00013 void testInputOperator();
00014 void testOutputOperator();
00015 void ivestiStudenta(Vector<Studentas>& studentai);
00016 void generuotiStudentus(Vector<Studentas>& studentai);
00017 
00018 template <typename konteineris>
00019 void nuskaitytiIsFailo(konteineris &studentai) {
00020     string failoPavadinimas;
00021     ifstream failas;
00022 
00023     while (true) {
00024         try {
00025             cout « "\nĮveskite failo pavadinimą: ";
00026             cin » failoPavadinimas;
00027 
00028             failas.open(failoPavadinimas);
00029             if (!failas) throw runtime_error("Nepavyko atidaryti failo!");
00030 
00031             break;
00032         }
00033         catch (const exception& e) {
00034             cout « e.what() « " Bandykite dar kartą.\n";
00035             cin.clear();
00036             cin.ignore(numeric_limits<streamsize>::max(), '\n');
00037         }
00038     }
00039 
00040     auto start = steady_clock::now();
00041 
00042     string eilute;
00043     getline(failas, eilute);
00044 
00045     while (getline(failas, eilute)) {
00046         istringstream line(eilute);
00047         Studentas stud;
00048 
00049 
00050         stud.readStudent(line);
00051 
00052         studentai.push_back(stud);
00053     }
00054 
00055     failas.close();
00056 
00057     auto end = steady_clock::now();
00058     auto trukme = duration_cast<duration<double»(end - start);
00059     cout « fixed « setprecision(3);
00060     cout « "Duomenų nuskaitymas užtruko: " « trukme.count() « " s\n" « endl;
00061 }
00062 
00063 template <typename konteineris>
```

```
00064 void rusiuotiStudentus(konteineris &studentai){
00065
00066     int rusiavimoPasirinkimas;
00067
00068     while (true) {
00069         try {
00070             cout « "Pasirinkite rikiavimo būdą:\n";
00071             cout « "1 - Pagal vardą (A-Z)\n";
00072             cout « "2 - Pagal pavardę (A-Z)\n";
00073             cout « "3 - Pagal galutinį vidurkį\n";
00074             cout « "4 - Pagal galutinę medianą\n";
00075             cout « "Jūsų pasirinkimas: ";
00076             cin » rusiavimoPasirinkimas;
00077
00078             if (cin.fail()) {
00079                 throw invalid_argument("Neteisinga įvestis! Įveskite tik skaičių.");
00080             }
00081             if (rusiavimoPasirinkimas < 1 || rusiavimoPasirinkimas > 4) {
00082                 throw out_of_range("Pasirinkimas turi būti nuo 1 iki 4.");
00083             }
00084             break;
00085         }
00086         catch (const exception &e) {
00087             cout « e.what() « " Bandykite dar kartą.\n";
00088             cin.clear();
00089             cin.ignore(numeric_limits<streamsize>::max(), '\n');
00090         }
00091     }
00092
00093     auto start = steady_clock::now();
00094
00095     switch (rusiavimoPasirinkimas) {
00096         case 1:
00097             std::sort(studentai.begin(), studentai.end(), [](const Studentas& a, const Studentas& b) {
00098                 return a.vardas() < b.vardas();
00099             });
00100             break;
00101         case 2:
00102             std::sort(studentai.begin(), studentai.end(), [](const Studentas& a, const Studentas& b) {
00103                 return a.pavarde() < b.pavarde();
00104             });
00105             break;
00106         case 3:
00107             std::sort(studentai.begin(), studentai.end(), [](const Studentas& a, const Studentas& b) {
00108                 return a.galutinisVid() < b.galutinisVid();
00109             });
00110             break;
00111         case 4:
00112             std::sort(studentai.begin(), studentai.end(), [](const Studentas& a, const Studentas& b) {
00113                 return a.galutinisMed() < b.galutinisMed();
00114             });
00115             break;
00116     }
00117
00118     auto end = steady_clock::now();
00119     auto trukme = duration_cast<duration<double»(end - start);
00120     cout « fixed « setprecision(3);
00121     cout « "Rikiavimas užtruko: " « trukme.count() « " s\n" « endl;
00122 }
00123
00124 template <typename konteineris>
00125 void strategija_3(konteineris& studentai, konteineris& nuskriaustukai) {
00126
00127     char grupavimoPasirinkimas;
00128     Studentas stud;
00129
00130     while (true) {
00131         try {
00132             cout « "Pasirinkite pagal ką bus sugrupuoti studentai (V - pagal vidurkį, M - pagal
      medianą): ";
00133             cin » grupavimoPasirinkimas;
00134
00135             if (grupavimoPasirinkimas != 'V' && grupavimoPasirinkimas != 'v' && grupavimoPasirinkimas
      != 'M' && grupavimoPasirinkimas != 'm') {
00136                 throw invalid_argument("Neteisinga įvestis! Pasirinkite V arba M.");
00137             }
00138
00139             break;
00140         }
00141         catch (const invalid_argument& e) {
00142             cout « e.what() « " Bandykite dar kartą.\n";
00143             cin.clear();
00144             cin.ignore(numeric_limits<streamsize>::max(), '\n');
00145             }
00146         }
00147
00148     auto start = steady_clock::now();
```

```
00149
00150     auto it = stable_partition(studentai.begin(), studentai.end(), [grupavimoPasirinkimas](const
      Studentas& stud) {
00151         if (grupavimoPasirinkimas == 'V' || grupavimoPasirinkimas == 'v') {
00152             return stud.galutinisVid() >= 5;
00153         } else {
00154             return stud.galutinisMed() >= 5;
00155         }
00156     });
00157
00158     nuskriaustukai.assign(it, studentai.end());
00159     studentai.erase(it, studentai.end());
00160
00161     auto end = steady_clock::now();
00162
00163     if constexpr (is_same_v<konteineris, vector<Studentas» || is_same_v<konteineris, deque<Studentas»)
      {
00164         studentai.shrink_to_fit();
00165     }
00166
00167     auto trukme = duration_cast<duration<double»(end - start);
00168     cout « fixed « setprecision(3);
00169     cout « "3 strategija užtruko: " « trukme.count() « " s\n" « endl;
00170 }
00171
00172 template <typename konteineris>
00173 void isvestiIDuFailus(konteineris& nuskriaustukai, konteineris& studentai){
00174
00175     ofstream outNuskriaustukai("nuskriaustukai.txt"), outKietekai("kietekai.txt");
00176     if (!outNuskriaustukai || !outKietekai) {
00177         cerr « "Nepavyko sukurti rezultatų failų!" « endl;
00178         return;
00179     }
00180
00181     outNuskriaustukai « left « setw(20) « "Pavarde" « setw(20) « "Vardas" « setw(20) « "Galutinis
      (Vid.)" « setw(20) « "Galutinis (Med.)" « endl;
00182     outNuskriaustukai « string(70, '-') « endl;
00183     outKietekai « left « setw(20) « "Pavarde" « setw(20) « "Vardas" « setw(20) « "Galutinis (Vid.)" «
      setw(20) « "Galutinis (Med.)" « endl;
00184     outKietekai « string(70, '-') « endl;
00185
00186     for (const auto& stud : nuskriaustukai) {
00187         outNuskriaustukai « left « setw(20) « stud.pavarde() « setw(20) « stud.vardas() « fixed «
      setprecision(2) « setw(20) « stud.galutinisVid() « setw(20) « stud.galutinisMed() « endl;
00188     }
00189     for (const auto& stud : studentai) {
00190         outKietekai « left « setw(20) « stud.pavarde() « setw(20) « stud.vardas() « fixed «
      setprecision(2) « setw(20) « stud.galutinisVid() « setw(20) « stud.galutinisMed() « endl;
00191     }
00192
00193     cout « "Failai \"nuskriaustukai.txt\" ir \"kietekai.txt\" sukurti!" « endl;
00194 }
00195
00196 void test_vector_push_back(unsigned int sz);
00197
00198 #endif
```

6.3 vector.h

```
00001 #ifndef VECTOR_H
00002 #define VECTOR_H
00003
00004 #include <stdexcept>
00005 #include <initializer_list>
00006 #include <algorithm>
00007 #include <iterator>
00008
00009 template <typename T>
00010 class Vector {
00011 private:
00012     T* data_;
00013     size_t size_;
00014     size_t capacity_;
00015
00016     void reallocate(size_t new_capacity) {
00017         T* new_data = new T[new_capacity];
00018         for (size_t i = 0; i < size_; ++i) {
00019             new_data[i] = std::move(data_[i]);
00020         }
00021         delete[] data_;
00022         data_ = new_data;
00023         capacity_ = new_capacity;
00024     }
00025
```

```
00026     public:
00027     // Kontruktoriai
00028     Vector() : data_(nullptr), size_(0), capacity_(0) {}
00029
00030     explicit Vector(size_t size) : data_(new T[size]), size_(size), capacity_(size) {}
00031
00032     Vector(std::initializer_list<T> init) : Vector(init.size()) {
00033         std::copy(init.begin(), init.end(), data_);
00034     }
00035
00036     // Kopijavimo konstruktorius
00037     Vector(const Vector& other) : data_(new T[other.capacity_]), size_(other.size_),
      capacity_(other.capacity_) {
00038         std::copy(other.data_, other.data_ + other.size_, data_);
00039     }
00040
00041     // Move kontruktorius
00042     Vector(Vector&& other) noexcept : data_(other.data_), size_(other.size_),
      capacity_(other.capacity_) {
00043         other.data_ = nullptr;
00044         other.size_ = 0;
00045         other.capacity_ = 0;
00046     }
00047
00048     // Destruktorius
00049     ~Vector() {
00050         delete[] data_;
00051     }
00052
00053     // Kopijavimo priskyrimo operatorius
00054     Vector& operator=(const Vector& other) {
00055         if (this != &other) {
00056             delete[] data_;
00057             data_ = new T[other.capacity_];
00058             size_ = other.size_;
00059             capacity_ = other.capacity_;
00060             std::copy(other.data_, other.data_ + other.size_, data_);
00061         }
00062         return *this;
00063     }
00064
00065     // Move priskyrimo 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     // Ištrinti elementą tam tikroje pozicijoje
00080     T* erase(T* pos) {
00081         size_t index = pos - data_;
00082         for (size_t i = index; i < size_ - 1; ++i) {
00083             data_[i] = std::move(data_[i + 1]);
00084         }
00085         --size_;
00086         return data_ + index;
00087     }
00088
00089     T* erase(T* first, T* last) {
00090         size_t start_index = first - data_;
00091         size_t end_index = last - data_;
00092         size_t range_size = end_index - start_index;
00093
00094         for (size_t i = start_index; i < size_ - range_size; ++i) {
00095             data_[i] = std::move(data_[i + range_size]);
00096         }
00097         size_ -= range_size;
00098         return data_ + start_index;
00099     }
00100
00101     // Shrink to fit
00102     void shrink_to_fit() {
00103         if (size_ < capacity_) {
00104             reallocate(size_);
00105         }
00106     }
00107
00108     // Swap du vektorius
00109     void swap(Vector& other) noexcept {
00110         std::swap(data_, other.data_);
```

```
00111         std::swap(size_, other.size_);
00112         std::swap(capacity_, other.capacity_);
00113     }
00114
00115     // Elementų pasiekimas
00116     T& operator[](size_t index) {
00117         if (index >= size_) throw std::out_of_range("Indeksas ne intervale");
00118         return data_[index];
00119     }
00120
00121     const T& operator[](size_t index) const {
00122         if (index >= size_) throw std::out_of_range("Indeksas ne intervale");
00123         return data_[index];
00124     }
00125
00126     T& at(size_t index) {
00127         if (index >= size_) throw std::out_of_range("Indeksas ne intervale");
00128         return data_[index];
00129     }
00130
00131     const T& at(size_t index) const {
00132         if (index >= size_) throw std::out_of_range("Indeksas ne intervale");
00133         return data_[index];
00134     }
00135
00136     // Pasiekti pirmą elementą
00137     T& front() {
00138         if (empty()) throw std::out_of_range("Vektorius tuščias");
00139         return data_[0];
00140     }
00141
00142     const T& front() const {
00143         if (empty()) throw std::out_of_range("Vektorius tuščias");
00144         return data_[0];
00145     }
00146
00147     // Pasiekti paskutinį elementą
00148     T& back() {
00149         if (empty()) throw std::out_of_range("Vektorius tuščias");
00150         return data_[size_ - 1];
00151     }
00152
00153     const T& back() const {
00154         if (empty()) throw std::out_of_range("Vektorius tuščias");
00155         return data_[size_ - 1];
00156     }
00157
00158     // Capacity
00159     size_t size() const { return size_; }
00160
00161     size_t capacity() const { return capacity_; }
00162
00163     bool empty() const { return size_ == 0; }
00164
00165     void reserve(size_t new_capacity) {
00166         if (new_capacity > capacity_) {
00167             reallocate(new_capacity);
00168         }
00169     }
00170
00171     void resize(size_t new_size) {
00172         if (new_size > capacity_) {
00173             reserve(new_size);
00174         }
00175         for (size_t i = size_; i < new_size; ++i) {
00176             data_[i] = T();
00177         }
00178         size_ = new_size;
00179     }
00180
00181     // Modifikatoriai
00182     void push_back(const T& value) {
00183         if (size_ == capacity_) {
00184             reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00185         }
00186         data_[size_++] = value;
00187     }
00188
00189     void push_back(T&& value) {
00190         if (size_ == capacity_) {
00191             reserve(capacity_ == 0 ? 1 : capacity_ * 2);
00192         }
00193         data_[size_++] = std::move(value);
00194     }
00195
00196     void pop_back() {
00197         if (size_ > 0) {
```

```
00198             --size_;
00199         }
00200     }
00201
00202     void clear() {
00203         size_ = 0;
00204     }
00205
00206     void assign(T* first, T* last) {
00207         clear();
00208         size_t new_size = last - first;
00209         if (new_size > capacity_) {
00210             reallocate(new_size);
00211         }
00212         for (size_t i = 0; i < new_size; ++i) {
00213             data_[i] = *(first + i);
00214         }
00215         size_ = new_size;
00216     }
00217
00218     // Lyginimo operatoriai
00219     bool operator==(const Vector& other) const {
00220         if (size_ != other.size_) return false;
00221         for (size_t i = 0; i < size_; ++i) {
00222             if (data_[i] != other.data_[i]) return false;
00223         }
00224         return true;
00225     }
00226
00227     bool operator!=(const Vector& other) const {
00228         return !(*this == other);
00229     }
00230
00231     // Iteratoriai
00232     T* begin() { return data_; }
00233     T* end() { return data_ + size_; }
00234     const T* begin() const { return data_; }
00235     const T* end() const { return data_ + size_; }
00236 };
00237
00238 #endif
```

Index