v2.0

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 Student Class Reference 7

4.1.1 Member Function Documentation 8

4.1.1.1 ppp() 8

4.2 Zmogus Class Reference 8

5 File Documentation **11**

5.1 failu-generavimas.h 11

5.2 student.h 11

5.3 vektoriai.h 13

Index **15**

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

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

Zmogus 8
    Student 7

Chapter 2

Class Index

2.1 Class List

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

Student 7
Zmogus 8

Chapter 3

File Index

3.1 File List

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

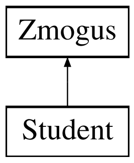
sources/failu-generavimas.h 11
sources/student.h 11
sources/vektoriai.h 13

Chapter 4

Class Documentation

4.1 Student Class Reference

Inheritance diagram for Student:



Public Member Functions

- void ppp () const override
- **Student** (const string &fName, const string &lName, const vector< int > &grades, int finalExamGrade, double median, double average)
- **Student** (const Student &other)
- **Student** (Student &&other) noexcept
- Student & **operator=** (const Student &other)
- Student & **operator=** (Student &&other) noexcept
- const vector< int > & **getGrades** () const
- int **getFinalExamGrade** () const
- double **getMedian** () const
- double **getAverage** () const
- double **getFinalMedian** () const
- double **getFinalAverage** () const
- double **getFinalGrade** () const
- void **setGrades** (const vector< int > &newGrades)
- void **setFinalExamGrade** (int examGrade)
- void **setMedian** (double medianValue)
- void **setAverage** (double averageValue)
- void **setFinalMedian** (double finalMedian)
- void **setFinalAverage** (double finalAverage)
- void **setFinalGrade** (double finalGradeValue)

Public Member Functions inherited from Zmogus

- string **getFirstName** () const
- string **getLastName** () const
- void **setFirstName** (const string &fName)
- void **setLastName** (const string &lName)

Friends

- std::istream & **operator**>> (istream &i, Student &student)
- std::ostream & **operator**<< (std::ostream &os, const Student &student)

Additional Inherited Members

Protected Member Functions inherited from Zmogus

- **Zmogus** (const string &fName, const string &lName)

Protected Attributes inherited from Zmogus

- string **firstName**
- string **lastName**

4.1.1 Member Function Documentation

4.1.1.1 ppp()

```
void Student::ppp ( ) const  [inline], [override], [virtual]
```

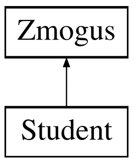
Implements Zmogus.

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

- sources/student.h

4.2 Zmogus Class Reference

Inheritance diagram for Zmogus:



Public Member Functions

- string **getFirstName** () const
- string **getLastName** () const
- void **setFirstName** (const string &fName)
- void **setLastName** (const string &lName)

Protected Member Functions

- **Zmogus** (const string &fName, const string &lName)
- virtual void **ppp** () const =0

Protected Attributes

- string **firstName**
- string **lastName**

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

- sources/student.h

Chapter 5

File Documentation

5.1 failu-generavimas.h

```cpp
#ifndef FAILU_GENERAVIMAS_H
#define FAILU_GENERAVIMAS_H

#include <iostream>
#include <fstream>
#include <iomanip>
#include <vector>
#include <chrono>
#include <thread>
#include <string>
#include <ctime>
#include <cstdlib>
#include "vektoriai.h"
#include "student.h"

using namespace std;

void writeCategorizedStudents(const vector<Student>& students, const string& filename);
void generateFiles();
void sortAndWriteToFile(const string& inputFilename);
void generatingFinal();

#endif
```

5.2 student.h

```cpp
#ifndef STUDENT_H
#define STUDENT_H

#include "vektoriai.h"

#include <vector>
#include <string>
#include <iomanip>
#include <algorithm>

using namespace std;

class Zmogus {
protected:
    string firstName;
    string lastName;

    Zmogus() = default;
    Zmogus(const string& fName, const string& lName) : firstName(fName), lastName(lName) {}
    virtual void ppp() const = 0;

public:
    virtual ~Zmogus() {}
    string getFirstName() const { return firstName; }
    string getLastName() const { return lastName; }

    void setFirstName(const string& fName) { firstName = fName; }
    void setLastName(const string& lName) { lastName = lName; }
```

```
};

class Student : public Zmogus {
private:
    vector<int> grades;
    int finalExamGrade;
    double median, average;
    double fin_median, fin_average, finalGrade;

public:
    void ppp() const override {}
    Student() : finalExamGrade(0), median(0.0), average(0.0), fin_median(0.0), fin_average(0.0),
finalGrade(0.0) {}
    Student(const string& fName, const string& lName, const vector<int>& grades, int finalExamGrade,
double median, double average)
     : Zmogus(fName,lName), grades(grades), finalExamGrade(finalExamGrade), median(median),
average(average), fin_median(0.0), fin_average(0.0), finalGrade(0.0) {}

   // Destructor
   ~Student() {
    grades.clear();
    firstName.clear();
    lastName.clear();
    }

    // Copy Constructor
    Student(const Student& other)
    : Zmogus(other.firstName, other.lastName), grades(other.grades),
finalExamGrade(other.finalExamGrade), median(other.median), average(other.average),
fin_median(other.fin_median), fin_average(other.fin_average), finalGrade(other.finalGrade) {}

    // Move Constructor
    Student(Student&& other) noexcept
    : Zmogus(move(other.firstName), move(other.lastName)),
      grades(move(other.grades)),
      finalExamGrade(move(other.finalExamGrade)),
      median(move(other.median)),
      average(move(other.average)),
      fin_median(move(other.fin_median)),
      fin_average(move(other.fin_average)),
      finalGrade(move(other.finalGrade)) {

    other.firstName.clear();
    other.lastName.clear();
    other.grades.clear();
}

    // Copy Assignment Operator
    Student& operator=(const Student& other) {
        if (this != &other) { // self-assignment check
        Zmogus::setFirstName(other.getFirstName());
        Zmogus::setLastName(other.getLastName());
            grades = other.grades;
            finalExamGrade = other.finalExamGrade;
            median = other.median;
            average = other.average;
            fin_median = other.fin_median;
            fin_average = other.fin_average;
            finalGrade = other.finalGrade;
        }
        return *this;
    }

    // Move Assignment Operator
  Student& operator=(Student&& other) noexcept {
    if (this != &other) {
        Zmogus::setFirstName(move(other.getFirstName()));
        Zmogus::setLastName(move(other.getLastName()));
        grades = move(other.grades);
        finalExamGrade = move(other.finalExamGrade);
        median = move(other.median);
        average = move(other.average);
        fin_median = move(other.fin_median);
        fin_average = move(other.fin_average);
        finalGrade = move(other.finalGrade);
        other.firstName.clear();
        other.lastName.clear();
        other.grades.clear();
    }
    return *this;
}

    // Input Operator
friend std::istream& operator»(istream& i, Student& student) {
    string firstName, lastName;
    i » firstName » lastName;
    student.setFirstName(firstName);
    student.setLastName(lastName);

    vector<int> grades;
    for (int j = 0; j < 15; ++j) {
        int grade;
        i » grade;
        grades.push_back(grade);
    }
    student.setGrades(grades);

    i » student.finalExamGrade;

    // final average
    double sum = 0;
    for (int grade : grades) {
        sum += grade;
    }
    double average = sum / grades.size();
    double finalAverage = average * 0.4 + student.finalExamGrade * 0.6;
    student.setFinalAverage(finalAverage);

    // final median
    sort(grades.begin(), grades.end());
    double finalMedian;
    if (grades.size() % 2 == 0) {
        finalMedian = (grades[grades.size() / 2 - 1] + grades[grades.size() / 2]) / 2.0;
    } else {
        finalMedian = grades[grades.size() / 2];
    }
    finalMedian = finalMedian * 0.4 + student.finalExamGrade * 0.6;
    student.setFinalMedian(finalMedian);

    return i;
}

// Output Operator
friend std::ostream& operator«(std::ostream& os, const Student& student) {
    os « setw(10) « student.getFirstName() « setw(20) « student.getLastName();
    double average = student.getAverage() * 0.4 + student.getFinalExamGrade() * 0.6;
    double median = student.getMedian() * 0.4 + student.getFinalExamGrade() * 0.6;
    os « fixed « setw(25) « setprecision(2) « average;
    os « fixed « setw(25) « setprecision(2) « median « '\n';
    return os;
}

    const vector<int>& getGrades() const { return grades; }
    int getFinalExamGrade() const { return finalExamGrade; }
    double getMedian() const { return median; }
    double getAverage() const { return average; }
    double getFinalMedian() const { return fin_median; }
    double getFinalAverage() const { return fin_average; }
    double getFinalGrade() const { return finalGrade; }


    void setGrades(const vector<int>& newGrades) { grades = newGrades; }
    void setFinalExamGrade(int examGrade) { finalExamGrade = examGrade; }
    void setMedian(double medianValue) { median = medianValue; }
    void setAverage(double averageValue) { average = averageValue; }
    void setFinalMedian(double finalMedian) { fin_median = finalMedian; }
    void setFinalAverage(double finalAverage) { fin_average = finalAverage; }
    void setFinalGrade(double finalGradeValue) { finalGrade = finalGradeValue; }
};

#endif
```

5.3 vektoriai.h

```
#ifndef VEKTORIAI_H
#define VEKTORIAI_H

//#include "student.h"

#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <algorithm>
#include <iomanip>
#include <ctime>
#include <cstdlib>
#include <sstream>
#include <limits>
#include <numeric>
```

```
#include <chrono>
#include <cassert>


using namespace std;

class Student;
bool isValidName(const string& name);
bool isValidGrade(const string& grade);
double calculateAverage(const Student& student);
double calculateMedian(const Student& student);
void randomGradeGenerator(int number, Student& student);
void generateNames(Student& student);
void readFromFile(const string& filename, vector<Student>& students);
void tests();

#endif
```

Index

ppp
    Student, 8

sources/failu-generavimas.h, 11
sources/student.h, 11
sources/vektoriai.h, 13
Student, 7
    ppp, 8

Zmogus, 8