My Project

Generated by Doxygen 1.10.0

Chapter 1

README

v2.0

Makefile set-up: **Makefile idiegimas naudojant Chocolatey (Windows)**

Isitikinkite, kad turite Chocolatey idiegima: patikrinkite, ar jusu kompiuteryje yra idiegta Chocolatey. Jei ne, idiekite pagal instrukcijas `https://chocolatey.org/install`.

Idiekite Makefile: atidarykite PowerShell kaip administratorius ir ivykdykite sia komanda:

choco install make

Patikrinkite idiegima: patikrinkite, ar Makefile sekmingai idiegtas, vykdydami komanda:

make –version

Jei viskas sekminga, turetumete pamatyti Make versijos informacija

Makefile idiegimas naudojant kitus metodus **MacOS:** Makefile iprastai yra idiegtas standartinėje MacOS distribucijoje, todel papildomu veiksmu paprastai nereikia

Linux: Daugumoje Linux distribuciju Makefile taip pat yra idiegtas is anksto. Jei reikia, naudokite savo paketu tvarkykle, pvz., apt-get, yum, dnf, arba kita pagal distribucija

Windows (be Chocolatey): Noredami idiegti Makefile Windows sistemoje be Chocolatey, galite naudoti rankinius idiegimo failus, kuriuos galite rasti internete. Paprastai tie failai turi .exe pletini ir gali buti lengvai idiegti, sekdamie pridedamas instrukcijas

Norint pradeti, i terminala reikia ivesti "make", kai viskas bus sukompiliuota, galima testi su programa, jei norima, galima rasyti "make clean" norint istrinti .o ir .exe failus

Programos naudojimas veikimo metu:

Vartotojas pasirenka, su kokiu konteineriu norima vygdyti programa ir pasirinktinai i terminala parasoma: ./vektoriai , ./list arba ./deque

Vos paleidus programa atsiras pasirinkimas ar pratestuoti musu turimus klases metodus , jei paspaudziame 't', tada pasirenkame numeri nuo 1-5 ir gauname testo rezultata

Toliau musu bus klausiama, ar norime ivesti duomenis ar skaityti is failo

1. Jei bus pasirenkamas duomenu ivedimas, bus reikalaujama pasirinkti ar norima ivesti/generuoti duomenis

1.1 Ar vienu, ar kitu budu reikes ivesti studentu vardus ir pavardes, toliau reikes ivesti studentu namu darbu ir egzamino pazymius

1.2 Jei bus pasirinktas duomenu generavimas, po vardu ir pavardziu irasymo nieko daryti nebereikes

1.3 Galiausiai reikes pasirinkti kur norime matyti duomenis ekrane ar faile

1. Jei pacioje pradzioje bus pasirinktas skaitymas, jusu bus klausiama ar norite generuoti naujus failus, jei ivesite 't'(taip), bus generuojami nauji failai, jei ivesite bet koki kita simboli, programa veiks toliau

2.1 Toliau, jusu bus klausiama ar norite skaityti naujai sukurtus failus, ar jau turimus

2.2 Bus prasoma ivesti, pasirinktinai, turimu/nauju failu kieki, jie bus nuskaitomi, isvedami apytiksliai testavimu laikai ekrane bei sukuriami nauji failai, kuriuose yra surusiuoti studentai pagal vidurki (nuskriaustieji/mokslinciai)

2.3 Galiausiai, kaip ir anksciau, bus isvedami apytiksliai testavimu laikai ekrane bei sukuriami nauji failai, kuriuose yra surusiuoti studentai pagal vidurki (nuskriaustieji/mokslinciai)

RELEASAI

0.1

Sukurta nauja repozitorija, realizuotos elementarios funkcijos, kaip vidurkio ir medianos skaiciavimas. Rezultate gavome, kad vektorius naudoti yra zymiai efektyviau atminties atzvilgiu.

0.2

Programa padaryta prieinamesne vartotojui, galima ne tik irasyti, bet ir skaityti is failo. Testuojama su 10000, 100000 ir 1000000 dydzio failais.

0.3

Prideti header failai, try/catch blokai. Rezultate programa tapo labiau strukturizuota bei klaidu gaudymas uzdrausdavo programos luzima.

0.4

Programa pagal vartotojo pasirinkima sukuria naujus failus, isskirsto mokinius i vargsiukus ir mokslincius, isveda i failus. Padaryti tikslus laiko matavimai.

1.0

Programa padaryta veikti su atskiro tipo konteineriais: deque, list ir vector. Kiekvienas pagal tris strategijas. Pagal matavimo rezultatus greiciausiai buvo vykdoma vector programa naudojant 3 strategija.

1.1

Atliktas repozitorijos kopijavimas. Programoje is strukturu pereinama i klases. Rezultate, programa veikia nasiau naudojant klases.

1.2

Igyvendinti visi "Rule of Five" ir isviesties bei ivesties operatoriai savai klasei.

1.5

Sukurta dar viena **bazine**, abstrakcioji klase, kuriai priklauso klase derived. Prideti konstruktoriaus, copy konstruktoriaus... testavimai.

2.0

Per Doxygen HTML formatu sukurta dokumentacija bei padaryti Unit testai naudojant patogu C++ framework'a supratimui.

Kaip atrodo Doxygen dokumentacija:

Unit testai:

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

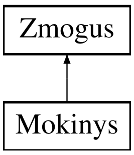
Here is a list of all documented files with brief descriptions:

Chapter 5

Class Documentation

5.1 Mokinys Class Reference

Inheritance diagram for Mokinys:



Public Member Functions

- **Mokinys** (string vard="", string pav="", Vector< int > nd={}, int e=0, double vid=0.0, double med=0.0)
- void patikrinimas () const override
- **Mokinys** (const Mokinys &other)
- **Mokinys** (Mokinys &&other) noexcept
- Mokinys & **operator=** (const Mokinys &other)
- void **clear** ()
- Mokinys & **operator=** (Mokinys &&other) noexcept
- string **getVardas** () const
- string **getPavarde** () const
- Vector< int > **getND** () const
- int **getEgzaminas** () const
- double **getVID** () const
- double **getMED** () const
- void **setVardas** (const string &name)
- void **setPavarde** (const string &surname)
- void **addND** (int nd)
- void **clearND** ()
- void **setEgzaminas** (int exam)
- void **setVID** (double vid)
- void **setMED** (double med)
- void **Vidurkis** (Vector< Mokinys > &A)
- void **Isvedimas** (const Vector< Mokinys > &A, int MOK_kiekis, string isvedimas)
- void **Isvedimas2** (const Vector< Mokinys > &A, int MOK_kiekis, string isvedimas)

- void **Skaitymas** (Vector< Mokinys > &Nuskriaustieji, Vector< Mokinys > &Mokslinciai, Vector< int > &IrasuSk, string failas, Vector< Mokinys > &A, int &temp, char strategija)
- void **StudentuRusiavimas** (Vector< Mokinys > &Nuskriaustieji, Vector< Mokinys > &Mokslinciai, Vector< Mokinys > &A, Vector< int > &IrasuSk, string failas, int &temp)
- void **StudentuRusiavimas2** (Vector< Mokinys > &Nuskriaustieji, Vector< Mokinys > &Mokslinciai, Vector< Mokinys > &A, Vector< int > &IrasuSk, string failas, int &temp)
- void **StudentuRusiavimas3** (Vector< Mokinys > &Nuskriaustieji, Vector< Mokinys > &Mokslinciai, Vector< Mokinys > &A, Vector< int > &IrasuSk, string failas, int &temp)
- void **Rikiavimas** (Vector< Mokinys > &Mokslinciai, Vector< Mokinys > &Nuskriaustieji, Vector< int > &IrasuSk)

Public Member Functions inherited from Zmogus

- **Zmogus** (string vard="", string pav="")

Static Public Member Functions

- static bool **PagalVidurki** (const Mokinys &a, const Mokinys &b)
- static bool **PagalMediana** (const Mokinys &a, const Mokinys &b)
- static bool **PagalVarda** (const Mokinys &a, const Mokinys &b)
- static bool **PagalPavarde** (const Mokinys &a, const Mokinys &b)

Friends

- std::ostream & **operator**<< (std::ostream &fr, const Mokinys &temp1)
- istream & **operator**>> (istream &fd, Mokinys &temp1)

Additional Inherited Members

Protected Attributes inherited from Zmogus

- string **vardas**
- string **pavarde**

5.1.1 Member Function Documentation

5.1.1.1 patikrinimas()

```
void Mokinys::patikrinimas ( ) const  [inline], [override], [virtual]
```

Implements Zmogus.

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

- mokinys.h
- mokinys.cpp

5.2 Vector< T > Class Template Reference

Public Types

- using **value_type** = T

Public Member Functions

- **Vector** (size_t size, size_t capacity, const T &defaultValue)
- **Vector** (size_t initialCapacity)
- **Vector** (size_t size, const T &defaultValue)
- **Vector** (std::initializer_list< T > initList)
- **Vector** (const Vector &other)
- **Vector** (Vector &&other) noexcept
- Vector & **operator=** (const Vector &other)
- Vector & **operator=** (Vector &&other) noexcept
- void **push_back** (const T &value)
- void **push_back** (T &&value)
- void **pop_back** ()
- size_t **size** () const
- size_t **capacity** () const
- bool **empty** () const
- T & **operator[ ]** (size_t index)
- const T & **operator[ ]** (size_t index) const
- void **clear** ()
- void **reserve** (size_t newCapacity)
- T ∗ **begin** ()
- T ∗ **end** ()
- const T ∗ **begin** () const
- const T ∗ **end** () const
- T & **back** ()
- const T & **back** () const
- T & **front** ()
- const T & **front** () const
- T ∗ **data_ptr** ()
- const T ∗ **data_ptr** () const
- void **erase** (size_t index)
- void **resize** (size_t newSize, const T &defaultValue=T())
- void **swap** (Vector &other)
- template<typename InputIterator >
  void **assign** (InputIterator first, InputIterator last)
- void **assign** (size_t count, const T &value)
- void **assign** (std::initializer_list< T > ilist)
- void **insert** (size_t index, const T &value)
- template<typename InputIterator >
  void **insert_range** (size_t index, InputIterator first, InputIterator last)
- void **append_range** (std::initializer_list< T > ilist)
- T & **at** (size_t index)
- const T & **at** (size_t index) const
- std::reverse_iterator< T ∗ > **rbegin** ()
- std::reverse_iterator< T ∗ > **rend** ()
- std::reverse_iterator< const T ∗ > **rbegin** () const
- std::reverse_iterator< const T ∗ > **rend** () const

Friends

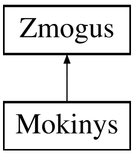
- bool **operator==** (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator!=** (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator<** (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator<=** (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator>** (const Vector< T > &lhs, const Vector< T > &rhs)
- bool **operator>=** (const Vector< T > &lhs, const Vector< T > &rhs)

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

- mokinys.h

5.3 Zmogus Class Reference

Inheritance diagram for Zmogus:



Public Member Functions

- **Zmogus** (string vard="", string pav="")
- virtual void **patikrinimas** () const =0

Protected Attributes

- string **vardas**
- string **pavarde**

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

- mokinys.h

Chapter 6

File Documentation

6.1 funkcijos.h

```
#ifndef FUNKCIJOS_H
#define FUNKCIJOS_H
#include "mokinys.h"

bool Patikrinimas(string kint);
void GeneruotiFailus(Vector<Mokinys>& Nuskriaustieji, Vector<Mokinys>& Mokslinciai, Vector<int>& IrasuSk, Vector<Mokinys>& A);
void testConstructor();
void testCopyConstructor();
void testMoveConstructor();
void testCopyAssignment();
void testMoveAssignment();
void Palyginimas();

#endif
```

6.2 mokinys.h

```
#ifndef MOKINYS_H
#define MOKINYS_H

#include <iostream>
#include <fstream>
#include <iomanip>
#include <string>
#include <Vector>
#include <sstream>
#include <algorithm>
#include <chrono>
#include <cstring>
#include <stdexcept>
#include <list>
#include <deque>
#include <cassert>
#include <utility>
#include <initializer_list>
#include <iterator>


using namespace std;

const char CRfv[] = "rezultatai.txt";
const char CRfv2[] = "naujas_failas.txt";
const char CRfv3[] = "mokslinciai.txt";
const char CRfv4[] = "nuskriaustieji.txt";

// const char CDfv[] = "kursiokai.txt";
const char CDfv0[] = "studentai10000.txt";
const char CDfv1[] = "studentai100000.txt";
const char CDfv2[] = "studentai1000000.txt";

template<typename T>
class Vector {
private:
    T* data;
    size_t _size;
    size_t _capacity;

    void reallocate(size_t newCapacity);

    template<typename InputIterator>
    void assign_impl(InputIterator first, InputIterator last, std::input_iterator_tag);

    template<typename RandomAccessIterator>
    void assign_impl(RandomAccessIterator first, RandomAccessIterator last,
std::random_access_iterator_tag);

public:
    Vector(size_t size, size_t capacity, const T& defaultValue)
    : _size(size), _capacity(capacity) {
    data = new T[_capacity];
    for (size_t i = 0; i < _size; ++i) {
        data[i] = defaultValue;
    }
}
    Vector();
    explicit Vector(size_t initialCapacity);
    Vector(size_t size, const T& defaultValue);
    Vector(std::initializer_list<T> initList);
    Vector(const Vector& other);
    Vector(Vector&& other) noexcept;
    Vector& operator=(const Vector& other);
    Vector& operator=(Vector&& other) noexcept;
    using value_type = T;
    ~Vector();

    void push_back(const T& value);
    void push_back(T&& value);
    void pop_back();
    size_t size() const;
    size_t capacity() const;
    bool empty() const;
    T& operator[](size_t index);
    const T& operator[](size_t index) const;
    void clear();
    void reserve(size_t newCapacity);
    T* begin();
    T* end();
    const T* begin() const;
    const T* end() const;
    T& back();
    const T& back() const;
    T& front();
    const T& front() const;
    T* data_ptr();
    const T* data_ptr() const;
    void erase(size_t index);
    void resize(size_t newSize, const T& defaultValue = T());
    void swap(Vector& other);

    template<typename InputIterator>
    void assign(InputIterator first, InputIterator last);
    void assign(size_t count, const T& value);
    void assign(std::initializer_list<T> ilist);

    void insert(size_t index, const T& value);
    template<typename InputIterator>
    void insert_range(size_t index, InputIterator first, InputIterator last);
    void append_range(std::initializer_list<T> ilist);

    T& at(size_t index);
    const T& at(size_t index) const;

    typename std::reverse_iterator<T*> rbegin();
    typename std::reverse_iterator<T*> rend();
    typename std::reverse_iterator<const T*> rbegin() const;
    typename std::reverse_iterator<const T*> rend() const;

    friend bool operator==(const Vector<T>& lhs, const Vector<T>& rhs) {
        if (lhs._size != rhs._size) {
            return false;
        }
        for (size_t i = 0; i < lhs._size; ++i) {
            if (lhs.data[i] != rhs.data[i]) {
                return false;
            }
        }
        return true;
    }

    friend bool operator!=(const Vector<T>& lhs, const Vector<T>& rhs) {
        return !(lhs == rhs);
    }

    friend bool operator<(const Vector<T>& lhs, const Vector<T>& rhs) {
        return std::lexicographical_compare(lhs.begin(), lhs.end(), rhs.begin(), rhs.end());
    }

    friend bool operator<=(const Vector<T>& lhs, const Vector<T>& rhs) {
        return !(rhs < lhs);
    }

    friend bool operator>(const Vector<T>& lhs, const Vector<T>& rhs) {
        return rhs < lhs;
    }

    friend bool operator>=(const Vector<T>& lhs, const Vector<T>& rhs) {
        return !(lhs < rhs);
    }
};

template<typename T>
Vector<T>::Vector() : data(nullptr), _size(0), _capacity(0) {}

template<typename T>
Vector<T>::Vector(size_t initialCapacity) : _size(0), _capacity(initialCapacity) {
    data = new T[_capacity];
}

template<typename T>
Vector<T>::Vector(size_t size, const T& defaultValue) : _size(size), _capacity(size) {
    data = new T[_capacity];
    std::fill(data, data + _size, defaultValue);
}

template<typename T>
Vector<T>::Vector(std::initializer_list<T> initList) : _size(initList.size()), _capacity(initList.size()) {
    data = new T[_capacity];
    std::copy(initList.begin(), initList.end(), data);
}

template<typename T>
Vector<T>::Vector(const Vector& other) : _size(other._size), _capacity(other._capacity) {
    data = new T[_capacity];
    std::copy(other.data, other.data + _size, data);
}

template<typename T>
Vector<T>::Vector(Vector&& other) noexcept : data(other.data), _size(other._size), _capacity(other._capacity) {
    other.data = nullptr;
    other._size = 0;
    other._capacity = 0;
}

template<typename T>
Vector<T>& Vector<T>::operator=(const Vector& other) {
    if (this != &other) {
        T* newData = new T[other._capacity];
        std::copy(other.data, other.data + other._size, newData);
        delete[] data;
        data = newData;
        _size = other._size;
        _capacity = other._capacity;
    }
    return *this;
}

template<typename T>
Vector<T>& Vector<T>::operator=(Vector&& other) noexcept {
    if (this != &other) {
        delete[] data;
        data = other.data;
        _size = other._size;
        _capacity = other._capacity;
        other.data = nullptr;
        other._size = 0;
        other._capacity = 0;
    }
    return *this;
}

template<typename T>
Vector<T>::~Vector() {
    delete[] data;
}

```

```
template<typename T>
void Vector<T>::push_back(const T& value) {
    if (_size == _capacity) {
        reallocate(_capacity == 0 ? 1 : _capacity * 2);
    }
    data[_size++] = value;
}

template<typename T>
void Vector<T>::push_back(T&& value) {
    if (_size == _capacity) {
        reallocate(_capacity == 0 ? 1 : _capacity * 2);
    }
    data[_size++] = std::move(value);
}

template<typename T>
void Vector<T>::pop_back() {
    if (_size > 0) {
        --_size;
    }
}

template<typename T>
size_t Vector<T>::size() const {
    return _size;
}

template<typename T>
size_t Vector<T>::capacity() const {
    return _capacity;
}

template<typename T>
bool Vector<T>::empty() const {
    return _size == 0;
}

template<typename T>
T& Vector<T>::operator[](size_t index) {
    return data[index];
}

template<typename T>
const T& Vector<T>::operator[](size_t index) const {
    return data[index];
}

template<typename T>
void Vector<T>::clear() {
    _size = 0;
}

template<typename T>
void Vector<T>::reserve(size_t newCapacity) {
    if (newCapacity > _capacity) {
        reallocate(newCapacity);
    }
}

template<typename T>
T* Vector<T>::begin() {
    return data;
}

template<typename T>
T* Vector<T>::end() {
    return data + _size;
}

template<typename T>
const T* Vector<T>::begin() const {
    return data;
}

template<typename T>
const T* Vector<T>::end() const {
    return data + _size;
}

template<typename T>
T& Vector<T>::back() {
    if (_size == 0) {
        throw std::out_of_range("Kvieciamas back() tusciam vektoriui");
    }
    return data[_size - 1];
}
```

```cpp

template<typename T>
const T& Vector<T>::back() const {
    if (_size == 0) {
        throw std::out_of_range("Kvieciamas back() tusciam vektoriui");
    }
    return data[_size - 1];
}

template<typename T>
T& Vector<T>::front() {
    if (_size == 0) {
        throw std::out_of_range("Kvieciamas front() tusciam vektoriui");
    }
    return data[0];
}

template<typename T>
const T& Vector<T>::front() const {
    if (_size == 0) {
        throw std::out_of_range("Kvieciamas front() tusciam vektoriui");
    }
    return data[0];
}

template<typename T>
T* Vector<T>::data_ptr() {
    return data;
}

template<typename T>
const T* Vector<T>::data_ptr() const {
    return data;
}

template<typename T>
void Vector<T>::erase(size_t index) {
    if (index < _size) {
        std::move(data + index + 1, data + _size, data + index);
        --_size;
    }
}

template<typename T>
void Vector<T>::resize(size_t newSize, const T& defaultValue) {
    if (newSize > _capacity) {
        reallocate(newSize);
    }
    if (newSize > _size) {
        std::fill(data + _size, data + newSize, defaultValue);
    }
    _size = newSize;
}

template<typename T>
void Vector<T>::swap(Vector& other) {
    std::swap(data, other.data);
    std::swap(_size, other._size);
    std::swap(_capacity, other._capacity);
}

template<typename T>
template<typename InputIterator>
void Vector<T>::assign(InputIterator first, InputIterator last) {
    using category = typename std::iterator_traits<InputIterator>::iterator_category;
    assign_impl(first, last, category());
}

template<typename T>
template<typename InputIterator>
void Vector<T>::assign_impl(InputIterator first, InputIterator last, std::input_iterator_tag) {
    clear();
    for (; first != last; ++first) {
        push_back(*first);
    }
}

template<typename T>
template<typename RandomAccessIterator>
void Vector<T>::assign_impl(RandomAccessIterator first, RandomAccessIterator last,
std::random_access_iterator_tag) {
    size_t newSize = std::distance(first, last);
    if (newSize > _capacity) {
        reallocate(newSize);
    }
    std::copy(first, last, data);
    _size = newSize;
}

template<typename T>
void Vector<T>::assign(std::initializer_list<T> ilist) {
    size_t newSize = ilist.size();
    if (newSize > _capacity) {
        reallocate(newSize);
    }
    std::copy(ilist.begin(), ilist.end(), data);
    _size = newSize;
}

template<typename T>
void Vector<T>::assign(size_t count, const T& value) {
    size_t newSize = count;
    if (newSize > _capacity) {
        reallocate(newSize);
    }
    std::fill_n(data, count, value);
    _size = count;
}

template<typename T>
void Vector<T>::insert(size_t index, const T& value) {
    if (_size == _capacity) {
        reallocate(_capacity == 0 ? 1 : _capacity * 2);
    }
    if (index < _size) {
        std::move_backward(data + index, data + _size, data + _size + 1);
    }
    data[index] = value;
    ++_size;
}

template<typename T>
template<typename InputIterator>
void Vector<T>::insert_range(size_t index, InputIterator first, InputIterator last) {
    size_t count = std::distance(first, last);
    if (_size + count > _capacity) {
        reallocate(_size + count);
    }
    if (index < _size) {
        std::move_backward(data + index, data + _size, data + _size + count);
    }
    std::copy(first, last, data + index);
    _size += count;
}

template<typename T>
void Vector<T>::append_range(std::initializer_list<T> ilist) {
    if (_size + ilist.size() > _capacity) {
        reallocate(_size + ilist.size());
    }
    std::copy(ilist.begin(), ilist.end(), data + _size);
    _size += ilist.size();
}

template<typename T>
T& Vector<T>::at(size_t index) {
    if (index >= _size) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

template<typename T>
const T& Vector<T>::at(size_t index) const {
    if (index >= _size) {
        throw std::out_of_range("Index out of range");
    }
    return data[index];
}

template<typename T>
typename std::reverse_iterator<T*> Vector<T>::rbegin() {
    return std::reverse_iterator<T*>(end());
}

template<typename T>
typename std::reverse_iterator<T*> Vector<T>::rend() {
    return std::reverse_iterator<T*>(begin());
}

template<typename T>
typename std::reverse_iterator<const T*> Vector<T>::rbegin() const {
    return std::reverse_iterator<const T*>(end());
}
```

```cpp

template<typename T>
typename std::reverse_iterator<const T*> Vector<T>::rend() const {
    return std::reverse_iterator<const T*>(begin());
}

template<typename T>
void Vector<T>::reallocate(size_t newCapacity) {
    T* newData = new T[newCapacity];
    if (data) {
        std::move(data, data + _size, newData);
        delete[] data;
    }
    data = newData;
    _capacity = newCapacity;
}

// Bazine ir Derived klases
class Zmogus
{
protected:
    string vardas;
    string pavarde;

public:
    Zmogus(string vard = "", string pav = "") : vardas(move(vard)), pavarde(move(pav)) {}
    virtual ~Zmogus() = default;
    virtual void patikrinimas() const = 0;
};

class Mokinys : public Zmogus
{
private:
    /* string vardas;
    string pavarde; */
    Vector<int> ND;
    int egzaminas;
    double VID;
    double MED;

public:
    // Constructor
    Mokinys(string vard = "", string pav = "", Vector<int> nd = {}, int e = 0, double vid = 0.0, double med = 0.0)
        : Zmogus(move(vard), move(pav)), ND(nd), egzaminas(e), VID(vid), MED(med) {}

    // Destructor
    ~Mokinys() = default;

    void patikrinimas() const override{};

    // Copy constructor
    Mokinys(const Mokinys &other) : Zmogus(other), ND(other.ND), egzaminas(other.egzaminas), VID(other.VID), MED(other.MED) {}

    // Move contructor
    Mokinys(Mokinys &&other) noexcept
        : Zmogus((move(other.vardas)), (move(other.pavarde))),
          ND(move(other.ND)), egzaminas(exchange(other.egzaminas, 0)),
          VID(exchange(other.VID, 0)), MED(exchange(other.MED, 0)) {}

    // Copy Assignment Operator
    Mokinys &operator=(const Mokinys &other)
    {
        Zmogus::operator=(other);
        ND = other.ND;
        egzaminas = other.egzaminas;
        VID = other.VID;
        MED = other.MED;
        return *this;
    }

    void clear() {
        vardas.clear();
        pavarde.clear();
        ND.clear();
        egzaminas = 0;
        VID = 0;
        MED = 0;
    }

    // Move Assignment Operator
        Mokinys& operator=(Mokinys&& other) noexcept {
        if (this != &other) {
            // Copy data from 'other' to 'this'
            vardas = std::move(other.vardas);
            pavarde = std::move(other.pavarde);
            ND = std::move(other.ND);
            egzaminas = other.egzaminas;
            VID = other.VID;
            MED = other.MED;

            // Clear ’other’
            other.clear();
        }
        return *this;
    }


    friend std::ostream &operator«(std::ostream &fr, const Mokinys &temp1)
    {
        fr « "Vardas: " « temp1.vardas « endl;
        fr « "Pavarde: " « temp1.pavarde « endl;
        fr « "Namu darbai: ";
        for (int pazymys : temp1.ND)
        {
            fr « pazymys « " ";
        }
        cout « endl;
        fr « "Egzamino pazymys: " « temp1.egzaminas « endl;
        fr « "Mediana: " « temp1.MED « endl;
        fr « "Vidurkis: " « temp1.VID « endl;
        return fr;
    }

    friend istream &operator»(istream &fd, Mokinys &temp1)
    {
        cout « "Iveskite varda: ";
        fd » temp1.vardas;
        cout « "Iveskite pavarde: ";
        fd » temp1.pavarde;
        cout « "Iveskite namu darbus: ";
        int pazymys;
        temp1.ND.clear();
        while (fd » pazymys && pazymys != 0)
        {
            temp1.ND.push_back(pazymys);
        }
        cout « "Iveskite egzamino pazymi: ";
        fd » temp1.egzaminas;
        cout « "Iveskite mediana: ";
        fd » temp1.MED;
        cout « "Iveskite vidurki: ";
        fd » temp1.VID;
        return fd;
    }


    // Getter functions
    string getVardas() const { return vardas; }
    string getPavarde() const { return pavarde; }
    Vector<int> getND() const { return ND; }
    int getEgzaminas() const { return egzaminas; }
    double getVID() const { return VID; }
    double getMED() const { return MED; }

    // Setter functions
    void setVardas(const string &name) { vardas = name; }
    void setPavarde(const string &surname) { pavarde = surname; }
    void addND(int nd) { ND.push_back(nd); }
    void clearND() { ND.clear(); }
    void setEgzaminas(int exam) { egzaminas = exam; }
    void setVID(double vid) { VID = vid; }
    void setMED(double med) { MED = med; }

    // Utility functions
    void Vidurkis(Vector<Mokinys> &A);
    void Isvedimas(const Vector<Mokinys> &A, int MOK_kiekis, string isvedimas);
    void Isvedimas2(const Vector<Mokinys> &A, int MOK_kiekis, string isvedimas);
    static bool PagalVidurki(const Mokinys &a, const Mokinys &b);
    static bool PagalMediana(const Mokinys &a, const Mokinys &b);
    static bool PagalVarda(const Mokinys &a, const Mokinys &b);
    static bool PagalPavarde(const Mokinys &a, const Mokinys &b);
    void Skaitymas(Vector<Mokinys> &Nuskriaustieji, Vector<Mokinys> &Mokslinciai, Vector<int> &IrasuSk, string failas, Vector<Mokinys> &A, int &temp, char strategija);
    void StudentuRusiavimas(Vector<Mokinys> &Nuskriaustieji, Vector<Mokinys> &Mokslinciai, Vector<Mokinys> &A, Vector<int> &IrasuSk, string failas, int &temp);
    void StudentuRusiavimas2(Vector<Mokinys> &Nuskriaustieji, Vector<Mokinys> &Mokslinciai, Vector<Mokinys> &A, Vector<int> &IrasuSk, string failas, int &temp);
    void StudentuRusiavimas3(Vector<Mokinys> &Nuskriaustieji, Vector<Mokinys> &Mokslinciai, Vector<Mokinys> &A, Vector<int> &IrasuSk, string failas, int &temp);
    void Rikiavimas(Vector<Mokinys> &Mokslinciai, Vector<Mokinys> &Nuskriaustieji, Vector<int> &IrasuSk);
};
```

```

#endif
```

Index