|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ ИУ - Информатика и системы управления

КАФЕДРА ИУ5

**Отчёт по проектно – технологической практике**

Студент Барышников Михаил Игоревич

*фамилия, имя, отчество*

Группа 22Б

Тип практики учебная

Название предприятия кафедра ИУ5

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Барышников М.И.**

*подпись, дата фамилия, и.о.*

Руководитель практики **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Козлов А.Д.**

*подпись, дата фамилия, и.о.*

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*2020 г.*

**Оглавление**

[**Введение** 2](#_Toc41758566)

[**Индивидуальное задание** 3](#_Toc41758567)

[**Анализ результатов** 4](#_Toc41758568)

[**Заключение** 5](#_Toc41758569)

[**Приложение** 6](#_Toc41758570)

# **Введение**

Цель выполнения данной работы – улучшить навыки разработки объектно-ориентированных программ, начать изучение стандартной библиотеки шаблонов на примере контейнера list (список).

# **Индивидуальное задание**

Написать программу, моделирующую управление каталогом в файловой системе.

Для каждого файла в каталоге содержатся следующие сведения: имя файла, дата создания, количество обращений к файлу. База хранится в текстовом файле, его размер может быть произвольным.

Программа должна обеспечивать:

□ начальное формирование каталога файлов в виде списка;

□ вывод каталога файлов;

□ обращение к файлу;

□ удаление файлов, дата создания которых раньше заданной;

□ выборку файла с наибольшим количеством обращений.

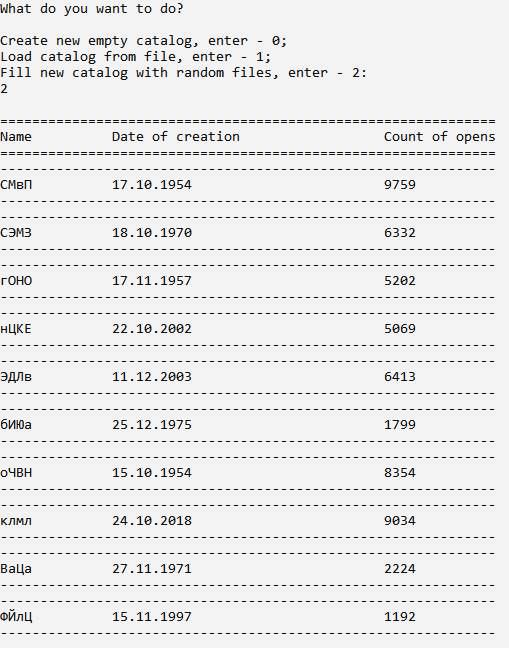
Выбор моделируемой функции должен осуществляться с помощью меню. Для представления базы в ОП использовать класс list, реализующий линейный двусвязный список.

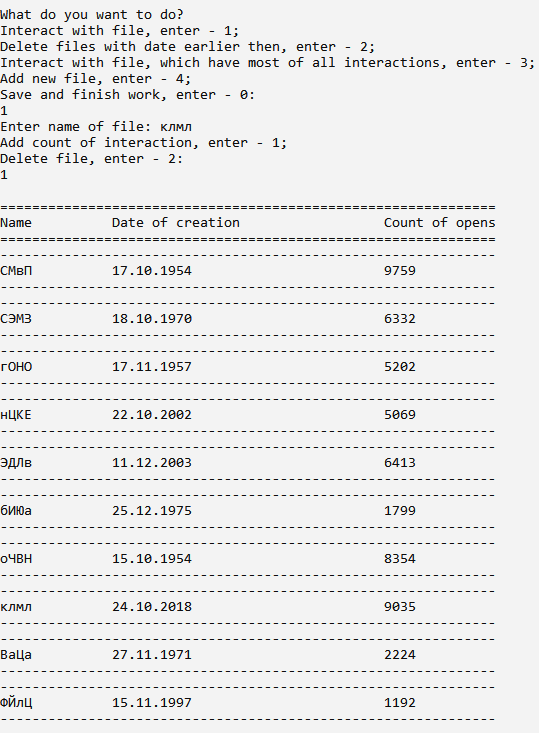
# **Анализ результатов**

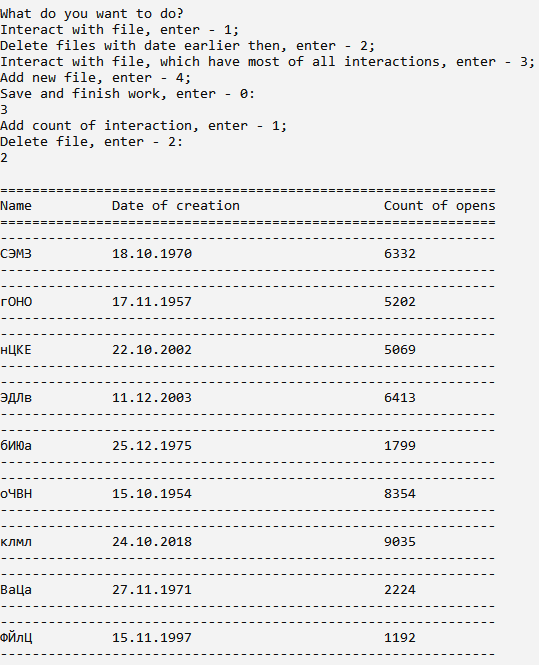
Текст программы находится в приложении.

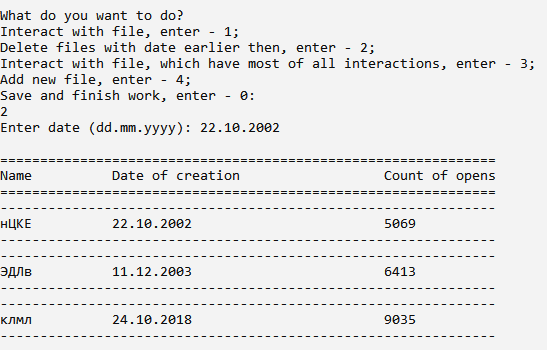
К функциям присутвующим в техническом задании, была добавлена возможность формирования случайного списка.

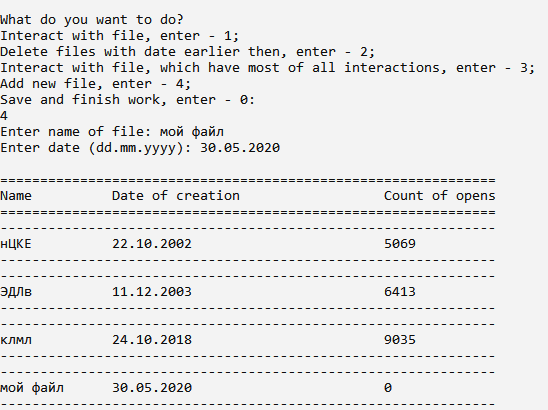
Результаты после выполнения программы получились следующие:

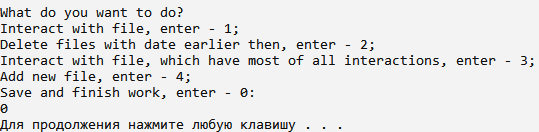


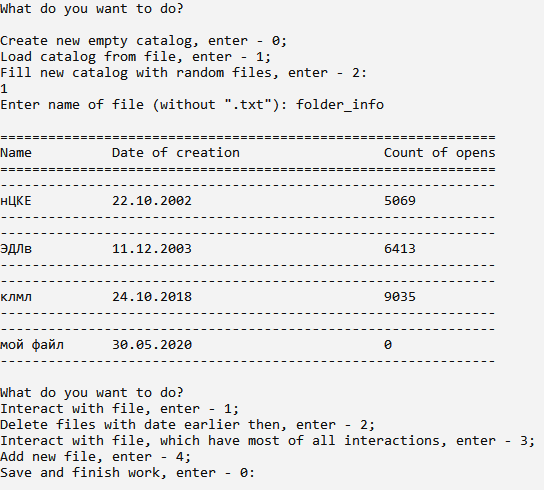












# **Заключение**

В результате реализации данной программы, по моделированию взаимодействия с файловым каталогом, был получен опыт создания шаблонных классов. А именно, был реализован контейнер list (список) – двусвязный список. Также для взаимодействия с ним был изучен класс iterator, который был так же реализован в данной программе. Опыт, полученный при изучении интерфейса iterator, пригодится в дальнейшем при разработке шаблонных классов, контейнеров и интерфейсов.

# **Приложение**

* main.cpp

#include "main\_functions.h"

int main()

{

setlocale(0, "");

SetConsoleCP(1251);

SetConsoleOutputCP(1251);

std::string name\_of\_file = "folder\_info", str;

std::fstream file;

list<FileInfo> catalog;

unsigned short int action;

while (!file.is\_open())

{

std::cout << "What do you want to do? \n\nCreate new empty catalog, enter - 0; \nLoad catalog from file, enter - 1; \nFill new catalog with random files, enter - 2:\n";

std::cin >> action;

std::cin.ignore();

switch (action)

{

case 1:

std::cout << "Enter name of file (without \".txt\"): ";

std::getline(std::cin, name\_of\_file);

file.open(name\_of\_file + ".txt", std::ios::in);

if (file.is\_open())

while (!file.eof())

{

std::getline(file, str);

if (str != "")

catalog.push\_back(get\_file\_info(str));

}

else

std::cout << "This file not found!\n\n";

break;

case 0:

file.open(name\_of\_file + ".txt", std::ios::out);

break;

case 2:

file.open(name\_of\_file + ".txt", std::ios::out);

srand(time(NULL));

for (int i = 0; i != 10; ++i)

{

str = "";

for (int j = 0; j != 4; ++j)

str += (char)(rand() % 48 - 64);

str += '|';

str += std::to\_string(rand() % 19 + 10) + '.';

str += std::to\_string(rand() % 3 + 10) + '.';

str += std::to\_string(rand() % 70 + 1950);

str += '|';

str += std::to\_string(rand() % 10000);

str += '|';

catalog.push\_back(get\_file\_info(str));

}

break;

}

}

file.close();

action = 1;

while (action)

{

list<FileInfo>::iterator it;

bool choice = false;

stream\_out\_folder\_info(catalog);

std::cout << "\nWhat do you want to do?\n";

std::cout << "Interact with file, enter - 1; \nDelete files with date earlier then, enter - 2; \nInteract with file, which have most of all interactions, enter - 3; \nAdd new file, enter - 4; \nSave and finish work, enter - 0:\n";

std::cin >> action;

std::cin.ignore();

switch (action)

{

case 4:

{

FileInfo\* f = new FileInfo;

std::cout << "Enter name of file: ";

std::getline(std::cin, str);

f->name = str;

std::cout << "Enter date (dd.mm.yyyy): ";

std::getline(std::cin, str);

f->date\_of\_creation = str;

f->count\_of\_open = 0;

catalog.push\_back(\*f);

break;

}

case 2:

if (!catalog.empty())

{

std::cout << "Enter date (dd.mm.yyyy): ";

std::getline(std::cin, str);

Date now = get\_date(str);

for (it = catalog.begin(); it != catalog.end();)

if (date\_to\_age(now, get\_date((\*it).date\_of\_creation)) > 0)

catalog.erase(it);

else

++it;

}

break;

case 3:

if (!catalog.empty())

{

choice = true;

it = catalog.begin();

int max = (\*it).count\_of\_open;

for (list<FileInfo>::iterator iter = catalog.begin(); iter != catalog.end(); ++iter)

if ((\*iter).count\_of\_open > max)

{

max = (\*iter).count\_of\_open;

it = iter;

}

}

else

{

std::cout << "Catalog is empty!";

}

;

case 1:

if (action == 1)

{

std::cout << "Enter name of file: ";

std::getline(std::cin, str);

for (it = catalog.begin(); it != catalog.end() && !(choice = (\*it).name == str); ++it);

}

if (choice)

{

std::cout << "Add count of interaction, enter - 1; \nDelete file, enter - 2:\n";

std::cin >> action;

std::cin.ignore();

switch (action)

{

case 1:

++(\*it).count\_of\_open;

break;

case 2:

catalog.erase(it);

break;

}

}

}

}

file.open(name\_of\_file + ".txt", std::ios::out);

if (file.is\_open())

for (list<FileInfo>::iterator it = catalog.begin(); it != catalog.end(); ++it)

file << (\*it).name + '|' + (\*it).date\_of\_creation + '|' + std::to\_string((\*it).count\_of\_open) + '|' + '\n';

system("pause");

return 0;

}

* main\_functions.h

#pragma once

#include "class\_list.h"

#include <iostream>

#include <fstream>

#include <string>

#include <windows.h>

#include <iomanip>

#include <cstdlib>

#include <ctime>

struct Date

{

int day;

int month;

int year;

};

struct FileInfo

{

std::string name;

std::string date\_of\_creation;

int count\_of\_open;

};

FileInfo get\_file\_info(std::string &str)

{

FileInfo f;

f.name = str.substr(0, str.find\_first\_of('|'));

str.erase(0, str.find\_first\_of('|') + 1);

f.date\_of\_creation = str.substr(0, str.find\_first\_of('|'));

str.erase(0, str.find\_first\_of('|') + 1);

f.count\_of\_open = atoi(str.substr(0, str.find\_first\_of('|')).c\_str());

str.erase(0, str.find\_first\_of('|') + 1);

return f;

}

void stream\_out\_folder\_info(const list<FileInfo> &folder)

{

std::cout << "\n==============================================================\n";

std::cout << "Name Date of creation Count of opens";

std::cout << "\n==============================================================\n";

if (!folder.empty())

{

for (list<FileInfo>::iterator it = folder.begin(); it != folder.end(); ++it)

{

std::cout << "--------------------------------------------------------------\n";

std::cout << std::left << (\*it).name << std::right << std::setw(24 - (\*it).name.length()) << (\*it).date\_of\_creation << std::left << " " << (\*it).count\_of\_open << "\n";

std::cout << "--------------------------------------------------------------\n";

}

}

else

{

std::cout << "No any folders here!\n";

}

}

double date\_to\_age(Date comp, Date practicants)

{

int mon[] = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };

double year = comp.year - practicants.year, days = comp.day - practicants.day;

int k = 0;

if (comp.month < practicants.month)

k = 1;

while (comp.month != practicants.month)

{

comp.month += -1 + 2 \* k;

days += (1 - 2 \* k) \* mon[comp.month - k - 1];

if ((!(comp.year % 4) && (comp.year % 100) || !(comp.year % 400)) && (comp.month - k == 2))

days += 1 - 2 \* k;

}

if (!(comp.year % 4) && (comp.year % 100) || !(comp.year % 400))

year += days / 366;

else

year += days / 365;

return year;

}

Date get\_date(std::string str)

{

Date d;

d.day = atoi(str.substr(0, str.find\_first\_of('.')).c\_str());

str.erase(0, str.find\_first\_of('.') + 1);

d.month = atoi(str.substr(0, str.find\_first\_of('.')).c\_str());

str.erase(0, str.find\_first\_of('.') + 1);

d.year = atoi(str.substr(0, str.length()).c\_str());

str.erase(0, str.length());

return d;

}

* class\_list.h

#pragma once

template <class ValueType>

class list;

template <typename ValueType>

class iterator;

template <class ValueType>

class element

{

friend class list<ValueType>;

friend class iterator<ValueType>;

ValueType elem;

element\* top;

element\* bot;

};

template <class ValueType>

class list

{

element<ValueType>\* front\_ptr;

element<ValueType>\* back\_ptr;

public:

typedef const iterator<ValueType> const\_iterator;

typedef iterator<ValueType> iterator;

list();

~list();

list(const list<ValueType>&);

list<ValueType>& operator=(const list<ValueType>&);

void erase(iterator&);

void push\_back(ValueType);

void push\_front(ValueType);

bool empty() const;

ValueType pop\_back();

ValueType pop\_front();

const\_iterator begin() const;

const\_iterator end() const;

iterator begin();

iterator end();

};

template <typename ValueType>

class iterator

{

friend class list<ValueType>;

mutable element<ValueType>\* ptr;

public:

iterator() : ptr(nullptr) {};

iterator(element<ValueType>\* elem) : ptr(elem) {}

const iterator& operator=(const iterator& it) const

{

ptr = it.ptr;

return \*this;

}

iterator& operator=(const iterator& it)

{

ptr = it.ptr;

return \*this;

}

ValueType operator\*() const

{

if (!ptr->bot)

throw std::exception("Out of range!");

return ptr->elem;

}

ValueType& operator\*()

{

if (!ptr->bot)

throw std::exception("Out of range!");

return ptr->elem;

}

const iterator operator++() const {

if (!ptr->bot)

throw std::exception("Out of range!");

else

ptr = ptr->bot;

return \*this;

}

iterator operator++() {

if (!ptr->bot)

throw std::exception("Out of range!");

else

ptr = ptr->bot;

return \*this;

}

const iterator operator--() const {

if (!ptr->top)

throw std::exception("Out of range!");

else

ptr = ptr->top;

return \*this;

}

iterator operator--() {

if (!ptr->top)

throw std::exception("Out of range!");

else

ptr = ptr->top;

return \*this;

}

bool operator==(const iterator& b) const

{

return ptr == b.ptr;

}

bool operator!=(const iterator& b) const

{

return ptr != b.ptr;

}

};

#include "methods\_class\_list.inl"

* methods\_class\_list.inl

#include "class\_list.h"

template <class ValueType>

inline list<ValueType>::list()

{

front\_ptr = new element<ValueType>;

back\_ptr = front\_ptr;

front\_ptr->bot = nullptr;

front\_ptr->top = nullptr;

//front\_ptr->elem = NULL;

}

template<class ValueType>

inline list<ValueType>::~list()

{

while (back\_ptr)

{

element<ValueType> \*temp = back\_ptr;

back\_ptr = back\_ptr->top;

delete temp;

}

}

template<class ValueType>

inline list<ValueType>::list(const list<ValueType> &main\_list)

{

front\_ptr = new element<ValueType>;

back\_ptr = front\_ptr;

front\_ptr->bot = nullptr;

front\_ptr->top = nullptr;

//front\_ptr->elem = NULL;

element<ValueType> \*el = main\_list.front\_ptr;

while (el->bot)

{

push\_back(el->elem);

el = el->bot;

}

}

template<class ValueType>

inline list<ValueType>& list<ValueType>::operator=(const list<ValueType> &main\_list)

{

front\_ptr = new element<ValueType>;

back\_ptr = front\_ptr;

front\_ptr->bot = nullptr;

front\_ptr->top = nullptr;

//front\_ptr->elem = NULL;

element<ValueType> \*el = main\_list.front\_ptr;

while (el->bot)

{

push\_back(el->elem);

el = el->bot;

}

return \*this;

}

template<class ValueType>

inline void list<ValueType>::erase(iterator &it\_cur)

{

if (empty())

throw std::exception("Out of range!");

it\_cur.ptr->bot->top = it\_cur.ptr->top;

if (it\_cur.ptr->top)

it\_cur.ptr->top->bot = it\_cur.ptr->bot;

else

front\_ptr = it\_cur.ptr->bot;

element<ValueType> \*temp = it\_cur.ptr;

it\_cur.ptr = it\_cur.ptr->bot;

delete temp;

}

template<class ValueType>

inline void list<ValueType>::push\_back(ValueType count)

{

element<ValueType> \*temp = new element<ValueType>;

temp->top = back\_ptr->top;

temp->bot = back\_ptr;

temp->elem = count;

back\_ptr->top = temp;

if (back\_ptr == front\_ptr)

front\_ptr = temp;

else

temp->top->bot = temp;

}

template<class ValueType>

inline void list<ValueType>::push\_front(ValueType count)

{

element<ValueType> \*temp = new element;

temp->bot = front\_ptr;

temp->top = front\_ptr->top; // можно заменить на nullptr

temp->elem = count;

front\_ptr->top = temp;

front\_ptr = temp;

}

template<class ValueType>

inline bool list<ValueType>::empty() const

{

return !front\_ptr->bot;

}

template<class ValueType>

inline ValueType list<ValueType>::pop\_back()

{

if (empty())

throw std::exception("Out of range!");

ValueType el = back\_ptr->top->elem;

element<ValueType> \*temp = back\_ptr->top;

back\_ptr->top = back\_ptr->top->top;

if (!back\_ptr->top)

front\_ptr = back\_ptr;

else

back\_ptr->top->bot = back\_ptr;

delete temp;

return el;

}

template<class ValueType>

inline ValueType list<ValueType>::pop\_front()

{

if (empty())

throw std::exception("Out of range!");

ValueType el = front\_ptr->elem;

element<ValueType> \*temp = front\_ptr;

front\_ptr = front\_ptr->bot;

front\_ptr->top = temp->top; // можно заменить на nullptr

delete temp;

return el;

}

template<class ValueType>

inline typename const iterator<ValueType> list<ValueType>::begin() const

{

return iterator(front\_ptr);

}

template<class ValueType>

inline typename const iterator<ValueType> list<ValueType>::end() const

{

return iterator(back\_ptr);

}

template<class ValueType>

inline typename iterator<ValueType> list<ValueType>::begin()

{

return iterator(front\_ptr);

}

template<class ValueType>

inline typename iterator<ValueType> list<ValueType>::end()

{

return iterator(back\_ptr);

}