MP-Frolov-Lab-2

1.0

Создано системой Doxygen 1.9.6

Глава 1

Методы программирования. Лабораторная работа №2. Алгоритмы поиска.

1.1 Вариант_26

Массив данных генеологического фонда: ФИО, год рождения, год смерти, число детей (сравнение по полям - год рождения, ФИО, число детей).
Сортировка: пирамидальная.
Алгоритмы поиска: линейный, бинарный и встроенный в multimap

1.2 Выполнил

Фролов Олег, СКБ201

Глава 2

Алфавитный указатель классов

2.1 Классы

Классы с их кратким описанием.

Глава 3

Список файлов

3.1 Файлы

Полный список файлов.

Глава 4

Классы

4.1 Класс Element

#include <Element.h>

Открытые члены

- Element ()
- Element (std::string fam, std::string im, std::string ot, int y, int yd, int ca)
- std::string getFam ()
- void setFam (std::string newFam)
- std::string getIm ()
- void setIm (std::string newIm)
- std::string getOt ()
- void setOt (std::string newOt)
- int getYear ()
- void setYear (int newYear)
- int getYearD ()
- void setYearD (int newYearD)
- int getChildAmount ()
- void setChildAmount (int newAmount)
- bool operator> (const Element &elem) const
- bool operator< (const Element &elem) const
- bool operator>= (const Element &elem) const
- bool operator<= (const Element &elem) const
- Element & operator= (Element &elem)

4.1.1 Подробное описание

См. определение в файле Element.h строка 5

4.1.2 Конструктор(ы)

4.1.2.1 Element() [1/2]

Element::Element ( )

Стандартный конструктор.

См. определение в файле Element.cpp строка 12

```
00012                {
00013     this->familiya = "Fam";
00014     this->imya = "Im";
00015     this->otchestvo = "Ot";
00016 }
```

4.1.2.2 Element() [2/2]

```
Element::Element (
                std::string fam,
                std::string im,
                std::string ot,
                int y,
                int yd,
                int ca )
```

Конструктор, которому передаются значения для всех полей класса.

См. определение в файле Element.cpp строка 21

```
00021                                                    {
00022     this->familiya = fam;
00023     this->imya = im;
00024     this->otchestvo = ot;
00025     this->year = y;
00026     this->year_death = yd;
00027     this->child_amount = ca;
00028 }
```

4.1.3 Методы

4.1.3.1 getChildAmount()

int Element::getChildAmount ( )

Функция получения количества детей.

См. определение в файле Element.cpp строка 103

```
00103                    {
00104     return this->child_amount;
00105 }
```

4.1.3.2 getFam()

std::string Element::getFam ( )

Функция получения фамилии.

См. определение в файле Element.cpp строка 33

```
00033                       {
00034     return this->familiya;
00035 }
```

4.1.3.3 getIm()

std::string Element::getIm ( )

Функция получения имени.

См. определение в файле Element.cpp строка 47

```
00047                       {
00048     return this->imya;
00049 }
```

4.1.3.4 getOt()

std::string Element::getOt ( )

Функция получения отчества.

См. определение в файле Element.cpp строка 61

```
00061                       {
00062     return this->otchestvo;
00063 }
```

4.1.3.5 getYear()

int Element::getYear ( )

Функция получения года рождения.

См. определение в файле Element.cpp строка 75

```
00075                   {
00076     return this->year;
00077 }
```

4.1.3.6 getYearD()

int Element::getYearD ( )

Функция получения года смерти.

См. определение в файле Element.cpp строка 89

```
00089                   {
00090     return this->year_death;
00091 }
```

4.1.3.7 operator<()

bool Element::operator< (
const Element & elem) const

Перегрузка оператора < для сравнения двух объектов класса Element (стр. 7).

См. определение в файле Element.cpp строка 165

```
00165                                           {
00166     if (year < elem.year) {
00167         return 1;
00168     }
00169     else if (year > elem.year) {
00170         //цикл для проверки первенства алфавита, если равные ФИО, то далее проверка по кол-ву детей
00171         if (familiya[0] == elem.familiya[0]) {
00172             if (imya[0] == elem.imya[0]) {
00173                 if (otchestvo[0] == elem.otchestvo[0]) {
00174                     if (child_amount < elem.child_amount)
00175                         return 1;
00176                     else
00177                         return 0;
00178                 }
00179                 else {
00180                     int first = alph.find(std::tolower(otchestvo[0]));
00181                     int second = alph.find(std::tolower(elem.otchestvo[0]));
00182                     if (first > second)
00183                         return 1;
00184                     else
00185                         return 0;
00186                 }
00187             }
00188             else {
00189                 int first = alph.find(std::tolower(imya[0]));
00190                 int second = alph.find(std::tolower(elem.imya[0]));
00191                 if (first > second)
00192                     return 1;
00193                 else
00194                     return 0;
00195             }
00196         }
00197         else {
00198             int first = alph.find(std::tolower(familiya[0]));
00199             int second = alph.find(std::tolower(elem.familiya[0]));
00200             if (first > second)
00201                 return 1;
00202             else
00203                 return 0;
00204         }
00205     }
00206     else {
00207         return 0;
00208     }
00209 }
```

4.1.3.8 operator<=()

```
bool Element::operator<= (
            const  Element & elem ) const
```

Перегрузка оператора <= для сравнения двух объектов класса Element (стр. 7).

См. определение в файле Element.cpp строка 263

```
00263                                       {
00264     if (year <= elem.year) {
00265         return 1;
00266     }
00267     else if (year > elem.year) {
00268         //цикл для проверки первенства алфавита, если равные ФИО, то далее проверка по кол-ву детей
00269         if (familiya[0] == elem.familiya[0]) {
00270             if (imya[0] == elem.imya[0]) {
00271                 if (otchestvo[0] == elem.otchestvo[0]) {
00272                     if (child_amount <= elem.child_amount)
00273                         return 1;
00274                     else
00275                         return 0;
00276                 }
00277                 else {
00278                     int first = alph.find(std::tolower(otchestvo[0]));
00279                     int second = alph.find(std::tolower(elem.otchestvo[0]));
00280                     if (first >= second)
00281                         return 1;
00282                     else
00283                         return 0;
00284                 }
00285             }
00286             else {
00287                 int first = alph.find(std::tolower(imya[0]));
00288                 int second = alph.find(std::tolower(elem.imya[0]));
00289                 if (first >= second)
00290                     return 1;
00291                 else
00292                     return 0;
00293             }
00294         }
00295         else {
00296             int first = alph.find(std::tolower(familiya[0]));
00297             int second = alph.find(std::tolower(elem.familiya[0]));
00298             if (first >= second)
00299                 return 1;
00300             else
00301                 return 0;
00302         }
00303     }
00304     else {
00305         return 0;
00306     }
00307 }
```

4.1.3.9 operator=()

```
Element & Element::operator= (
            Element & elem )
```

Перегрузка оператора = для сравнения двух объектов класса Element (стр. 7).

См. определение в файле Element.cpp строка 312

```
00312                                     {
00313     this->setFam(elem.getFam());
00314     this->setIm(elem.getIm());
00315     this->setOt(elem.getOt());
00316     this->setYear(elem.getYear());
00317     this->setYearD(elem.getYearD());
00318     this->setChildAmount(elem.getChildAmount());
00319     return *this;
00320 }
```

4.1.3.10 operator>()

```
bool Element::operator> (
                const  Element & elem ) const
```

Перегрузка оператора > для сравнения двух объектов класса Element (стр. 7).

См. определение в файле Element.cpp строка 117

```
00117                                       {
00118     if (year > elem.year) {
00119         return 1;
00120     }
00121     else if (year == elem.year) {
00122         //цикл для проверки первенства алфавита, если равные ФИО, то далее проверка по кол-ву детей
00123         if (familiya[0] == elem.familiya[0]) {
00124             if (imya[0] == elem.imya[0]) {
00125                 if (otchestvo[0] == elem.otchestvo[0]) {
00126                     if (child_amount > elem.child_amount)
00127                         return 1;
00128                     else
00129                         return 0;
00130                 }
00131                 else {
00132                     int first = alph.find(std::tolower(otchestvo[0]));
00133                     int second = alph.find(std::tolower(elem.otchestvo[0]));
00134                     if (first < second)
00135                         return 1;
00136                     else
00137                         return 0;
00138                 }
00139             }
00140             else {
00141                 int first = alph.find(std::tolower(imya[0]));
00142                 int second = alph.find(std::tolower(elem.imya[0]));
00143                 if (first < second)
00144                     return 1;
00145                 else
00146                     return 0;
00147             }
00148         }
00149         else {
00150             int first = alph.find(std::tolower(familiya[0]));
00151             int second = alph.find(std::tolower(elem.familiya[0]));
00152             if (first < second)
00153                 return 1;
00154             else
00155                 return 0;
00156         }
00157     } else {
00158         return 0;
00159     }
00160 }
```

4.1.3.11 operator>=()

```
bool Element::operator>= (
                const  Element & elem ) const
```

Перегрузка оператора >= для сравнения двух объектов класса Element (стр. 7).

См. определение в файле Element.cpp строка 214

```
00214                                       {
00215     if (year >= elem.year) {
00216         return 1;
00217     }
00218     else if (year < elem.year) {
00219         //цикл для проверки первенства алфавита, если равные ФИО, то далее проверка по кол-ву детей
00220         if (familiya[0] == elem.familiya[0]) {
00221             if (imya[0] == elem.imya[0]) {
00222                 if (otchestvo[0] == elem.otchestvo[0]) {
00223                     if (child_amount >= elem.child_amount)
00224                         return 1;
00225                     else
```

```
00226                     return 0;
00227                 }
00228                 else {
00229                     int first = alph.find(std::tolower(otchestvo[0]));
00230                     int second = alph.find(std::tolower(elem.otchestvo[0]));
00231                     if (first <= second)
00232                         return 1;
00233                     else
00234                         return 0;
00235                 }
00236             }
00237             else {
00238                 int first = alph.find(std::tolower(imya[0]));
00239                 int second = alph.find(std::tolower(elem.imya[0]));
00240                 if (first <= second)
00241                     return 1;
00242                 else
00243                     return 0;
00244             }
00245         }
00246         else {
00247             int first = alph.find(std::tolower(familiya[0]));
00248             int second = alph.find(std::tolower(elem.familiya[0]));
00249             if (first <= second)
00250                 return 1;
00251             else
00252                 return 0;
00253         }
00254     }
00255     else {
00256         return 0;
00257     }
00258 }
```

4.1.3.12 setChildAmount()

```
void Element::setChildAmount (
                int newAmount )
```

Функция установки количества детей.

См. определение в файле Element.cpp строка 110

```
00110                                   {
00111     this->child_amount = newAmount;
00112 }
```

4.1.3.13 setFam()

```
void Element::setFam (
                std::string newFam )
```

Функция установки фамилии.

См. определение в файле Element.cpp строка 40

```
00040                                  {
00041     this->familiya = newFam;
00042 }
```

4.1.3.14 setIm()

```
void Element::setIm (
                std::string newIm )
```

Функция установки имени.

См. определение в файле Element.cpp строка 54

```
00054                                {
00055     this->imya= newIm;
00056 }
```

4.1.3.15 setOt()

```
void Element::setOt (
                std::string newOt )
```

Функция установки отчества.

См. определение в файле Element.cpp строка 68

```
00068                                {
00069     this->otchestvo = newOt;
00070 }
```

4.1.3.16 setYear()

```
void Element::setYear (
                int newYear )
```

Функция установки года рождения.

См. определение в файле Element.cpp строка 82

```
00082                                {
00083     this->year = newYear;
00084 }
```

4.1.3.17 setYearD()

```
void Element::setYearD (
                int newYearD )
```

Функция установки года смерти.

См. определение в файле Element.cpp строка 96

```
00096                                {
00097     this->year_death = newYearD;
00098 }
```

Объявления и описания членов классов находятся в файлах:

- Element.h
- Element.cpp

Глава 5

Файлы

5.1 Файл Element.cpp

```
#include "Element.h"
#include <string>
```

Переменные

- std::string alph = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"

5.1.1 Переменные

5.1.1.1 alph

std::string alph = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя"

Кириллица для сравнения ФИО.

См. определение в файле Element.cpp строка 7

5.2 Element.cpp

См. документацию.

```
00001 #include "Element.h"
00002 #include <string>
00003
00007 std::string alph = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";
00008
00012 Element::Element() {
00013     this->familiya = "Fam";
00014     this->imya = "Im";
00015     this->otchestvo = "Ot";
00016 }
00017
00021 Element::Element(std::string fam, std::string im, std::string ot, int y, int yd, int ca) {
00022     this->familiya = fam;
00023     this->imya = im;
00024     this->otchestvo = ot;
00025     this->year = y;
00026     this->year_death = yd;
00027     this->child_amount = ca;
00028 }
00029
00033 std::string Element::getFam() {
00034     return this->familiya;
00035 }
00036
00040 void Element::setFam(std::string newFam) {
00041     this->familiya = newFam;
00042 }
00043
00047 std::string Element::getIm() {
00048     return this->imya;
00049 }
00050
00054 void Element::setIm(std::string newIm) {
00055     this->imya= newIm;
00056 }
00057
00061 std::string Element::getOt() {
00062     return this->otchestvo;
00063 }
00064
00068 void Element::setOt(std::string newOt) {
00069     this->otchestvo = newOt;
00070 }
00071
00075 int Element::getYear() {
00076     return this->year;
00077 }
00078
00082 void Element::setYear(int newYear) {
00083     this->year = newYear;
00084 }
00085
00089 int Element::getYearD() {
00090     return this->year_death;
00091 }
00092
00096 void Element::setYearD(int newYearD) {
00097     this->year_death = newYearD;
00098 }
00099
00103 int Element::getChildAmount() {
00104     return this->child_amount;
00105 }
00106
00110 void Element::setChildAmount(int newAmount) {
00111     this->child_amount = newAmount;
00112 }
00113
00117 bool Element::operator> (const Element& elem) const {
00118     if (year > elem.year) {
00119         return 1;
00120     }
00121     else if (year == elem.year) {
00122         //цикл для проверки первенства алфавита, если равные ФИО, то далее проверка по кол-ву детей
00123         if (familiya[0] == elem.familiya[0]) {
00124             if (imya[0] == elem.imya[0]) {
00125                 if (otchestvo[0] == elem.otchestvo[0]) {
00126                     if (child_amount > elem.child_amount)
00127                         return 1;
00128                     else
00129                         return 0;
00130                 }
```

```
00131                 else {
00132                     int first = alph.find(std::tolower(otchestvo[0]));
00133                     int second = alph.find(std::tolower(elem.otchestvo[0]));
00134                     if (first < second)
00135                         return 1;
00136                     else
00137                         return 0;
00138                 }
00139             }
00140             else {
00141                 int first = alph.find(std::tolower(imya[0]));
00142                 int second = alph.find(std::tolower(elem.imya[0]));
00143                 if (first < second)
00144                     return 1;
00145                 else
00146                     return 0;
00147             }
00148         }
00149         else {
00150             int first = alph.find(std::tolower(familiya[0]));
00151             int second = alph.find(std::tolower(elem.familiya[0]));
00152             if (first < second)
00153                 return 1;
00154             else
00155                 return 0;
00156         }
00157     } else {
00158         return 0;
00159     }
00160 }
00161
00165 bool Element::operator< (const Element& elem) const {
00166     if (year < elem.year) {
00167         return 1;
00168     }
00169     else if (year > elem.year) {
00170         //цикл для проверки первенства алфавита, если равные ФИО, то далее проверка по кол-ву детей
00171         if (familiya[0] == elem.familiya[0]) {
00172             if (imya[0] == elem.imya[0]) {
00173                 if (otchestvo[0] == elem.otchestvo[0]) {
00174                     if (child_amount < elem.child_amount)
00175                         return 1;
00176                     else
00177                         return 0;
00178                 }
00179                 else {
00180                     int first = alph.find(std::tolower(otchestvo[0]));
00181                     int second = alph.find(std::tolower(elem.otchestvo[0]));
00182                     if (first > second)
00183                         return 1;
00184                     else
00185                         return 0;
00186                 }
00187             }
00188             else {
00189                 int first = alph.find(std::tolower(imya[0]));
00190                 int second = alph.find(std::tolower(elem.imya[0]));
00191                 if (first > second)
00192                     return 1;
00193                 else
00194                     return 0;
00195             }
00196         }
00197         else {
00198             int first = alph.find(std::tolower(familiya[0]));
00199             int second = alph.find(std::tolower(elem.familiya[0]));
00200             if (first > second)
00201                 return 1;
00202             else
00203                 return 0;
00204         }
00205     }
00206     else {
00207         return 0;
00208     }
00209 }
00210
00214 bool Element::operator>= (const Element& elem) const {
00215     if (year >= elem.year) {
00216         return 1;
00217     }
00218     else if (year < elem.year) {
00219         //цикл для проверки первенства алфавита, если равные ФИО, то далее проверка по кол-ву детей
00220         if (familiya[0] == elem.familiya[0]) {
00221             if (imya[0] == elem.imya[0]) {
00222                 if (otchestvo[0] == elem.otchestvo[0]) {
00223                     if (child_amount >= elem.child_amount)
```

```
00224                     return 1;
00225                 else
00226                     return 0;
00227             }
00228             else {
00229                 int first = alph.find(std::tolower(otchestvo[0]));
00230                 int second = alph.find(std::tolower(elem.otchestvo[0]));
00231                 if (first <= second)
00232                     return 1;
00233                 else
00234                     return 0;
00235             }
00236         }
00237         else {
00238             int first = alph.find(std::tolower(imya[0]));
00239             int second = alph.find(std::tolower(elem.imya[0]));
00240             if (first <= second)
00241                 return 1;
00242             else
00243                 return 0;
00244         }
00245     }
00246     else {
00247         int first = alph.find(std::tolower(familiya[0]));
00248         int second = alph.find(std::tolower(elem.familiya[0]));
00249         if (first <= second)
00250             return 1;
00251         else
00252             return 0;
00253     }
00254 }
00255 else {
00256     return 0;
00257 }
00258 }
00259
00263 bool Element::operator<= (const Element& elem) const {
00264     if (year <= elem.year) {
00265         return 1;
00266     }
00267     else if (year > elem.year) {
00268         //цикл для проверки первенства алфавита, если равные ФИО, то далее проверка по кол-ву детей
00269         if (familiya[0] == elem.familiya[0]) {
00270             if (imya[0] == elem.imya[0]) {
00271                 if (otchestvo[0] == elem.otchestvo[0]) {
00272                     if (child_amount <= elem.child_amount)
00273                         return 1;
00274                     else
00275                         return 0;
00276                 }
00277                 else {
00278                     int first = alph.find(std::tolower(otchestvo[0]));
00279                     int second = alph.find(std::tolower(elem.otchestvo[0]));
00280                     if (first >= second)
00281                         return 1;
00282                     else
00283                         return 0;
00284                 }
00285             }
00286             else {
00287                 int first = alph.find(std::tolower(imya[0]));
00288                 int second = alph.find(std::tolower(elem.imya[0]));
00289                 if (first >= second)
00290                     return 1;
00291                 else
00292                     return 0;
00293             }
00294         }
00295         else {
00296             int first = alph.find(std::tolower(familiya[0]));
00297             int second = alph.find(std::tolower(elem.familiya[0]));
00298             if (first >= second)
00299                 return 1;
00300             else
00301                 return 0;
00302         }
00303     }
00304     else {
00305         return 0;
00306     }
00307 }
00308
00312 Element& Element::operator= (Element& elem) {
00313     this->setFam(elem.getFam());
00314     this->setIm(elem.getIm());
00315     this->setOt(elem.getOt());
00316     this->setYear(elem.getYear());
```

```
00317     this->setYearD(elem.getYearD());
00318     this->setChildAmount(elem.getChildAmount());
00319     return *this;
00320 }
```

5.3 Файл Element.h

#include <iostream>
#include <string>

Классы

- class Element

5.4 Element.h

См. документацию.

```
00001 #pragma once
00002 #include <iostream>
00003 #include <string>
00004
00005 class Element {
00006 private:
00007     std::string familiya = "";
00008     std::string imya = "";
00009     std::string otchestvo = "";
00010     int year = 1900;
00011     int year_death = 1920;
00012     int child_amount = 0;
00013     int sex = 0;
00014 public:
00015     Element();
00016     Element(std::string fam, std::string im, std::string ot, int y, int yd, int ca);
00017     std::string getFam();
00018     void setFam(std::string newFam);
00019     std::string getIm();
00020     void setIm(std::string newIm);
00021     std::string getOt();
00022     void setOt(std::string newOt);
00023     int getYear();
00024     void setYear(int newYear);
00025     int getYearD();
00026     void setYearD(int newYearD);
00027     int getChildAmount();
00028     void setChildAmount(int newAmount);
00029     bool operator> (const Element& elem) const;
00030     bool operator< (const Element& elem) const;
00031     bool operator>= (const Element& elem) const;
00032     bool operator<= (const Element& elem) const;
00033     Element& operator= (Element& elem);
00034 };
```

5.5 Файл Main.cpp

#include <iostream>
#include <fstream>
#include "Element.h"
#include <vector>
#include <chrono>
#include <cstdlib>
#include <map>
#include <sstream>

Функции

- void findMultimap (std::multimap< std::string, int >map, std::string searchQuery)
- void linearSearch (std::vector< Element > spisok, std::string elem)
- int checkTwoStrings (std::string firstS, std::string secondS)
- int checkFIO (Element elem1, Element elem2, int flag)
- void binarySearch1 (std::vector< Element > spisok, std::string query)
- void heapify1 (std::vector< Element > &spisok, int N, int i)
- void heapSort1 (std::vector< Element > &spisok, int N)
- int main ()

5.5.1 Функции

5.5.1.1 binarySearch1()

```
void binarySearch1 (
                std::vector< Element > spisok,
                std::string query )
```

Функция для бинарного поиска заданного элемента в контейнере объектов класса Element (стр. 7).

См. определение в файле Main.cpp строка 128

```
00128                                                {
00129     int low = 0;
00130     int high = spisok.size() - 1;
00131     int mid;
00132
00133     std::stringstream ss(query);
00134     std::vector<std::string> words;
00135     std::string word;
00136     Element elem;
00137     while (ss » word) {
00138         words.push_back(word);
00139     }
00140     elem.setFam(words[0]);
00141     elem.setIm(words[1]);
00142     elem.setOt(words[2]);
00143     while (low <= high) {
00144         mid = (high + low) / 2;
00145         if (checkFIO(spisok[mid], elem, 2)==2) {
00146             std::cout « query « " was found at index " « mid « "\n";
00147             break;
00148         }
00149         else if (checkFIO(spisok[mid], elem, 0)) {
00150             high = mid - 1;
00151         }
00152         else {
00153             low = mid + 1;
00154         }
00155     }
00156     if (low > high)
00157         std::cout « query « " is not in this array!\n";
00158 }
```

5.5.1.2 checkFIO()

```
int checkFIO (
                Element elem1,
                Element elem2,
               int flag )
```

Функция для проверки равенства ФИО двух объектов класса Element (стр. 7).

См. определение в файле Main.cpp строка 87

```
00087                                       {
00088     std::string alph = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";
00089     std::string fio = "";
00090     std::string fio2 = "";
00091     fio = elem1.getFam();
00092     fio2 = elem2.getFam();
00093     int first = alph.find(std::tolower(fio[0]));
00094     int second = alph.find(std::tolower(fio2[0]));
00095     if (first == second) {
00096         for (int i = 0; i < fio.length() && i < fio2.length() && first == second; ++i) {
00097             first = alph.find(std::tolower(fio[i]));
00098             second = alph.find(std::tolower(fio2[i]));
00099         }
00100     }
00101     if (first == second) {
00102         fio = elem1.getIm();
00103         fio2 = elem2.getIm();
00104         for (int i = 0; i < fio.length() && i < fio2.length() && first == second; ++i) {
00105             first = alph.find(std::tolower(fio[i]));
00106             second = alph.find(std::tolower(fio2[i]));
00107         }
00108     }
00109     if (first == second) {
00110         fio = elem1.getOt();
00111         fio2 = elem2.getOt();
00112         for (int i = 0; i < fio.length() && i < fio2.length() && first == second; ++i) {
00113             first = alph.find(std::tolower(fio[i]));
00114             second = alph.find(std::tolower(fio2[i]));
00115         }
00116     }
00117     if (first > second)
00118         return 1;
00119     else if (flag == 2 && first == second)
00120         return 2;
00121     else
00122         return 0;
00123 }
```

5.5.1.3 checkTwoStrings()

```
int checkTwoStrings (
                std::string firstS,
                std::string secondS )
```

Функция для сравнения двух строк по кириллице. Если в ходе цикла, пробегающегося по всем буквам двух строк,
в одном из двух слов встречаются разные буквы, то та строка, чья буква встречается в алфавите раньше, считается большей.

См. определение в файле Main.cpp строка 60

```
00060                                       {
00061     std::string alph = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";
00062     //if true return 1, if false return 0
00063     int first = alph.find(std::tolower(firstS[0]));
00064     int second = alph.find(std::tolower(secondS[0]));
00065     if (first == second) {
00066         int count = 0;
00067 
00068         while (first == second) {
00069             first = alph.find(std::tolower(firstS[count]));
```

```
00070             second = alph.find(std::tolower(secondS[count]));
00071             firstS.insert(count, " ");
00072             ++count;
00073         }
00074         if (first < second) {
00075             return 0;
00076         } else
00077             return 1;
00078     } else if (first < second)
00079         return 0;
00080     else
00081         return 1;
00082 }
```

5.5.1.4 findMultimap()

```
void findMultimap (
                std::multimap< std::string, int > map,
                std::string searchQuery )
```

Функция для поиска заданного элемента в multimap, который содержит объекты класса Element (стр. 7).

См. определение в файле Main.cpp строка 25

```
00025                                                   {
00026     auto search = map.find(searchQuery);
00027     if (search != map.end()) {
00028         std::cout « "\nFound " « search->first « " at " « search->second « "\n";
00029     }
00030     else {
00031         std::cout « "\nNot Found\n";
00032     }
00033 }
```

5.5.1.5 heapify1()

```
void heapify1 (
                std::vector< Element > & spisok,
                int N,
                int i )
```

Функция для построения кучи, требуемой для корректной работы пирамидальной сортировки.

См. определение в файле Main.cpp строка 163

```
00163                                           {
00164     int largest = i;
00165     int left = 2 * i + 1;
00166     int right = 2 * i + 2;
00167     if (left<N && /*spisok[left]>spisok[largest]*/checkFIO(spisok[left], spisok[largest], 0)==1)
00168         largest = left;
00169     if (right<N && /*spisok[right]>spisok[largest]*/checkFIO(spisok[right], spisok[largest], 0)==1)
00170         largest = right;
00171     if (largest != i) {
00172         Element temp;
00173         temp = spisok[i];
00174         spisok[i] = spisok[largest];
00175         spisok[largest] = temp;
00176         heapify1(spisok, N, largest);
00177     }
00178 }
```

5.5.1.6 heapSort1()

```
void heapSort1 (
                std::vector< Element > & spisok,
                int N )
```

Функция для пирамидальной сортировки контейнера из объектов класса Element (стр. 7).

См. определение в файле Main.cpp строка 183

```
00183                                                {
00184     for (int i = N / 2 - 1; i >= 0; --i) {
00185         heapify1(spisok, N, i);
00186     }
00187     for (int i = N - 1; i >= 0; --i) {
00188         Element temp;
00189         temp = spisok[0];
00190         spisok[0] = spisok[i];
00191         spisok[i] = temp;
00192         heapify1(spisok, i, 0);
00193     }
00194 }
```

5.5.1.7 linearSearch()

```
void linearSearch (
                std::vector< Element > spisok,
                std::string elem )
```

Функция для прямого поиска заданного элемента в контейнере объектов класса Element (стр. 7).

См. определение в файле Main.cpp строка 38

```
00038                                                    {
00039     for (int i = 0; i < spisok.size(); ++i) {
00040         std::string fio = "";
00041         fio.append(spisok[i].getFam());
00042         fio.append(" ");
00043         fio.append(spisok[i].getIm());
00044         fio.append(" ");
00045         fio.append(spisok[i].getOt());
00046         if (fio == elem) {
00047             std::cout « elem « " was found at index " « i « "\n";
00048             break;
00049         }
00050         else if (i + 1 == spisok.size()) {
00051             std::cout « elem « " is not in this array!\n";
00052         }
00053     }
00054 }
```

5.5.1.8 main()

```
int main ( )
```

См. определение в файле Main.cpp строка 196

```
00196           {
00197
00198     std::system("chcp 1251");
00199     int file_number = 0;
00200     std::vector<Element> spisok;
00201     std::multimap<std::string, int> map;
00202     int choosedAlg = 0;
00203     std::cout « "\nChoose search algorithm (0 - linear (already sorted), 1 - binar (already sorted), 2 - sort and binar, 3 -
        multimap (search by key)): \n";
00204     std::cin » choosedAlg;
```

```
00205     while (choosedAlg < 0 || choosedAlg >= 4 || std::cin.fail()) {
00206         if (std::cin.fail()) {
00207             std::cin.clear();
00208             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00209             std::cout « "Your input is not a number!" « '\n';
00210             choosedAlg = -1;
00211         }
00212         else {
00213             std::cout « "Your input must be in interval 0-3!" « '\n';
00214             choosedAlg = 0;
00215         }
00216         std::cout « "Input again!" « '\n';
00217         std::cin » choosedAlg;
00218     }
00219     std::cout « "Choose file(0-7)" « '\n';
00220     std::cin » file_number;
00221     while (file_number < 0 || file_number >= 8 || std::cin.fail()) {
00222         if (std::cin.fail()) {
00223             std::cin.clear();
00224             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00225             std::cout « "Your input is not a number!" « '\n';
00226             file_number = -1;
00227         }
00228         else {
00229             std::cout « "Your input must be in interval 0-7!" « '\n';
00230             file_number = 0;
00231         }
00232         std::cout « "Input again!" « '\n';
00233         std::cin » file_number;
00234     }
00235     setlocale(LC_ALL, "rus");
00236     std::string line = "";
00237     std::string filename = "C:\\Users\\fov-2\\source\\repos\\data_";
00238     filename.append(std::to_string(file_number));
00239     filename.append(".txt");
00240     std::ifstream in(filename);
00241     if (in.is_open()) {
00242         while (getline(in, line)) {
00243             std::string word = "";
00244             int flag = 0;
00245             int amount = 0;
00246             Element human;
00247             for (int i = 0; i < line.size(); ++i) {
00248                 if (line[i] == '(')
00249                     flag = 2;
00250                 else if (line[i] == ')')
00251                     flag = 0;
00252                 else if (line[i] == ' ' || line[i] == '\n') {
00253                     if (amount == 0) {
00254                         human.setFam(word);
00255                     }
00256                     else if (amount == 1)
00257                         human.setIm(word);
00258                     else if (amount == 2)
00259                         human.setOt(word);
00260                     else if (amount == 3)
00261                         human.setYear(std::stoi(word));
00262                     else if (amount == 4)
00263                         human.setYearD(std::stoi(word));
00264                     word = "";
00265                     ++amount;
00266                 }
00267                 else if (flag == 0) {
00268                     char symbol = line[i];
00269                     if (symbol != ' ') {
00270                         word.push_back(symbol);
00271                     }
00272                 }
00273
00274                 if (amount == 5) {
00275                     amount = 0;
00276                     human.setChildAmount(line[line.size() - 1] - '0');
00277                     spisok.push_back(human);
00278                 }
00279             }
00280         }
00281     }
00282     in.close();
00283     //----
00284     for (int i = 0; i < spisok.size(); ++i) {
00285         std::string fio = "";
00286         fio.append(spisok[i].getFam());
00287         fio.append(" ");
00288         fio.append(spisok[i].getIm());
00289         fio.append(" ");
00290         fio.append(spisok[i].getOt());
00291         map.insert(std::make_pair(fio, i));
```

```
00292     }
00293     //----
00294     std::cout « spisok.size() « std::endl;
00295     std::cout « "\nParsing done!\n" « std::endl;
00296 
00297     while (choosedAlg != -2) {
00298         std::string searchQuery;
00299         std::cin.get();
00300         std::cout « "\n\nInput your search query, please!\n";
00301         getline(std::cin, searchQuery);
00302         std::cout « "\n" « searchQuery « "\n";
00303         if (choosedAlg == 0) {
00304             //check time of linear search!
00305             std::cout « "\n\nLinear searching started!\n";
00306             auto begin = std::chrono::steady_clock::now();
00307             linearSearch(spisok, searchQuery);
00308             auto end = std::chrono::steady_clock::now();
00309             auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
00310             std::cout « "The time: " « elapsed_ms.count() « " ms";
00311             std::cout « "\nLinear searching ended!\n\n";
00312         }
00313         else if (choosedAlg == 1) {
00314             //sort and check time of binary search!
00315             std::vector<Element> res = spisok;
00316             heapSort1(res, res.size());
00317             std::cout « "\n\nBinary searching started!\n";
00318             auto begin = std::chrono::steady_clock::now();
00319             binarySearch1(res, searchQuery);
00320             auto end = std::chrono::steady_clock::now();
00321             auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
00322             std::cout « "The time: " « elapsed_ms.count() « " ms";
00323             std::cout « "\nBinary searching ended!\n\n";
00324         }
00325         else if (choosedAlg == 2) {
00326             //check time of sort and binary search both!
00327             std::cout « "\nHeap started!\n";
00328             auto begin = std::chrono::steady_clock::now();
00329             std::vector<Element> res = spisok;
00330             heapSort1(res, res.size());
00331             std::cout « "\Heap ended!\n";
00332             std::cout « "\nBinary searching started!\n";
00333             binarySearch1(res, searchQuery);
00334             std::cout « "\nBinary searching ended!\n";
00335             auto end = std::chrono::steady_clock::now();
00336             auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
00337             std::cout « "The time: " « elapsed_ms.count() « " ms\n\n";
00338         }
00339         else if (choosedAlg == 3) {
00340             //find in multimap by key
00341             auto begin = std::chrono::steady_clock::now();
00342             findMultimap(map, searchQuery);
00343             auto end = std::chrono::steady_clock::now();
00344             auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
00345             std::cout « "The time: " « elapsed_ms.count() « " ms";
00346         }
00347         std::cout « "\n\nEnd!\n\n";
00348         std::cout « "Would you like to search again (0 - linear (already sorted), 1 - binary (already sorted), 2 - sort and
      binary, 3 - multimap and search by key,  -2 - end)?\n";
00349         std::cin » choosedAlg;
00350         std::cout « "\nAgain.\n";
00351     }
00352     return 0;
00353 }
```

5.6 Main.cpp

См. документацию.

```
00001 #include <iostream>
00002 #include <fstream>
00003 #include "Element.h"
00004 #include <vector>
00005 #include <chrono>
00006 #include <cstdlib>
00007 #include <map>
00008 #include <sstream>
00009 
00025 void findMultimap(std::multimap<std::string, int>map, std::string searchQuery) {
00026     auto search = map.find(searchQuery);
00027     if (search != map.end()) {
00028         std::cout « "\nFound " « search->first « " at " « search->second « "\n";
00029     }
00030     else {
```

```
00031         std::cout « "\nNot Found\n";
00032     }
00033 }
00034
00038 void linearSearch(std::vector<Element> spisok, std::string elem) {
00039     for (int i = 0; i < spisok.size(); ++i) {
00040         std::string fio = "";
00041         fio.append(spisok[i].getFam());
00042         fio.append(" ");
00043         fio.append(spisok[i].getIm());
00044         fio.append(" ");
00045         fio.append(spisok[i].getOt());
00046         if (fio == elem) {
00047             std::cout « elem « " was found at index " « i « "\n";
00048             break;
00049         }
00050         else if (i + 1 == spisok.size()) {
00051             std::cout « elem « " is not in this array!\n";
00052         }
00053     }
00054 }
00055
00060 int checkTwoStrings(std::string firstS, std::string secondS) {
00061     std::string alph = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";
00062     //if true return 1, if false return 0
00063     int first = alph.find(std::tolower(firstS[0]));
00064     int second = alph.find(std::tolower(secondS[0]));
00065     if (first == second) {
00066         int count = 0;
00067
00068         while (first == second) {
00069             first = alph.find(std::tolower(firstS[count]));
00070             second = alph.find(std::tolower(secondS[count]));
00071             firstS.insert(count, " ");
00072             ++count;
00073         }
00074         if (first < second) {
00075             return 0;
00076         } else
00077             return 1;
00078     } else if (first < second)
00079         return 0;
00080     else
00081         return 1;
00082 }
00083
00087 int checkFIO(Element elem1, Element elem2, int flag) {
00088     std::string alph = "абвгдеёжзийклмнопрстуфхцчшщъыьэюя";
00089     std::string fio = "";
00090     std::string fio2 = "";
00091     fio = elem1.getFam();
00092     fio2 = elem2.getFam();
00093     int first = alph.find(std::tolower(fio[0]));
00094     int second = alph.find(std::tolower(fio2[0]));
00095     if (first == second) {
00096         for (int i = 0; i < fio.length() && i < fio2.length() && first == second; ++i) {
00097             first = alph.find(std::tolower(fio[i]));
00098             second = alph.find(std::tolower(fio2[i]));
00099         }
00100     }
00101     if (first == second) {
00102         fio = elem1.getIm();
00103         fio2 = elem2.getIm();
00104         for (int i = 0; i < fio.length() && i < fio2.length() && first == second; ++i) {
00105             first = alph.find(std::tolower(fio[i]));
00106             second = alph.find(std::tolower(fio2[i]));
00107         }
00108     }
00109     if (first == second) {
00110         fio = elem1.getOt();
00111         fio2 = elem2.getOt();
00112         for (int i = 0; i < fio.length() && i < fio2.length() && first == second; ++i) {
00113             first = alph.find(std::tolower(fio[i]));
00114             second = alph.find(std::tolower(fio2[i]));
00115         }
00116     }
00117     if (first > second)
00118         return 1;
00119     else if (flag == 2 && first == second)
00120         return 2;
00121     else
00122         return 0;
00123 }
00124
00128 void binarySearch1(std::vector<Element> spisok, std::string query) {
00129     int low = 0;
00130     int high = spisok.size() - 1;
```

```
00131     int mid;
00132
00133     std::stringstream ss(query);
00134     std::vector<std::string> words;
00135     std::string word;
00136     Element elem;
00137     while (ss » word) {
00138         words.push_back(word);
00139     }
00140     elem.setFam(words[0]);
00141     elem.setIm(words[1]);
00142     elem.setOt(words[2]);
00143     while (low <= high) {
00144         mid = (high + low) / 2;
00145         if (checkFIO(spisok[mid], elem, 2)==2) {
00146             std::cout « query « " was found at index " « mid « "\n";
00147             break;
00148         }
00149         else if (checkFIO(spisok[mid], elem, 0)) {
00150             high = mid - 1;
00151         }
00152         else {
00153             low = mid + 1;
00154         }
00155     }
00156     if (low > high)
00157         std::cout « query « " is not in this array!\n";
00158 }
00159
00163 void heapify1(std::vector<Element>& spisok, int N, int i) {
00164     int largest = i;
00165     int left = 2 * i + 1;
00166     int right = 2 * i + 2;
00167     if (left<N && /*spisok[left]>spisok[largest]*/checkFIO(spisok[left], spisok[largest], 0)==1)
00168         largest = left;
00169     if (right<N && /*spisok[right]>spisok[largest]*/checkFIO(spisok[right], spisok[largest], 0)==1)
00170         largest = right;
00171     if (largest != i) {
00172         Element temp;
00173         temp = spisok[i];
00174         spisok[i] = spisok[largest];
00175         spisok[largest] = temp;
00176         heapify1(spisok, N, largest);
00177     }
00178 }
00179
00183 void heapSort1(std::vector<Element>& spisok, int N) {
00184     for (int i = N / 2 - 1; i >= 0; --i) {
00185         heapify1(spisok, N, i);
00186     }
00187     for (int i = N - 1; i >= 0; --i) {
00188         Element temp;
00189         temp = spisok[0];
00190         spisok[0] = spisok[i];
00191         spisok[i] = temp;
00192         heapify1(spisok, i, 0);
00193     }
00194 }
00195
00196 int main() {
00197
00198     std::system("chcp 1251");
00199     int file_number = 0;
00200     std::vector<Element> spisok;
00201     std::multimap<std::string, int> map;
00202     int choosedAlg = 0;
00203     std::cout « "\nChoose search algorithm (0 - linear (already sorted), 1 - binar (already sorted), 2 - sort and binar, 3 -
        multimap (search by key)): \n";
00204     std::cin » choosedAlg;
00205     while (choosedAlg < 0 || choosedAlg >= 4 || std::cin.fail()) {
00206         if (std::cin.fail()) {
00207             std::cin.clear();
00208             std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00209             std::cout « "Your input is not a number!" « '\n';
00210             choosedAlg = -1;
00211         }
00212         else {
00213             std::cout « "Your input must be in interval 0-3!" « '\n';
00214             choosedAlg = 0;
00215         }
00216         std::cout « "Input again!" « '\n';
00217         std::cin » choosedAlg;
00218     }
00219     std::cout « "Choose file(0-7)" « '\n';
00220     std::cin » file_number;
00221     while (file_number < 0 || file_number >= 8 || std::cin.fail()) {
00222         if (std::cin.fail()) {
```

```
00223           std::cin.clear();
00224           std::cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
00225           std::cout << "Your input is not a number!" << '\n';
00226           file_number = -1;
00227       }
00228       else {
00229           std::cout << "Your input must be in interval 0-7!" << '\n';
00230           file_number = 0;
00231       }
00232       std::cout << "Input again!" << '\n';
00233       std::cin >> file_number;
00234   }
00235   setlocale(LC_ALL, "rus");
00236   std::string line = "";
00237   std::string filename = "C:\\Users\\fov-2\\source\\repos\\data_";
00238   filename.append(std::to_string(file_number));
00239   filename.append(".txt");
00240   std::ifstream in(filename);
00241   if (in.is_open()) {
00242       while (getline(in, line)) {
00243           std::string word = "";
00244           int flag = 0;
00245           int amount = 0;
00246           Element human;
00247           for (int i = 0; i < line.size(); ++i) {
00248               if (line[i] == '(')
00249                   flag = 2;
00250               else if (line[i] == ')')
00251                   flag = 0;
00252               else if (line[i] == ' ' || line[i] == '\n') {
00253                   if (amount == 0) {
00254                       human.setFam(word);
00255                   }
00256                   else if (amount == 1)
00257                       human.setIm(word);
00258                   else if (amount == 2)
00259                       human.setOt(word);
00260                   else if (amount == 3)
00261                       human.setYear(std::stoi(word));
00262                   else if (amount == 4)
00263                       human.setYearD(std::stoi(word));
00264                   word = "";
00265                   ++amount;
00266               }
00267               else if (flag == 0) {
00268                   char symbol = line[i];
00269                   if (symbol != ' ') {
00270                       word.push_back(symbol);
00271                   }
00272               }
00273
00274               if (amount == 5) {
00275                   amount = 0;
00276                   human.setChildAmount(line[line.size() - 1] - '0');
00277                   spisok.push_back(human);
00278               }
00279           }
00280       }
00281   }
00282   in.close();
00283   //----
00284   for (int i = 0; i < spisok.size(); ++i) {
00285       std::string fio = "";
00286       fio.append(spisok[i].getFam());
00287       fio.append(" ");
00288       fio.append(spisok[i].getIm());
00289       fio.append(" ");
00290       fio.append(spisok[i].getOt());
00291       map.insert(std::make_pair(fio, i));
00292   }
00293   //----
00294   std::cout << spisok.size() << std::endl;
00295   std::cout << "\nParsing done!\n" << std::endl;
00296
00297   while (choosedAlg != -2) {
00298       std::string searchQuery;
00299       std::cin.get();
00300       std::cout << "\n\nInput your search query, please!\n";
00301       getline(std::cin, searchQuery);
00302       std::cout << "\n" << searchQuery << "\n";
00303       if (choosedAlg == 0) {
00304           //check time of linear search!
00305           std::cout << "\n\nLinear searching started!\n";
00306           auto begin = std::chrono::steady_clock::now();
00307           linearSearch(spisok, searchQuery);
00308           auto end = std::chrono::steady_clock::now();
00309           auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
```

```
00310             std::cout « "The time: " « elapsed_ms.count() « " ms";
00311             std::cout « "\nLinear searching ended!\n\n";
00312         }
00313         else if (choosedAlg == 1) {
00314             //sort and check time of binary search!
00315             std::vector<Element> res = spisok;
00316             heapSort1(res, res.size());
00317             std::cout « "\n\nBinary searching started!\n";
00318             auto begin = std::chrono::steady_clock::now();
00319             binarySearch1(res, searchQuery);
00320             auto end = std::chrono::steady_clock::now();
00321             auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
00322             std::cout « "The time: " « elapsed_ms.count() « " ms";
00323             std::cout « "\nBinary searching ended!\n\n";
00324         }
00325         else if (choosedAlg == 2) {
00326             //check time of sort and binary search both!
00327             std::cout « "\nHeap started!\n";
00328             auto begin = std::chrono::steady_clock::now();
00329             std::vector<Element> res = spisok;
00330             heapSort1(res, res.size());
00331             std::cout « "\Heap ended!\n";
00332             std::cout « "\nBinary searching started!\n";
00333             binarySearch1(res, searchQuery);
00334             std::cout « "\nBinary searching ended!\n";
00335             auto end = std::chrono::steady_clock::now();
00336             auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
00337             std::cout « "The time: " « elapsed_ms.count() « " ms\n\n";
00338         }
00339         else if (choosedAlg == 3) {
00340             //find in multimap by key
00341             auto begin = std::chrono::steady_clock::now();
00342             findMultimap(map, searchQuery);
00343             auto end = std::chrono::steady_clock::now();
00344             auto elapsed_ms = std::chrono::duration_cast<std::chrono::milliseconds>(end - begin);
00345             std::cout « "The time: " « elapsed_ms.count() « " ms";
00346         }
00347         std::cout « "\n\nEnd!\n\n";
00348         std::cout « "Would you like to search again (0 - linear (already sorted), 1 - binary (already sorted), 2 - sort and
      binary, 3 - multimap and search by key,  -2 - end)?\n";
00349         std::cin » choosedAlg;
00350         std::cout « "\nAgain.\n";
00351     }
00352     return 0;
00353 }
```

Предметный указатель