Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования

**Национальный исследовательский университет «МЭИ»**

**Типовой расчет на тему**

***«Моделирование типов. Ссылочные структуры»***

**Вариант 10**

*Задание выполнил: Дарья Константиновна Гусева*

*Студент группы A-12-19*

*Проверил: Воробьева.И.А.*

*Оценка:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

*Замечания:\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

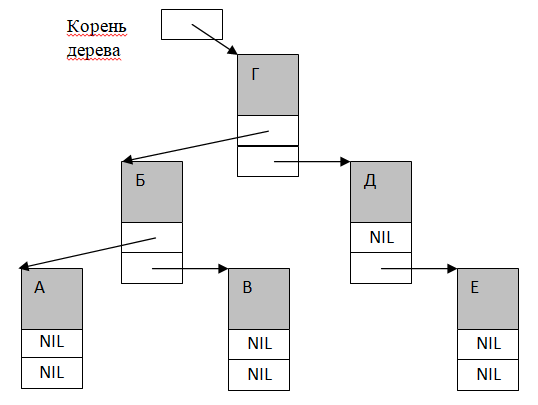
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Москва 2019 НИУ «МЭИ»

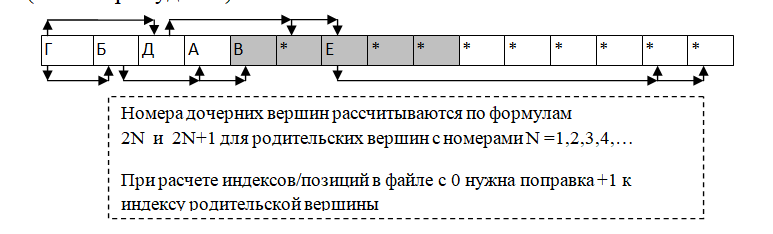
**Часть 1. Создать новый тип «бинарное дерево» и написать базовые операции**

Бинарное дерево — иерархическая структура данных, в которой каждый узел имеет не более двух потомков (детей). Как правило, первый называется родительским узлом, а дети называются левым и правым наследниками. Двоичное дерево не является упорядоченным ориентированным деревом.

1. **Бинарное дерево на основе ссылочного типа «Нелинейный связный список»**



1. **Бинарное дерево на основе одномерного динамического массива.**



Дерево состоит из структури хранящей контейнер для элементов (одномерный динамический массив), индекс текущего элемента и все необходимые методы.

**Новый тип данных:**

Структура Values хранит информацию о человеке, содержит 6 полей типа std::string и одно логическое поле типа boolean, а также функцию (конструктор) для присваивания значении этим полям.

struct Values

{

std::string m\_MiddleName;

std::string m\_Name;

std::string m\_LastName;

std::string m\_Date;

std::string m\_Death;

std::string m\_Place;

bool m\_Printed = false;

Values() {}

Values(Values\* values)

{

m\_MiddleName = values->m\_MiddleName;

m\_Name = values->m\_Name;

m\_LastName = values->m\_LastName;

m\_Date = values->m\_Date;

if (!values->m\_Death.empty())

m\_Death = values->m\_Death;

if (!values->m\_Place.empty())

m\_Place = values->m\_Place;

}

};

**Определим структуру Human – элемент дерева на основе нелинейного списка:**

struct Human {

//информация о человеке

std::string m\_MiddleName; //имя

std::string m\_Name; //фамилия

std::string m\_LastName; //отчество

std::string m\_Date; //дата рождения

std::string m\_Death; //дата смерти

std::string m\_Place; //место рождения

bool m\_Printed; //вспомогательное логическое поле для вывода информации

//связы

Human\* m\_Left; //левое поддерево

Human\* m\_Right; //правое поддерево

Human\* m\_Parent; //родительская вершина

//конструктор (функция для инициализации данных)

Human(const Values& values, Human\* left, Human\* right, Human\* parent)

{

m\_MiddleName = values.m\_MiddleName;

m\_Name = values.m\_Name;

m\_LastName = values.m\_LastName;

m\_Date = values.m\_Date;

if (!values.m\_Death.empty())

m\_Death = values.m\_Death;

if (!values.m\_Place.empty())

m\_Place = values.m\_Place;

m\_Left = left;

m\_Right = right;

m\_Parent = parent;

m\_Printed = values.m\_Printed;

}

};

**Структуру - элемент дерева на основе динамического массива:**

*Структуру для динамического массива не надо определять так как принцип работы дерева на основании массива требует только структуру Tree – дерево хранящую этот массив и номер текущего элемента, а также все необходимые методы для работы с данным деревом. Массив состоит из элементов типа Values.*

**Структура Tree – дерево, для работы с нелинейным списком**

struct Tree {

Human\* m\_First; //указатель на первый элемент дерева (корень)

Human\* m\_Current; //указатель на текущий элемент дерева

//возвращает левое поддерево

Human\* CheckLeft();

//возвращает правое поддерево

Human\* CheckRight();

//Возвращает текущее значение

Human\* CheckCurrent();

//проверит на существование первого элемента

bool CheckFirst();

//создание пустого дерева

void CreateFirst(const Values& value);

//сделать текущим корень дерева

void Root();

//сделать текущим левое поддерево

void Left();

//сделать текущим правое поддерево

void Right();

//сделать текущим родительскую вершину для текущей вершины

void Parent();

//узнать значение текущей вершины

void GetValue(const char& c);

//изменит значение имени текущей вершины

void ChangeName(std::string\* name);

//изменит значение отчества текущей вершины

void ChangeSurname(std::string\* middleName);

//изменит значение фамилии текущей вершины

void ChangeLastName(std::string\* lastName);

//изменит значение даты рождения текущей вершины

void ChangeDate(std::string\* date);

//изменит значение даты смерти текущей вершины

void ChangeDeath(std::string\* death);

//изменит значение места рождения текущей вершины

void ChangePlace(std::string\* place);

//изменить вся значения текущей вершины

void ChangeAll(const Values& human);

//создать левое поддерево для текущей вершины

void AddLeft(const Values& human);

//создать правое поддерево для текущей вершины

void AddRight(const Values& human);

//проверка на пустоту дерева(поддерева, начинающегося с текущей вершины)

bool CheckIfEmpty();

//удалить дерево

void Delete();

//поставить указатель текущего элемента на самый левый элемент дерева

void LeftMost();

//создание упорядоченного по ключу дерева из файла

void CreateFromFile();

//сравнивание двух дат

bool CompareDates(const std::string\* date\_root, const std::string\* date\_new);

//добавление нового элемента по ключу в интерактивном режиме

void AddByKey(const Values& human);

//вывод дерева от самого старшего до самого младшего

void PrintByAge();

//возвращает значение логического поля m\_Printed

bool Printed();

//вывод дерева Корень-ЛевоеПоддерево-ПравоеПоддерево

void PrintRoot();

//восстановление значения m\_Print к false для всех элементов дерева

void ResetPrint();

//создание генеалогического дерева из файла

void GenTree();

//вывод всех предков кто родился в Москве и прожил не менее 70 лет

void SearchGen();

};

**Структура Tree – дерево, для работы с динамическим массивом**

struct Tree {

//MAX\_ARRAY\_SIZE = 100

Values \*m\_Values[MAX\_ARRAY\_SIZE]; //массив хранящий элементы дерева

~Tree()

{

delete[] m\_Values; //удаление массива

}

//текущий элемент

unsigned int m\_Current = 0;

//проверит на существование первого элемента

bool CheckFirst();

//возвращает левое поддерево

Values\* CheckLeft();

//возвращает правое поддерево

Values\* CheckRight();

//возвращает текущий элемент

Values\* CheckCurrent();

//проверит на существование первого элемента

bool CheckFirst();

//создание пустого дерева

void CreateFirst(const Values& value);

//сделать текущим корень дерева

void Root();

//сделать текущим левое поддерево

void Left();

//сделать текущим правое поддерево

void Right();

//сделать текущим родительскую вершину для текущей вершины

void Parent();

//узнать значение текущей вершины

void GetValue(const char& c);

//изменит значение имени текущей вершины

void ChangeName(std::string\* name);

//изменит значение отчества текущей вершины

void ChangeSurname(std::string\* middleName);

//изменит значение фамилии текущей вершины

void ChangeLastName(std::string\* lastName);

//изменит значение даты рождения текущей вершины

void ChangeDate(std::string\* date);

//изменит значение даты смерти текущей вершины

void ChangeDeath(std::string\* death);

//изменит значение места рождения текущей вершины

void ChangePlace(std::string\* place);

//изменить вся значения текущей вершины

void ChangeAll(const Values& human);

//создать левое поддерево для текущей вершины

void AddLeft(const Values& human);

//создать правое поддерево для текущей вершины

void AddRight(const Values& human);

//проверка на пустоту дерева(поддерева, начинающегося с текущей вершины)

bool CheckIfEmpty();

//удалить дерево

void Delete();

//поставить указатель текущего элемента на самый левый элемент дерева

void LeftMost();

//создание упорядоченного по ключу дерева из файла

void CreateFromFile();

//сравнивание двух дат

bool CompareDates(const std::string\* date\_root, const std::string\* date\_new);

//добавление нового элемента по ключу в интерактивном режиме

void AddByKey(const Values& human);

//вывод дерева от самого старшего до самого младшего

void PrintByAge();

//возвращает значение логического поля m\_Printed

bool Printed();

//вывод дерева Корень-ЛевоеПоддерево-ПравоеПоддерево

void PrintRoot();

//восстановление значения m\_Print к false для всех элементов дерева

void ResetPrint();

//создание генеалогического дерева из файла

void GenTree();

//вывод всех предков кто родился в Москве и прожил не менее 70 лет

void SearchGen();

};

**>>Описание базовых операций**

//проверка на пустоту дерева(поддерева, начинающегося с текущей вершины) - список;

bool Tree::CheckIfEmpty()//проверяем существуют ли левый и правый элемент

{

//если существует хотя бы один поддерево не пустое – в противном пустое

return (m\_Current->m\_Left != nullptr || m\_Current->m\_Right != nullptr) ? false : true;

}

//проверка на пустоту дерева(поддерева, начинающегося с текущей вершины) - массив;

bool Tree::CheckIfEmpty()//аналогично проверки в списке

{

return (CheckLeft() != nullptr || CheckRight() != nullptr) ? false : true;

}

//- создание дерева из одного элемента - список

void Tree::CreateFirst(const Values& human) //передаем ссылку на элемент содержащий

{ //только поля с значениям

//проверяем существует ли первый элемент

if (m\_First == nullptr) { //если не существует

//создаем новый элемент типа Human с левым, правым и предыдущим указателям на nullptr

m\_First = new Human(human, nullptr, nullptr, nullptr);

m\_Current = m\_First; //присваиваем текущему эл-ту первый элемент

return; //закончиваем

} //если уже существует выводим сообщение об его существованию

printf("Уже существует корень!\n");

}

//- создание дерева из одного элемента - массив

void Tree::CreateFirst(const Values& human) //все аналогично кроме 2ой строки тела функции

{

if (m\_Values[0] == nullptr) {

//присваиваем первому элементу массива новый элемент

m\_Values[0] = new Values(human);

m\_Current = 0;

return;

}

printf("Уже существует корень!\n");

}

//сделать текущим по адресу в списке корень дерева – список

void Tree::Root()

{

if (m\_First != nullptr) {//проверяем существует ли этот элемент

m\_Current = m\_First; //присваиваем указателю на текущий эл-т первый

}

}

//сделать текущим по адресу в списке корень дерева – массив

void Tree::Root()

{

if (m\_Values[0] != nullptr) {

m\_Current = 0; //присваиваем текущему эл-ту первый

}

}

//- сделать текущим левое поддерево (от текущей вершины) – список

void Tree::Left()

{

if (m\_Current->m\_Left != nullptr) {//проверяем существует ли левый эл-т

m\_Current = m\_Current->m\_Left; //присваиваем указателю на текущий

return; //указатель на левый эл-т

}

}

//- сделать текущим левое поддерево (от текущей вершины) – массив

void Tree::Left()

{//проверяем существует ли левый эл-т, если существует меняем значение m\_Current

//на значение m\_Current \* 2 + 1. В противном ничего не делаем

m\_Current = CheckLeft() != nullptr ? (m\_Current \* 2) + 1 : m\_Current;

}

//- сделать текущим правое поддерево (от текущей вершины) – список

void Tree::Right()

{

if (m\_Current->m\_Right != nullptr) {//проверяем существует ли правый эл-т

m\_Current = m\_Current->m\_Right; //присваиваем указателю на текущий

return; //указатель на правый эл-т

}

}

//- сделать текущим правое поддерево (от текущей вершины) – массив

void Tree::Right()

{//проверяем существует ли правый эл-т, если существует меняем значение m\_Current

//на значение m\_Current \* 2 + 2. В противном ничего не делаем

m\_Current = CheckRight() != nullptr ? (m\_Current \* 2) + 2 : m\_Current;

}

//сделать текущим родительскую вершину для текущей вершины - список

void Tree::Parent()

{

if (m\_Current->m\_Parent != nullptr) {//проверяем существует ли родительский эл-т

m\_Current = m\_Current->m\_Parent; //присваиваем указателю на текущий

return; //указатель на родительский эл-т

}

}

//сделать текущим родительскую вершину для текущей вершины - массив

void Tree::Parent()

{

if (m\_Current != 0) //проверяем чтобы текущий эл-т не был равен первому

//если не равен проверяем m\_Current на чётность

//в случае когда чётный присваиваем значение (m\_Current – 2) / 2

//в случае когда нечётный присваиваем значение (m\_Current – 1) / 2

m\_Current = m\_Current % 2 == 0 ? (m\_Current - 2) / 2 : (m\_Current - 1) / 2;

else //в противном ничего не делаем

m\_Current = 0;

}

//узнать значение текущей вершины - список

void Tree::GetValue(const char& c)

{

if (c == 'a') //проверяем мод (если выводим только одну вершину не меняем значение

m\_Current->m\_Printed = true; //m\_Printed // если выводим целое дерево меняем

printf("Имя: %s\n", m\_Current->m\_Name.c\_str());//вывод имени

printf("Отчество: %s\n", m\_Current->m\_MiddleName.c\_str());//вывод отчества

printf("Фамилия: %s\n", m\_Current->m\_LastName.c\_str());//вывод фамилии

printf("Дата рождения: %s\n", m\_Current->m\_Date.c\_str());//вывод даты рождения

if (!m\_Current->m\_Death.empty())//если не пустая строка

printf("Дата смерти: %s\n", m\_Current->m\_Death.c\_str());//вывод даты смерти

if (!m\_Current->m\_Place.empty())//если не пустая строка

printf("Место рождения: %s\n", m\_Current->m\_Place.c\_str());//вывод места рож

}

//узнать значение текущей вершины - массив

void Tree::GetValue(const char& c) // все аналогично только обращение к элементам не

{ //по ссылке а по индексу в массиве

if (c == 'a')

CheckCurrent()->m\_Printed = true;

printf("Имя: %s\n", m\_Values[m\_Current]->m\_Name.c\_str());

printf("Отчество: %s\n", m\_Values[m\_Current]->m\_MiddleName.c\_str());

printf("Фамилия: %s\n", m\_Values[m\_Current]->m\_LastName.c\_str());

printf("Дата рождения: %s\n", m\_Values[m\_Current]->m\_Date.c\_str());

if (!m\_Values[m\_Current]->m\_Death.empty())

printf("Дата смерти: %s\n", m\_Values[m\_Current]->m\_Death.c\_str());

if (!m\_Values[m\_Current]->m\_Place.empty())

printf("Место рождения: %s\n", m\_Values[m\_Current]->m\_Place.c\_str());

}

//изменить значение имени текущей вершины - список

void Tree::ChangeName(std::string\* name) //передаем новое значение по указатель

{

m\_Current->m\_Name = \*name; //меняем текущее значение на новое

printf("Изменено значение имени!\n");//выводим сообщение об изменении

}

//изменить значение имени текущей вершины - массив

void Tree::ChangeName(std::string\* name) //все аналогично, только обращение не по ссылкам

{

m\_Values[m\_Current]->m\_Name = \*name; //а по индексу в массиве

printf("Изменено значение имени!\n");

}

!!!!!!!!!!!!!!!!!СЛЕДУЮЩИЕ ФУНКЦИИ АНАЛОГИЧНИ!!!!!!!!!!!!!!!!!!

//изменить значение отчества текущей вершины - список

void Tree::ChangeSurname(std::string\* middleName)

{

m\_Current->m\_MiddleName = \*middleName;

printf("Изменено значение отчества!\n");

}

//изменить значение отчества текущей вершины - массив

void Tree::ChangeSurname(std::string\* middleName)

{

m\_Values[m\_Current]->m\_MiddleName = \*middleName;

printf("Изменено значение отчества!\n");

}

//изменить значение фамилии текущей вершины - список

void Tree::ChangeLastName(std::string\* lastName)

{

m\_Current->m\_LastName = \*lastName;

printf("Изменено значение фамилии!\n");

}

//изменить значение фамилии текущей вершины - массив

void Tree::ChangeLastName(std::string\* lastName)

{

m\_Values[m\_Current]->m\_LastName = \*lastName;

printf("Изменено значение фамилии!\n");

}

//изменить значение даты рождения текущей вершины - список

void Tree::ChangeDate(std::string\* date)

{

m\_Current->m\_Date = \*date;

printf("Изменено значение даты рождения!\n");

}

//изменить значение даты рождения текущей вершины - массив

void Tree::ChangeDate(std::string\* date)

{

m\_Values[m\_Current]->m\_Date = \*date;

printf("Изменено значение дати рождения!\n");

}

//изменить значение даты смерти текущей вершины - список

void Tree::ChangeDeath(std::string\* death)

{

m\_Current->m\_Death = \*death;

printf("Изменено значение даты смерти!\n");

}

//изменить значение даты смерти текущей вершины - массив

void Tree::ChangeDeath(std::string\* death)

{

m\_Values[m\_Current]->m\_Death = \*death;

printf("Изменено значение дати смерти!\n");

}

//изменить значение места рождения текущей вершины - список

void Tree::ChangePlace(std::string\* place)

{

m\_Current->m\_Place = \*place;

printf("Изменено значение места рождения!\n");

}

//изменить значение места рождения текущей вершины - массив

void Tree::ChangePlace(std::string\* place)

{

m\_Values[m\_Current]->m\_Place = \*place;

printf("Изменено значение места рождения!\n");

}

//изменить значение текущей вершины полностью - список

void Tree::ChangeAll(const Values& human) //передаем новые значения по ссылке

{

//каждый из элементов текущего значения меняем на соответствующее новое значение

m\_Current->m\_Name = human.m\_Name;

m\_Current->m\_LastName = human.m\_LastName;

m\_Current->m\_MiddleName = human.m\_MiddleName;

m\_Current->m\_Date = human.m\_Date;

if (!human.m\_Death.empty())

m\_Current->m\_Death = human.m\_Death;

if (!human.m\_Place.empty())

m\_Current->m\_Place = human.m\_Place;

printf("Изменени все значения!\n");//выводим сообщение о том что изменили значения

}

//изменить значение текущей вершины полностью - список

void Tree::ChangeAll(const Values& human) //аналогично, только способ обращения к текущим

{ //элементам по индексу в массиве а не по ссылке

m\_Values[m\_Current]->m\_Name = human.m\_Name;

m\_Values[m\_Current]->m\_LastName = human.m\_LastName;

m\_Values[m\_Current]->m\_MiddleName = human.m\_MiddleName;

m\_Values[m\_Current]->m\_Date = human.m\_Date;

if (!human.m\_Death.empty())

m\_Values[m\_Current]->m\_Death = human.m\_Death;

if (!human.m\_Place.empty())

m\_Values[m\_Current]->m\_Place = human.m\_Place;

printf("Изменени все значения!\n");

}

//создать левое поддерево для текущей вершины - список

void Tree::AddLeft(const Values& human) //передаем новый элемент по ссылке

{

if (m\_Current->m\_Left == nullptr) //проверяем что левый элемент не существует

{ //добавляем на место nullptr новый элемент с указателем на текущий и два

//nullptr для левого и правого указателя

m\_Current->m\_Left = new Human(human, nullptr, nullptr, m\_Current);

m\_Current = m\_Current->m\_Left; //делаем левый текущим

printf("Добавленно левое поддерево\n");//выводим сообщение о том что

return; //добавлено левое поддерево

}//выводим сообщение что левое поддерево уже существует

printf("Левое поддерево для данной вершины уже существует!");

}

//создать левое поддерево для текущей вершины - массив

void Tree::AddLeft(const Values& human) //аналогично

{

if (CheckLeft() == nullptr)

{

m\_Values[m\_Current \* 2 + 1] = new Values(human);

Left();

printf("Добавленно левое поддерево\n");

return;

}

printf("Левое поддерево для данной вершины уже существует!");

}

//создать правое поддерево для текущей вершины - список

void Tree::AddRight(const Values& human) //аналогично левом

{

if (m\_Current->m\_Right == nullptr)

{

m\_Current->m\_Right = new Human(human, nullptr, nullptr, m\_Current);

m\_Current = m\_Current->m\_Right;

printf("Добавленно правое поддерево\n");

return;

}

printf("Правое поддерево для данной вершины уже существует!");

}

//создать правое поддерево для текущей вершины - массив

void Tree::AddRight(const Values& human) //Аналогично левом

{

if (CheckRight() == nullptr)

{

m\_Values[m\_Current \* 2 + 2] = new Values(human);

Right();

printf("Добавленно правое поддерево\n");

return;

}

printf("Правое поддерево для данной вершины уже существует!");

}

**>>Конец описания базовых операций**

**Часть 2. Создать консольное приложение** **позволяющее выбрать и выполнить следующие действия над деревом:**

- удалить дерево (освободить память);

-создать упорядоченное по ключу бинарное дерево из текстового файла;

- создать неупорядоченное бинарное дерево из текстового файла (Корень – Левое\_поддерево – Правое\_поддерево, \*\* – признак пустого поддерева);

- добавить один элемент, со значениями введенными в интерактивном режиме в упорядоченное дерево по ключу для построения упорядоченного дерева;

- найти решение задачи (вариант 10):

Для каждого человека в генеалогическом дереве указаны следующие сведения: Фамилия, Имя, Отчество, дата рождения, а также при необходимости дата смерти и место рождения. **Для каждой вершины левое поддерево отвечает за отца, правое – за мать.** Например, дерево Ивана (указаны для упрощения только имена):

а) Создав соответствующее упорядоченное дерево, вывести информацию обо всех людях от самого старшего до самого младшего по дате рождения.

б) В генеалогическом дереве найти всех предков, кто родился в Москве. прожил не менее 70 лет.

- вывести дерево 2 разными способами: (Корень – Левое\_поддерево – Правое\_поддерево, Левое\_поддерево – Правое\_поддерево – Корень)

После каждой операции выводить содержимого дерева в текстовый лог-файл для контроля за состоянием бинарного дерева после каждой операции с начала и до конца работы программы первым из вышеуказанных способов.

Формат входного файла для задачи “a”

Имя

Отчество

Фамилия

дата рождения (дд.мм.гггг)

дата смерти (дд.мм.гггг)

место рождения

Формат входного файла для задачи “б”

Имя

Отчество

Фамилия

дата рождения (дд.мм.гггг)

дата смерти (дд.мм.гггг)

место рождения

\*\*

Имя

Отчество

Фамилия

дата рождения (дд.мм.гггг)

дата смерти (дд.мм.гггг)

место рождения

\*\*

\*\*

\*\* - обозначает что нет соответствующего поддерева, в нашем случае левого. Следовательно следующая запись является правим поддеревом с пустим левым и правим поддеревем.

Интерфейс созданного приложения

При запуске программы получаем следующий интерфейс

=====================================================

Здравствуйте! Пожалуйста выберите задание!

«Бинарное дерево»:

1) на основе ссылочного типа «Нелинейный связный список»

2) на основе динамического массива

E) чтобы закончит программу

=====================================================

После выбора опции 1 или 2 получаем новый интерфейс

-проверка на пустоту дерева : 1

-создание пустого дерева : 2

-сделать текущим корень дерева : 3

-сделать текущим левое поддерево(от текущей вершины) : 4

-сделать текущим правое поддерево(от текущей вершины) : 5

-сделать текущим родительскую вершину для текущей вершины : 6

-узнать значение текущей вершины : 7

-изменить значение текущей вершины : 8

-создать левое поддерево для текущей вершины : 9

-создать правое поддерево для текущей вершины : A

-удалить дерево : D

-создать упорядоченное по ключу : U

-добавить один элемент, со значениями введенными в интерактивном режиме (с клавиатуры) в упорядоченное дерево по ключу : N

-вывод от самого старшего до самого младшего : O

-вывести дерево левое-корень-правое : L

-вывести дерево корень-левое-правое : P

-создать неупорядоченное бинарное дерево : B

-найти всех предков, кто родился в Москве. прожил не менее 70 лет : F

-очистить консоль : C

-закончить : E

В случае выбора 8ой опции получаем новый интерфейс позволяющий уточнение

Если хотите изменить все значения нажмите : 0

Для изменения имени : 1

Для изменения фамилии : 2

Для изменения отчества : 3

Для изменения даты рождения : 4

Для изменения даты смерти : 5

Для изменения места рождения : 6

Сброс : E

**Функциональные тесты ввод и корректировка**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ теста** | **Входные данные** | **Ожидаемый результат**  **(после фильтрации)** | **Смысл теста** |
| 1 |  | Ошибка! Не смогли открыт данный файл! | Неверный путь к файлу |
| 2 | Неверный путь к файлу | Ошибка! Не смогли открыт данный файл! | Неверный путь к файлу |
| 3 | Пётр  Петров  Сергеевич  25 01 200  -  - | Ошибка в файле! Дата! | Ошибка записи в дате рождения |
| 4 | Пётр  Петров  Сергеевич  25 10 1900  25-02-1990  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: 2-2-1990  Место рождения: - | В файле единственная запись с указаной датой смерти и она удовлетворяет условию. |
| 5 | Пётр  Петров  Сергеевич  25 01 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: - | В файле единственная запись без указаной даты смерти и она удовлетворяет условию. |
| 6 | Пётр  Петров  Сергеевич  25 01 2005  -  -  Пётрааа  Петров  Сергеевич  25 01 2030  -  -  Пётр  Петров  Сергеевич  50 01 2005  -  -  Пётр  Петров  Сергеевич  05 13 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: -  Ошибка в файле! Дата!  Ошибка в файле! Дата! | Несколько записей не удовлетворяют условию(не верные даты), одна удовлетворяет. |
| 7 | Пётр  Петров  Сергеевич  25 01 2005  -  -  Пётрааа  Петров  Сергеевич  25 1 2002  -  -  Пётр  Петров  Сергеевич  20 1 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: -  Ошибка в файле! Дата!  Ошибка в файле! Дата! | Несколько записей не удовлетворяют условию(не верный формат даты), одна удовлетворяет |
| 8 | Пётр  Петров  Сергеевич  31 12 2020  -  Москва  Пётр  Иванов  Сергеевич  31.01.1900  25-08-1990  -  Иван  Петров  Сергеевич  01-12-2000  -  -  Иван  Петров  Сергеевич  01-11-2020  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 31 12 2020  Дата смерти: -  Место рождения: Москва  Имя: Пётр  Отчество: Иванов  Фамилия: Сергеевич  Дата рождения: 31.01.1900  Дата смерти: 25-08-1990  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 01-12-2000  Дата смерти: -  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 01-11-2020  Дата смерти: -  Место рождения: - | Поля для даты рождения указаны на максимальных граничных значений |
| 9 | Пётр  Петров  Сергеевич  32 13 2021  -  Москва  Пётр  Иванов  Сергеевич  31.13.1900  25-08-1990  -  Иван  Петров  Сергеевич  32-12-2000  -  -  Иван  Петров  Сергеевич  01-11-2025  -  - | Ошибка в файле! Дата!  Ошибка в файле! Дата!  Ошибка в файле! Дата!  Ошибка в файле! Дата! | Поля для даты рождения указаны больше максимальных граничных значений |
| 10 | Пётр  Петров  Сергеевич  02 01 1900  31-12-2020  Москва  Пётр  Иванов  Сергеевич  02 01 1900  31-11-2019  -  Иван  Петров  Сергеевич  02 01 1900  30-12-2019  -  Иван  Петров  Сергеевич  02 01 1900  30-11-2020  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 02 01 1900  Дата смерти: 31-12-2020  Место рождения: Москва  Имя: Пётр  Отчество: Иванов  Фамилия: Сергеевич  Дата рождения: 02 01 1900  Дата смерти: 31-11-2019  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 02 01 1900  Дата смерти: 30-12-2019  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 02 01 1900  Дата смерти: 30-11-2020  Место рождения: - | Поля для даты смерти указаны на максимальных граничных значений |
| 11 | Пётр  Петров  Сергеевич  02 01 1900  32-13-2021  Москва  Пётр  Иванов  Сергеевич  02 01 1900  32-11-2019  -  Иван  Петров  Сергеевич  02 01 1900  30-13-2019  -  Иван  Петров  Сергеевич  02 01 1900  30-11-2021  - | Ошибка в файле! Дата!  Ошибка в файле! Дата!  Ошибка в файле! Дата!  Ошибка в файле! Дата! | Поля для даты смерти указаны больше максимальных граничных значений |
| 12 | Пётр  Петров  Сергеевич  12 12 1950  31-12-2020  Москва  Пётр  Иванов  Сергеевич  02 01 2020  31-12-2020  -  Иван  Петров  Сергеевич  02 12 1900  30-12-2019  -  Иван  Петров  Сергеевич  31 01 1900  31-11-2020  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 12 12 1950  Дата смерти: 31-12-2020  Место рождения: Москва  Имя: Пётр  Отчество: Иванов  Фамилия: Сергеевич  Дата рождения: 02 01 2020  Дата смерти: 31-12-2020  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 02 12 1900  Дата смерти: 30-12-2019  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 31 01 1900  Дата смерти: 30-11-2020  Место рождения: - | Поля для даты рождения и смерти указаны на максимальных граничных значений |
| 13 | Пётр  Петров  Сергеевич  25 01 2005  -  Москва  Пётр  Иванов  Сергеевич  25 01 1900  25-08-1990  -  Иван  Петров  Сергеевич  25 01 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: Москва  Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 1900  Дата смерти: 25-08-1990  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: - | В файле несколько записей, все удовлетворяют условию |

**Функциональные тесты ввод и корректировка интерактивный режим**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ теста** | **Входные данные** | **Ожидаемый результат**  **(после фильтрации)** | **Смысл теста** |
| 1 | Пётр  Петров  Сергеевич  25 01 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: - | Все данные верны |
| 2 | Пётр  Петров  Сергеевич  25 01 20 | Пожалуйста введите верную дуту! | Дата рождения не удовлетворяет условыю (неверный ввод) |
| 3 | Пётр  Петров  Сергеевич  25 01 2021 | Пожалуйста введите верную дуту! | Дата рождения не удовлетворяет условыю (больше граничной даты) |
| 4 | Пётр  Петров  Сергеевич  25 01 2000  asd | Пожалуйста введите верную дуту! | Дата смерти не удовлетворяет условыю  (неверный ввод) |
| 5 | Пётр  Петров  Сергеевич  25 01 1960  25 05 2020  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 1960  Дата смерти: 25 05 2020  Место рождения: - | Все данные верны |

**Функциональные тесты для генеологического дерева**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ теста** | **Входные данные** | **Ожидаемый результат**  **(после фильтрации)** | **Смысл теста** |
| 1 |  | Ошибка! Не смогли открыт данный файл! | Неверный путь к файлу |
| 2 | Неверный путь к файлу | Ошибка! Не смогли открыт данный файл! | Неверный путь к файлу |
| 3 | Пётр  Петров  Сергеевич  25 01 2005  -  -  \*\*  \*\* | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: -  \*\*  \*\* | В файле единственная запись |
| 4 | Пётр  Петров  Сергеевич  25 01 2005  -  Москва  Пётр  Иванов  Сергеевич  25 01 1900  25-08-1990  -  Иван  Петров  Сергеевич  25 01 2005  -  -  \*\*  \*\*  \*\*  \*\* | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: Москва  Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 1900  Дата смерти: 25-08-1990  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: -  \*\*  \*\*  \*\*  \*\* | В файле несколько записей, все удовлетворяют условию |

**Функциональные тесты для упорядоченого дерева**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ теста** | **Входные данные** | **Ожидаемый результат**  **(после фильтрации)** | **Смысл теста** |
| 1 | Пётр  Петров  Сергеевич  25 01 200  -  - | Сначала создайте корень дерева! №2! | Ошибка записи в дате рождения |
| 2 | Пётр  Петров  Сергеевич  25 10 1900  25-02-1990  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: 2-2-1990  Место рождения: - | В файле единственная запись с указаной датой смерти и она удовлетворяет условию. |
| 3 | Пётр  Петров  Сергеевич  25 01 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: - | В файле единственная запись без указаной даты смерти и она удовлетворяет условию. |
| 4 | Пётр  Петров  Сергеевич  25 01 2005  -  -  Пётрааа  Петров  Сергеевич  25 01 2030  -  -  Пётр  Петров  Сергеевич  50 01 2005  -  -  Пётр  Петров  Сергеевич  05 13 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: - | Несколько записей не удовлетворяют условию(не верные даты), одна удовлетворяет. |
| 5 | Пётр  Петров  Сергеевич  25 01 2005  -  -  Пётрааа  Петров  Сергеевич  25 1 2002  -  -  Пётр  Петров  Сергеевич  20 1 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: - | Несколько записей не удовлетворяют условию(не верный формат даты), одна удовлетворяет |
| 6 | Пётр  Петров  Сергеевич  31 12 2020  -  Москва  Пётр  Иванов  Сергеевич  31.01.1900  25-08-1990  -  Иван  Петров  Сергеевич  01-12-2000  -  -  Иван  Петров  Сергеевич  01-11-2020  -  - | Имя: Пётр  Отчество: Иванов  Фамилия: Сергеевич  Дата рождения: 31.01.1900  Дата смерти: 25-08-1990  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 01-12-2000  Дата смерти: -  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 01-11-2020  Дата смерти: -  Место рождения: -  Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 31 12 2020  Дата смерти: -  Место рождения: Москва | Поля для даты рождения указаны на максимальных граничных значений |
| 7 | Пётр  Петров  Сергеевич  32 13 2021  -  Москва  Пётр  Иванов  Сергеевич  31.13.1900  25-08-1990  -  Иван  Петров  Сергеевич  32-12-2000  -  -  Иван  Петров  Сергеевич  01-11-2025  -  - |  | Поля для даты рождения указаны больше максимальных граничных значений |
| 8 | Пётр  Петров  Сергеевич  02 01 1900  31-12-2020  Москва  Пётр  Иванов  Сергеевич  02 01 1900  31-11-2019  -  Иван  Петров  Сергеевич  02 01 1900  30-12-2019  -  Иван  Петров  Сергеевич  02 01 1900  30-11-2020  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 02 01 1900  Дата смерти: 31-12-2020  Место рождения: Москва  Имя: Пётр  Отчество: Иванов  Фамилия: Сергеевич  Дата рождения: 02 01 1900  Дата смерти: 31-11-2019  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 02 01 1900  Дата смерти: 30-12-2019  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 02 01 1900  Дата смерти: 30-11-2020  Место рождения: - | Поля для даты смерти указаны на максимальных граничных значений. Все даты рождения одиноковые |
| 9 | Пётр  Петров  Сергеевич  02 01 1900  32-13-2021  Москва  Пётр  Иванов  Сергеевич  02 01 1900  32-11-2019  -  Иван  Петров  Сергеевич  02 01 1900  30-13-2019  -  Иван  Петров  Сергеевич  02 01 1900  30-11-2021  - |  | Поля для даты смерти указаны больше максимальных граничных значений |
| 10 | Пётр  Петров  Сергеевич  12 12 1950  31-12-2020  Москва  Пётр  Иванов  Сергеевич  02 01 2020  31-12-2020  -  Иван  Петров  Сергеевич  02 12 1900  30-12-2019  -  Иван  Петров  Сергеевич  31 01 1900  31-11-2020  - | Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 31 01 1900  Дата смерти: 31-11-2020  Место рождения: -  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 02 12 1900  Дата смерти: 30-12-2019  Место рождения: -  Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 12 12 1950  Дата смерти: 31-12-2020  Место рождения: Москва  Имя: Пётр  Отчество: Иванов  Фамилия: Сергеевич  Дата рождения: 02 01 2020  Дата смерти: 31-12-2020  Место рождения: - | Поля для даты рождения и смерти указаны на максимальных граничных значений |
| 11 | Пётр  Петров  Сергеевич  25 01 2005  -  Москва  Пётр  Иванов  Сергеевич  25 01 1900  25-08-1990  -  Иван  Петров  Сергеевич  25 01 2005  -  - | Имя: Пётр  Отчество: Иванов  Фамилия: Сергеевич  Дата рождения: 25 01 1900  Дата смерти: 25-08-1990  Место рождения: -  Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: Москва  Имя: Иван  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 2005  Дата смерти: -  Место рождения: - | В файле несколько записей, все удовлетворяют условию |

**Функциональные тесты для поиска по месту рождения (Москва) и не мене 70 лет прожил (включая живие)**

|  |  |  |  |
| --- | --- | --- | --- |
| **№ теста** | **Входные данные** | **Ожидаемый результат**  **(после фильтрации)** | **Смысл теста** |
| 1 | Пётр  Петров  Сергеевич  25 01 200  -  - | Сначала создайте корень дерева! №2! | Ошибка записи в дате рождения |
| 2 | Пётр  Петров  Сергеевич  25 10 1900  25-02-1990  Moskva | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 10 1900  Дата смерти: 25-02-1990  Место рождения: Moskva | В файле единственная запись с указаной датой смерти и она удовлетворяет условию.  Транслит места рождения |
| 3 | Пётр  Петров  Сергеевич  25 01 1900  -  Moscow | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 1900  Дата смерти: -  Место рождения: Moscow | В файле единственная запись без указаной даты смерти и она удовлетворяет условию.  Москва на английском |
| 4 | Пётр  Петров  Сергеевич  25 01 1900  -  Москва  Пётрааа  Петров  Сергеевич  25 01 2030  -  Moskva  Пётр  Петров  Сергеевич  01 01 2005  -  Москва  Пётр  Петров  Сергеевич  05 13 2005  -  - | Имя: Пётр  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 1900  Дата смерти: -  Место рождения: Москва | Несколько записей не удовлетворяют условию(не верные даты).  Не указаные даты смерти. |
| 5 | Пётр  Петров  Сергеевич  25 01 2000  25 01 2020  -  Пётрааа  Петров  Сергеевич  25 01 1900  25 01 1970  Moskva  Пётр  Петров  Сергеевич  20 1 2005  -  - | Имя: Пётрааа  Отчество: Петров  Фамилия: Сергеевич  Дата рождения: 25 01 1900  Дата смерти: 25 01 1970  Место рождения: Moskva | Несколько записей не удовлетворяют условию(не верный формат даты). Указана дата смерти. Причем граничная |
| 6 | Пётр  Петров  Сергеевич  25 01 2000  25 01 2020  -  Пётрааа  Петров  Сергеевич  25 01 1900  25 01 1980  -  Пётр  Петров  Сергеевич  20 1 2005  -  - |  | Несколько записей не удовлетворяют условию(не верный формат даты). Указана граничная дата смерти, но город не указан |

**Программный код процедуры для нахождения ошибок в вводе**

bool CheckDate(const std::string\* date)

{

if (date->size() < 10 || date->size() > 11)

return false;

try

{

if (std::stoi(&date->at(0)) > 31 || std::stoi(&date->at(3)) > 12 || std::stoi(&date->at(6)) > CURRENT\_YEAR) //CURRENT\_YEAR = 2020

return false;

}

catch (std::invalid\_argument& e) { return false; }

return true;

}

**Программный код процедуры для поиска по месту рождения**

bool IsMoscow(const std::string\* place)

{

if (place->compare("Москва") == 0 || place->compare("Moskva") == 0 || place->compare("Moscow") == 0)

return true;

return false;

}

bool Calculator(const std::string\* date\_root, const std::string\* date\_new)

{

char yyyy\_root[4];

char yyyy\_new[4];

for (int i = 0; i < 4; i++)

{

yyyy\_root[i] = date\_root->at(i + 6);

if (date\_new->size() < 10)

yyyy\_new[i] = TODAY[i + 6];

else

yyyy\_new[i] = date\_new->at(i + 6);

}

if (std::stoi(yyyy\_new) - std::stoi(yyyy\_root) >= 70)

return true;

return false;

}

**Заключение**

В данной работе была реализована структура «бинарное дерево» двумя разными способами: с помощью одномерного динамического массива и с помощью ссылочного типа «Нелинейный связный список».

Реализация с помощью ссылочного типа позволяет создать дерево, превышающий размер доступной оперативной памяти, однако из-за последовательного доступа работа с большими списками затруднительна. Больше плюсов у реализации с помощью ссылочного типа не было найдено. Поскольку связанный список использует указатели для доступа к элементам, время выполнения многократно медленнее, чем в реализации с массивом. Оба способа сравнимы по сложности реализации, но вариант использованием ссылочного типа оказывается удобнее тем, что, например, не нужно оставлять запас для добавления элементов, как это делалось в реализации с помощью одномерного динамического массива.

**Приложения**

Приложение 1

**tree.h**

#pragma once

#include "../additional/values.h"

namespace linked\_tree {

struct Human {

//информация о человеке

std::string m\_MiddleName;

std::string m\_Name;

std::string m\_LastName;

std::string m\_Date;

std::string m\_Death;

std::string m\_Place;

bool m\_Printed;

//связы

Human\* m\_Left;

Human\* m\_Right;

Human\* m\_Parent;

//конструктор

Human(const Values& values, Human\* left, Human\* right, Human\* parent)

{

m\_MiddleName = values.m\_MiddleName;

m\_Name = values.m\_Name;

m\_LastName = values.m\_LastName;

m\_Date = values.m\_Date;

if (!values.m\_Death.empty())

m\_Death = values.m\_Death;

if (!values.m\_Place.empty())

m\_Place = values.m\_Place;

m\_Left = left;

m\_Right = right;

m\_Parent = parent;

m\_Printed = values.m\_Printed;

}

};

struct Tree {

Human\* m\_First;

Human\* m\_Current;

//проверить левый

Human\* CheckLeft();

//проверить правый

Human\* CheckRight();

//Проверит текущий

Human\* CheckCurrent();

//проверит на существование первого элемента

bool CheckFirst();

//создание пустого дерева

void CreateFirst(const Values& value);

//сделать текущим корень дерева;

void Root();

//сделать текущим левое поддерево

void Left();

//сделать текущим правое поддерево

void Right();

//сделать текущим родительскую вершину для текущей вершины

void Parent();

//узнать значение текущей вершины;

void GetValue(const char& c);

void ChangeName(std::string\* name);

void ChangeSurname(std::string\* middleName);

void ChangeLastName(std::string\* lastName);

void ChangeDate(std::string\* date);

void ChangeDeath(std::string\* death);

void ChangePlace(std::string\* place);

//изменить значение текущей вершины;

void ChangeAll(const Values& human);

//создать левое поддерево для текущей вершины;

void AddLeft(const Values& human);

//создать правое поддерево для текущей вершины;

void AddRight(const Values& human);

//проверка на пустоту дерева(поддерева, начинающегося с текущей вершины);

bool CheckIfEmpty();

//Удалить дерево

void Delete();

//Самый левый элемент

void LeftMost();

void CreateFromFile();

bool CompareDates(const std::string\* date\_root, const std::string\* date\_new);

void AddByKey(const Values& human);

void PrintByAge();

bool Printed();

void PrintRoot();

void ResetPrint();

void GenTree();

void SearchGen();

};

}

**tree.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "tree.h"

#include "../additional/filereader.h"

#include "../additional/calculator.h"

#include "../additional/date.h"

namespace linked\_tree {

bool Tree::CheckFirst()

{

return m\_First == nullptr;

}

void Tree::CreateFirst(const Values& human)

{

if (m\_First == nullptr) {

m\_First = new Human(human, nullptr, nullptr, nullptr);

m\_Current = m\_First;

return;

}

printf("Уже существует корень!\n");

}

Human\* Tree::CheckLeft()

{

return (m\_Current->m\_Left != nullptr) ? m\_Current->m\_Left : nullptr;

}

Human\* Tree::CheckRight()

{

return (m\_Current->m\_Right != nullptr) ? m\_Current->m\_Right : nullptr;

}

Human\* Tree::CheckCurrent()

{

return (m\_Current != nullptr) ? m\_Current : nullptr;

}

void Tree::Root()

{

if (m\_First != nullptr) {

m\_Current = m\_First;

}

}

void Tree::Left()

{

if (m\_Current->m\_Left != nullptr) {

m\_Current = m\_Current->m\_Left;

return;

}

}

void Tree::Right()

{

if (m\_Current->m\_Right != nullptr) {

m\_Current = m\_Current->m\_Right;

return;

}

}

//сделать текущим родительскую вершину для текущей вершины(для удобства можно в способе 1 ввести третью связь – обратную связь с родительской вершиной, чтобы не искать ее по всему дереву – Трёхсвязный список);

void Tree::Parent()

{

if (m\_Current->m\_Parent != nullptr) {

m\_Current = m\_Current->m\_Parent;

return;

}

}

//узнать значение текущей вершины;

void Tree::GetValue(const char& c)

{

if (c == 'a')

m\_Current->m\_Printed = true;

printf("Имя: %s\n", m\_Current->m\_Name.c\_str());

printf("Отчество: %s\n", m\_Current->m\_MiddleName.c\_str());

printf("Фамилия: %s\n", m\_Current->m\_LastName.c\_str());

printf("Дата рождения: %s\n", m\_Current->m\_Date.c\_str());

if (!m\_Current->m\_Death.empty())

printf("Дата смерти: %s\n", m\_Current->m\_Death.c\_str());

if (!m\_Current->m\_Place.empty())

printf("Место рождения: %s\n", m\_Current->m\_Place.c\_str());

}

void Tree::ChangeName(std::string\* name)

{

m\_Current->m\_Name = \*name;

printf("Изменено значение имени!\n");

}

void Tree::ChangeSurname(std::string\* middleName)

{

m\_Current->m\_MiddleName = \*middleName;

printf("Изменено значение отчества!\n");

}

void Tree::ChangeLastName(std::string\* lastName)

{

m\_Current->m\_LastName = \*lastName;

printf("Изменено значение фамилии!\n");

}

void Tree::ChangeDate(std::string\* date)

{

m\_Current->m\_Date = \*date;

printf("Изменено значение даты рождения!\n");

}

void Tree::ChangeDeath(std::string\* death)

{

m\_Current->m\_Death = \*death;

printf("Изменено значение даты смерти!\n");

}

void Tree::ChangePlace(std::string\* place)

{

m\_Current->m\_Place = \*place;

printf("Изменено значение места рождения!\n");

}

//изменить значение текущей вершины;

void Tree::ChangeAll(const Values& human)

{

m\_Current->m\_Name = human.m\_Name;

m\_Current->m\_LastName = human.m\_LastName;

m\_Current->m\_MiddleName = human.m\_MiddleName;

m\_Current->m\_Date = human.m\_Date;

if (!human.m\_Death.empty())

m\_Current->m\_Death = human.m\_Death;

if (!human.m\_Place.empty())

m\_Current->m\_Place = human.m\_Place;

printf("Изменени все значения!\n");

}

//создать левое поддерево для текущей вершины;

void Tree::AddLeft(const Values& human)

{

if (m\_Current->m\_Left == nullptr)

{

m\_Current->m\_Left = new Human(human, nullptr, nullptr, m\_Current);

m\_Current = m\_Current->m\_Left;

printf("Добавленно левое поддерево\n");

return;

}

printf("Левое поддерево для данной вершины уже существует!");

}

//создать правое поддерево для текущей вершины;

void Tree::AddRight(const Values& human)

{

if (m\_Current->m\_Right == nullptr)

{

m\_Current->m\_Right = new Human(human, nullptr, nullptr, m\_Current);

m\_Current = m\_Current->m\_Right;

printf("Добавленно правое поддерево\n");

return;

}

printf("Правое поддерево для данной вершины уже существует!");

}

//проверка на пустоту дерева(поддерева, начинающегося с текущей вершины);

bool Tree::CheckIfEmpty()

{

return (m\_Current->m\_Left != nullptr || m\_Current->m\_Right != nullptr) ? false : true;

}

void Tree::LeftMost()

{

while (m\_Current->m\_Left != nullptr)

Left();

}

void Tree::Delete()

{

Root();

LeftMost();

while (true) {

while (true) {

if (CheckRight() != nullptr)

{

Right();

LeftMost();

}

else

break;

}

Parent();

if (CheckLeft() != nullptr)

{

delete m\_Current->m\_Left;

m\_Current->m\_Left = nullptr;

}

else if (CheckRight() != nullptr)

{

delete m\_Current->m\_Right;

m\_Current->m\_Right = nullptr;

}

else if (m\_Current == m\_First)

{

delete m\_Current;

m\_First = nullptr;

printf("Дерево удаленно! \n");

return;

}

}

}

void Tree::CreateFromFile()

{

FILE\* pFile = fopen("src\\information\\example1.txt", "r");

if (pFile == nullptr) {

printf("Ошибка! Не смогли открыт данный файл!");

return;

}

File\* f = new File(pFile);

while (f->m\_Position < f->m\_Size) {

Values values;

if (!f->ToObject(&values)) {

printf("Ошибка в файле! \n");

continue;

}

if (!CheckDate(&values.m\_Date) || (!CheckDate(&values.m\_Death) && values.m\_Death.size() > 1))

continue;

if (this->m\_First == nullptr)

CreateFirst(values);

else

{

m\_Current = m\_First;

while (true) {

if (CompareDates(&m\_Current->m\_Date, &values.m\_Date) && CheckRight() != nullptr)

Right();

else if (CompareDates(&m\_Current->m\_Date, &values.m\_Date) && CheckRight() == nullptr)

{

AddRight(values);

break;

}

else if (!CompareDates(&m\_Current->m\_Date, &values.m\_Date) && CheckLeft() != nullptr)

Left();

else

{

AddLeft(values);

break;

}

}

}

}

fclose(pFile);

}

bool Tree::CompareDates(const std::string\* date\_root, const std::string\* date\_new)

{

char dd\_root[2];

char mm\_root[2];

char yyyy\_root[4];

char dd\_new[2];

char mm\_new[2];

char yyyy\_new[4];

for (int i = 0; i < 2; i++)

{

dd\_root[i] = date\_root->at(i);

dd\_new[i] = date\_new->at(i);

}

for (int i = 0; i < 2; i++)

{

mm\_root[i] = date\_root->at(i + 3);

mm\_new[i] = date\_new->at(i + 3);

}

for (int i = 0; i < 4; i++)

{

yyyy\_root[i] = date\_root->at(i + 6);

yyyy\_new[i] = date\_new->at(i + 6);

}

try {

if (std::stoi(yyyy\_new) < std::stoi(yyyy\_root))

return false;

else if (std::stoi(yyyy\_new) > std::stoi(yyyy\_root))

return true;

else if (std::stoi(yyyy\_new) == std::stoi(yyyy\_root))

{

if (std::stoi(mm\_new) < std::stoi(mm\_root))

return false;

else if (std::stoi(mm\_new) > std::stoi(mm\_root))

return true;

else if (std::stoi(mm\_new) == std::stoi(mm\_root))

{

if (std::stoi(dd\_new) < std::stoi(dd\_root))

return false;

else if (std::stoi(dd\_new) > std::stoi(dd\_root))

return true;

else

return true;

}

}

}

catch (std::invalid\_argument& e)

{

printf("Ошибка! Дата рождения запись в файле неверная! Пожалуйста исправите ошибку, и перезапустите программу!\n");

return false;

}

return false;

}

bool Tree::Printed()

{

return m\_Current->m\_Printed;

}

void Tree::AddByKey(const Values& human)

{

m\_Current = m\_First;

while (true) {

if (CompareDates(&m\_Current->m\_Date, &human.m\_Date) && CheckRight() != nullptr)

Right();

else if (CompareDates(&m\_Current->m\_Date, &human.m\_Date) && CheckRight() == nullptr)

{

AddRight(human);

break;

}

else if (!CompareDates(&m\_Current->m\_Date, &human.m\_Date) && CheckLeft() != nullptr)

Left();

else

{

AddLeft(human);

break;

}

}

}

void Tree::PrintByAge()

{

Root();

LeftMost();

GetValue('a');

while (true)

{

if (m\_Current->m\_Right != nullptr && !CheckRight()->m\_Printed)

{

Right();

LeftMost();

GetValue('a');

}

else {

Parent();

if (Printed())

{

if (m\_Current == m\_First && (CheckRight() == nullptr || CheckRight()->m\_Printed))

break;

continue;

}

GetValue('a');

}

}

ResetPrint();

}

void Tree::ResetPrint()

{

Root();

LeftMost();

m\_Current->m\_Printed = false;

while (true)

{

if (m\_Current->m\_Right != nullptr && CheckRight()->m\_Printed)

{

Right();

LeftMost();

m\_Current->m\_Printed = false;

}

else {

Parent();

if (!Printed())

{

if (m\_Current == m\_First && (CheckRight() == nullptr || !CheckRight()->m\_Printed))

return;

continue;

}

m\_Current->m\_Printed = false;

}

}

}

void Tree::PrintRoot()

{

Root();

GetValue('a');

while (true) {

while (CheckLeft() != nullptr && !CheckLeft()->m\_Printed)

{

Left();

GetValue('a');

}

if (CheckLeft() == nullptr && (CheckRight() == nullptr || !CheckRight()->m\_Printed))

printf("\*\*\n");

if (CheckRight() != nullptr && !CheckRight()->m\_Printed)

{

Right();

GetValue('a');

continue;

}

if (CheckRight() == nullptr)

printf("\*\*\n");

Parent();

if (m\_Current == m\_First && CheckRight() == nullptr && CheckLeft() == nullptr)

break;

else if (m\_Current == m\_First && CheckRight() == nullptr && CheckLeft() != nullptr)

{

if (CheckLeft()->m\_Printed)

break;

else

continue;

}

else if (m\_Current == m\_First && CheckRight() != nullptr && CheckLeft() != nullptr)

{

if (CheckRight()->m\_Printed)

break;

}

}

ResetPrint();

}

void Tree::GenTree()

{

FILE\* pFile = fopen("src\\information\\example2.txt", "r");

if (pFile == nullptr) {

printf("Ошибка! Не смогли открыт данный файл!");

return;

}

File\* f = new File(pFile);

bool check = true;

while (f->m\_Position < f->m\_Size) {

Values values;

if (m\_First == nullptr)

{

f->GenTree(&values);

CreateFirst(values);

}

else {

check = f->GenTree(&values);

if (check && CheckLeft() == nullptr)

{

AddLeft(values);

continue;

}

else

{

if (!check && CheckLeft() == nullptr)

check = f->GenTree(&values);

if (check)

{

AddRight(values);

continue;

}

else

{

Parent();

while ((CheckLeft() != nullptr && CheckRight() != nullptr) && m\_Current != m\_First)

Parent();

}

}

}

}

fclose(pFile);

}

void Tree::SearchGen()

{

Root();

if (Calculator(&m\_Current->m\_Date, &m\_Current->m\_Death) && IsMoscow(&m\_Current->m\_Place))

GetValue('a');

else

m\_Current->m\_Printed = true;

while (true) {

while (CheckLeft() != nullptr && !CheckLeft()->m\_Printed)

{

Left();

if (Calculator(&m\_Current->m\_Date, &m\_Current->m\_Death) && IsMoscow(&m\_Current->m\_Place))

GetValue('a');

else

m\_Current->m\_Printed = true;

}

if (CheckRight() != nullptr && !CheckRight()->m\_Printed)

{

Right();

if (Calculator(&m\_Current->m\_Date, &m\_Current->m\_Death) && IsMoscow(&m\_Current->m\_Place))

GetValue('a');

else

m\_Current->m\_Printed = true;

continue;

}

Parent();

if (m\_Current == m\_First && CheckRight() == nullptr && CheckLeft() == nullptr)

break;

else if (m\_Current == m\_First && CheckRight() == nullptr && CheckLeft() != nullptr)

{

if (CheckLeft()->m\_Printed)

break;

else

continue;

}

else if (m\_Current == m\_First && CheckRight() != nullptr && CheckLeft() != nullptr)

{

if (CheckRight()->m\_Printed)

break;

}

}

ResetPrint();

}

}

Приложение 2

**array.h**

#pragma once

#include "../additional/values.h"

namespace array\_tree {

#define MAX\_ARRAY\_SIZE 100

struct Tree {

Values\*\* m\_Values = (Values\*\*)malloc(MAX\_ARRAY\_SIZE \* sizeof(Values));

Tree()

{

memset(m\_Values, 0, MAX\_ARRAY\_SIZE \* sizeof(Values));

}

~Tree()

{

delete[] m\_Values;

}

unsigned int m\_Current = 0;

//проверит на существование первого элемента

bool CheckFirst();

//проверить левый

Values\* CheckLeft();

//проверить правый

Values\* CheckRight();

//Проверит текущий

Values\* CheckCurrent();

//создание пустого дерева

void CreateFirst(const Values& value);

//сделать текущим корень дерева;

void Root();

//сделать текущим левое поддерево

void Left();

//сделать текущим правое поддерево

void Right();

//сделать текущим родительскую вершину для текущей вершины

void Parent();

//узнать значение текущей вершины;

void GetValue(const char& c);

void ChangeName(std::string\* name);

void ChangeSurname(std::string\* middleName);

void ChangeLastName(std::string\* lastName);

void ChangeDate(std::string\* date);

void ChangeDeath(std::string\* death);

void ChangePlace(std::string\* place);

//изменить значение текущей вершины;

void ChangeAll(const Values& human);

//создать левое поддерево для текущей вершины;

void AddLeft(const Values& human);

//создать правое поддерево для текущей вершины;

void AddRight(const Values& human);

//проверка на пустоту дерева(поддерева, начинающегося с текущей вершины);

bool CheckIfEmpty();

//Удалить дерево

void Delete();

//Самый левый элемент

void LeftMost();

void CreateFromFile();

bool CompareDates(const std::string\* date\_root, const std::string\* date\_new);

void AddByKey(const Values& human);

void PrintByAge();

bool Printed();

void PrintRoot();

void ResetPrint();

void GenTree();

void SearchGen();

};

}

**array.cpp**

#include "array.h"

#include "../additional/filereader.h"

#include "../additional/calculator.h"

#include "../additional/date.h"

namespace array\_tree {

bool Tree::CheckFirst()

{

return m\_Values[0] == nullptr;

}

void Tree::CreateFirst(const Values& human)

{

if (m\_Values[0] == nullptr) {

m\_Values[0] = new Values(human);

m\_Current = 0;

return;

}

printf("Уже существует корень!\n");

}

void Tree::Root()

{

if (m\_Values[0] != nullptr) {

m\_Current = 0;

}

}

Values\* Tree::CheckLeft()

{

return (m\_Values[m\_Current \* 2 + 1] != nullptr) ? m\_Values[m\_Current \* 2 + 1] : nullptr;

}

Values\* Tree::CheckRight()

{

return (m\_Values[m\_Current \* 2 + 2] != nullptr) ? m\_Values[m\_Current \* 2 + 2] : nullptr;

}

Values\* Tree::CheckCurrent()

{

return (m\_Values[m\_Current] != nullptr) ? m\_Values[m\_Current] : nullptr;

}

void Tree::Left()

{

m\_Current = CheckLeft() != nullptr ? (m\_Current \* 2) + 1 : m\_Current;

}

void Tree::Right()

{

m\_Current = CheckRight() != nullptr ? (m\_Current \* 2) + 2 : m\_Current;

}

//сделать текущим родительскую вершину для текущей вершины(для удобства можно в способе 1 ввести третью связь – обратную связь с родительской вершиной, чтобы не искать ее по всему дереву – Трёхсвязный список);

void Tree::Parent()

{

if (m\_Current != 0)

m\_Current = m\_Current % 2 == 0 ? (m\_Current - 2) / 2 : (m\_Current - 1) / 2;

else

m\_Current = 0;

}

//узнать значение текущей вершины;

void Tree::GetValue(const char& c)

{

if (c == 'a')

CheckCurrent()->m\_Printed = true;

printf("Имя: %s\n", m\_Values[m\_Current]->m\_Name.c\_str());

printf("Отчество: %s\n", m\_Values[m\_Current]->m\_MiddleName.c\_str());

printf("Фамилия: %s\n", m\_Values[m\_Current]->m\_LastName.c\_str());

printf("Дата рождения: %s\n", m\_Values[m\_Current]->m\_Date.c\_str());

if (!m\_Values[m\_Current]->m\_Death.empty())

printf("Дата смерти: %s\n", m\_Values[m\_Current]->m\_Death.c\_str());

if (!m\_Values[m\_Current]->m\_Place.empty())

printf("Место рождения: %s\n", m\_Values[m\_Current]->m\_Place.c\_str());

}

void Tree::ChangeName(std::string\* name)

{

m\_Values[m\_Current]->m\_Name = \*name;

printf("Изменено значение имени!\n");

}

void Tree::ChangeSurname(std::string\* middleName)

{

m\_Values[m\_Current]->m\_MiddleName = \*middleName;

printf("Изменено значение отчества!\n");

}

void Tree::ChangeLastName(std::string\* lastName)

{

m\_Values[m\_Current]->m\_LastName = \*lastName;

printf("Изменено значение фамилии!\n");

}

void Tree::ChangeDate(std::string\* date)

{

m\_Values[m\_Current]->m\_Date = \*date;

printf("Изменено значение дати рождения!\n");

}

void Tree::ChangeDeath(std::string\* death)

{

m\_Values[m\_Current]->m\_Death = \*death;

printf("Изменено значение дати смерти!\n");

}

void Tree::ChangePlace(std::string\* place)

{

m\_Values[m\_Current]->m\_Place = \*place;

printf("Изменено значение места рождения!\n");

}

//изменить значение текущей вершины;

void Tree::ChangeAll(const Values& human)

{

m\_Values[m\_Current]->m\_Name = human.m\_Name;

m\_Values[m\_Current]->m\_LastName = human.m\_LastName;

m\_Values[m\_Current]->m\_MiddleName = human.m\_MiddleName;

m\_Values[m\_Current]->m\_Date = human.m\_Date;

if (!human.m\_Death.empty())

m\_Values[m\_Current]->m\_Death = human.m\_Death;

if (!human.m\_Place.empty())

m\_Values[m\_Current]->m\_Place = human.m\_Place;

printf("Изменени все значения!\n");

}

//создать левое поддерево для текущей вершины;

void Tree::AddLeft(const Values& human)

{

if (CheckLeft() == nullptr)

{

m\_Values[m\_Current \* 2 + 1] = new Values(human);

Left();

printf("Добавленно левое поддерево\n");

return;

}

printf("Левое поддерево для данной вершины уже существует!");

}

//создать правое поддерево для текущей вершины;

void Tree::AddRight(const Values& human)

{

if (CheckRight() == nullptr)

{

m\_Values[m\_Current \* 2 + 2] = new Values(human);

Right();

printf("Добавленно правое поддерево\n");

return;

}

printf("Правое поддерево для данной вершины уже существует!");

}

//проверка на пустоту дерева(поддерева, начинающегося с текущей вершины);

bool Tree::CheckIfEmpty()

{

return (CheckLeft() != nullptr || CheckRight() != nullptr) ? false : true;

}

void Tree::LeftMost()

{

while (m\_Values[m\_Current \* 2 + 1] != nullptr)

Left();

}

void Tree::Delete()

{

Root();

LeftMost();

while (true) {

while (true) {

if (CheckRight() != nullptr)

{

Right();

LeftMost();

}

else

break;

}

Parent();

if (CheckLeft() != nullptr)

{

delete CheckLeft();

m\_Values[m\_Current \* 2 + 1] = nullptr;

}

else if (CheckRight() != nullptr)

{

delete CheckRight();

m\_Values[m\_Current \* 2 + 2] = nullptr;

}

else if (m\_Current == 0)

{

delete m\_Values[0];

m\_Values[0] = nullptr;

printf("Дерево удаленно! \n");

return;

}

}

}

void Tree::CreateFromFile()

{

FILE\* pFile = fopen("src\\information\\example1.txt", "r");

if (pFile == nullptr) {

printf("Ошибка! Не смогли открыт данный файл! \n");

return;

}

File\* f = new File(pFile);

while (f->m\_Position < f->m\_Size) {

Values values;

if (!f->ToObject(&values)) {

printf("Ошибка в файле! \n");

continue;

}

if (!CheckDate(&values.m\_Date) || (!CheckDate(&values.m\_Death) && values.m\_Death.size() > 1))

continue;

if (m\_Values[0] == nullptr)

CreateFirst(values);

else

{

m\_Current = 0;

while (true) {

if (CompareDates(&m\_Values[m\_Current]->m\_Date, &values.m\_Date) && CheckRight() != nullptr)

Right();

else if (CompareDates(&m\_Values[m\_Current]->m\_Date, &values.m\_Date) && CheckRight() == nullptr)

{

AddRight(values);

break;

}

else if (!CompareDates(&m\_Values[m\_Current]->m\_Date, &values.m\_Date) && CheckLeft() != nullptr)

Left();

else

{

AddLeft(values);

break;

}

}

}

}

fclose(pFile);

}

bool Tree::CompareDates(const std::string\* date\_root, const std::string\* date\_new)

{

char dd\_root[2];

char mm\_root[2];

char yyyy\_root[4];

char dd\_new[2];

char mm\_new[2];

char yyyy\_new[4];

for (int i = 0; i < 2; i++)

{

dd\_root[i] = date\_root->at(i);

dd\_new[i] = date\_new->at(i);

}

for (int i = 0; i < 2; i++)

{

mm\_root[i] = date\_root->at(i + 3);

mm\_new[i] = date\_new->at(i + 3);

}

for (int i = 0; i < 4; i++)

{

yyyy\_root[i] = date\_root->at(i + 6);

yyyy\_new[i] = date\_new->at(i + 6);

}

if (std::stoi(yyyy\_new) < std::stoi(yyyy\_root))

return false;

else if (std::stoi(yyyy\_new) > std::stoi(yyyy\_root))

return true;

else if (std::stoi(yyyy\_new) == std::stoi(yyyy\_root))

{

if (std::stoi(mm\_new) < std::stoi(mm\_root))

return false;

else if (std::stoi(mm\_new) > std::stoi(mm\_root))

return true;

else if (std::stoi(mm\_new) == std::stoi(mm\_root))

{

if (std::stoi(dd\_new) < std::stoi(dd\_root))

return false;

else if (std::stoi(dd\_new) > std::stoi(dd\_root))

return true;

else

return true;

}

}

return false;

}

bool Tree::Printed()

{

return CheckCurrent()->m\_Printed;

}

void Tree::AddByKey(const Values& human)

{

m\_Current = 0;

while (true) {

if (CompareDates(&CheckCurrent()->m\_Date, &human.m\_Date) && CheckRight() != nullptr)

Right();

else if (CompareDates(&CheckCurrent()->m\_Date, &human.m\_Date) && CheckRight() == nullptr)

{

AddRight(human);

break;

}

else if (!CompareDates(&CheckCurrent()->m\_Date, &human.m\_Date) && CheckLeft() != nullptr)

Left();

else

{

AddLeft(human);

break;

}

}

}

void Tree::PrintByAge()

{

Root();

LeftMost();

GetValue('a');

while (true)

{

if (CheckRight() != nullptr && !CheckRight()->m\_Printed)

{

Right();

LeftMost();

GetValue('a');

}

else {

Parent();

if (Printed())

{

if (m\_Current == 0 && (CheckRight() == nullptr || CheckRight()->m\_Printed))

break;

continue;

}

GetValue('a');

}

}

ResetPrint();

}

void Tree::ResetPrint()

{

Root();

LeftMost();

CheckCurrent()->m\_Printed = false;

while (true)

{

if (CheckRight() != nullptr && CheckRight()->m\_Printed)

{

Right();

LeftMost();

CheckCurrent()->m\_Printed = false;

}

else {

Parent();

if (!Printed())

{

if (m\_Current == 0 && (CheckRight() == nullptr || !CheckRight()->m\_Printed))

return;

continue;

}

CheckCurrent()->m\_Printed = false;

}

}

}

void Tree::PrintRoot()

{

Root();

GetValue('a');

while (true) {

while (CheckLeft() != nullptr && !CheckLeft()->m\_Printed)

{

Left();

GetValue('a');

}

if (CheckLeft() == nullptr && (CheckRight() == nullptr || !CheckRight()->m\_Printed))

printf("\*\*\n");

if (CheckRight() != nullptr && !CheckRight()->m\_Printed)

{

Right();

GetValue('a');

continue;

}

if (CheckRight() == nullptr)

printf("\*\*\n");

Parent();

if (m\_Current == 0 && CheckRight() == nullptr && CheckLeft() == nullptr)

break;

else if (m\_Current == 0 && CheckRight() == nullptr && CheckLeft() != nullptr)

{

if (CheckLeft()->m\_Printed)

break;

else

continue;

}

else if (m\_Current == 0 && CheckRight() != nullptr && CheckLeft() != nullptr)

{

if (CheckRight()->m\_Printed)

break;

}

}

ResetPrint();

}

void Tree::GenTree()

{

FILE\* pFile = fopen("src\\information\\example2.txt", "r");

File\* f = new File(pFile);

bool check = true;

while (f->m\_Position < f->m\_Size) {

Values values;

if (CheckFirst())

{

f->GenTree(&values);

CreateFirst(values);

}

else {

check = f->GenTree(&values);

if (check && CheckLeft() == nullptr)

{

AddLeft(values);

continue;

}

else

{

if (!check && CheckLeft() == nullptr)

check = f->GenTree(&values);

if (check)

{

AddRight(values);

continue;

}

else

{

Parent();

while ((CheckLeft() != nullptr && CheckRight() != nullptr) && m\_Current != 0)

Parent();

}

}

}

}

}

void Tree::SearchGen()

{

Root();

if (Calculator(&CheckCurrent()->m\_Date, &CheckCurrent()->m\_Death) && IsMoscow(&CheckCurrent()->m\_Place))

GetValue('a');

else

CheckCurrent()->m\_Printed = true;

while (true) {

while (CheckLeft() != nullptr && !CheckLeft()->m\_Printed)

{

Left();

if (Calculator(&CheckCurrent()->m\_Date, &CheckCurrent()->m\_Death) && IsMoscow(&CheckCurrent()->m\_Place))

GetValue('a');

else

CheckCurrent()->m\_Printed = true;

}

if (CheckRight() != nullptr && !CheckRight()->m\_Printed)

{

Right();

if (Calculator(&CheckCurrent()->m\_Date, &CheckCurrent()->m\_Death) && IsMoscow(&CheckCurrent()->m\_Place))

GetValue('a');

else

CheckCurrent()->m\_Printed = true;

continue;

}

Parent();

if (m\_Current == 0 && CheckRight() == nullptr && CheckLeft() == nullptr)

break;

else if (m\_Current == 0 && CheckRight() == nullptr && CheckLeft() != nullptr)

{

if (CheckLeft()->m\_Printed)

break;

else

continue;

}

else if (m\_Current == 0 && CheckRight() != nullptr && CheckLeft() != nullptr)

{

if (CheckRight()->m\_Printed)

break;

}

}

ResetPrint();

}

}

Приложение 3

**console.h**

#pragma once

#include "file.h"

#include "../additional/date.h"

Values CreateValues();

std::string GetString(const std::string& value);

void Header();

void Choice();

void ProgramLoop(const char\* task);

**console.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "console.h"

Values CreateValues()

{

Values values;

printf("Введите имя!\n");

std::cin >> values.m\_Name;

printf("Введите фамилию!\n");

std::cin >> values.m\_LastName;

printf("Введите отчество!\n");

std::cin >> values.m\_MiddleName;

printf("Введите дату рождения!\n");

std::cin >> values.m\_Date;

while (!CheckDate(&values.m\_Date))

{

printf("Пожалуйста введите верную дуту! \n");

std::cin.clear();

std::cin >> values.m\_Date;

}

printf("Введите дату смерти!\n");

std::cin >> values.m\_Death;

if (values.m\_Death.size() > 1) {

while (!CheckDate(&values.m\_Death))

{

printf("Пожалуйста введите верную дуту! \n");

std::cin.clear();

std::cin >> values.m\_Death;

if (values.m\_Death.size() <= 1)

break;

}

}

printf("Введите место рождения!\n");

std::cin >> values.m\_Place;

return values;

}

std::string GetString(const std::string& value)

{

printf("Введите %s", value.c\_str());

std::string str;

std::cin >> str;

std::cin.clear();

return str;

}

void Header()

{

printf("-проверка на пустоту дерева : 1\n"

"-создание пустого дерева : 2\n"

"-сделать текущим корень дерева : 3\n"

"-сделать текущим левое поддерево(от текущей вершины) : 4\n"

"-сделать текущим правое поддерево(от текущей вершины) : 5\n"

"-сделать текущим родительскую вершину для текущей вершины : 6\n"

"-узнать значение текущей вершины : 7\n"

"-изменить значение текущей вершины : 8\n"

"-создать левое поддерево для текущей вершины : 9\n"

"-создать правое поддерево для текущей вершины : A\n"

"-удалить дерево : D\n"

"-создать упорядоченное по ключу : U\n"

"-добавить один элемент, со значениями введенными в интерактивном режиме в упорядоченное дерево по ключу : N\n"

"-вывод от самого старшего до самого младшего : O\n"

"-вывести дерево левое-корень-правое : L\n"

"-вывести дерево корень-левое-правое : P\n"

"-создать неупорядоченное бинарное дерево : B\n"

"-найти всех предков, кто родился в Москве. прожил не менее 70 лет : F\n"

"-очистить консоль : C\n"

"-закончить : E\n");

}

void Choice()

{

printf("Если хотите изменить все значения нажмите : 0\n"

"Для изменения имени : 1\n"

"Для изменения фамилии : 2\n"

"Для изменения отчества : 3\n"

"Для изменения даты рождения : 4\n"

"Для изменения даты смерти : 5\n"

"Для изменения места рождения : 6\n"

"Сброс : E\n");

}

template<typename T>

void Loop(T tree)

{

bool created = false;

Header();

Writer\* w = new Writer();

unsigned char choice;

std::cin >> choice;

while (choice != 'E') {

if (choice != '2' && choice != 'E' && choice != 'U' && choice != 'B' && tree->CheckFirst())

{

printf("Сначала создайте корень дерева! №2!\n");

scanf(" %c", &choice);

continue;

}

switch (choice)

{

case '1':

if (tree->CheckIfEmpty())

{

printf("Дерево пустое\n");

break;

}

printf("Дерево не пустое\n");

w->Print(tree);

break;

case '2':

if (tree->CheckFirst())

{

{tree->CreateFirst(CreateValues()); }

printf("Создан первый элемент\n");

w->Print(tree);

break;

}

printf("Корень уже существует\n");

break;

case '3':

tree->Root();

break;

case '4':

tree->Left();

break;

case '5':

tree->Right();

break;

case '6':

tree->Parent();

break;

case '7':

tree->GetValue('n');

break;

case '8':

{

Choice();

char c;

std::cin >> c;

switch (c)

{

case '0':

{tree->ChangeAll(CreateValues()); }

break;

case '1':

tree->ChangeName(&GetString("имя \n"));

break;

case '2':

tree->ChangeLastName(&GetString("фамилию \n"));

break;

case '3':

tree->ChangeSurname(&GetString("отчество \n"));

break;

case '4':

tree->ChangeDate(&GetString("дату рождения \n"));

break;

case '5':

tree->ChangeDeath(&GetString("дату смерти \n"));

break;

case '6':

tree->ChangePlace(&GetString("место рождения \n"));

break;

case 'E': printf("Сброс \n");

break;

default:

printf("Неверный ввод! Сброс \n");

break;

}

}

w->Print(tree);

break;

case '9':

{tree->AddLeft(CreateValues()); }

w->Print(tree);

break;

case 'A':

{tree->AddRight(CreateValues()); }

w->Print(tree);

break;

case 'D':

tree->Delete();

break;

case 'U':

tree->CreateFromFile();

w->Print(tree);

break;

case 'N':

{tree->AddByKey(CreateValues()); }

w->Print(tree);

break;

case 'O':

tree->PrintByAge();

w->Print(tree);

break;

case 'L':

tree->PrintByAge();

w->Print(tree);

break;

case 'P':

tree->PrintRoot();

w->Print(tree);

break;

case 'B':

tree->GenTree();

w->Print(tree);

break;

case 'F':

tree->SearchGen();

break;

case 'C':

system("CLS");

Header();

w->Print(tree);

break;

default:

break;

}

scanf(" %c", &choice);

}

}

void ProgramLoop(const char\* task)

{

if (\*task == '1')

{

linked\_tree::Tree\* tree = new linked\_tree::Tree();

Loop(tree);

}

else if(\*task == '2')

{

array\_tree::Tree\* tree = new array\_tree::Tree();

Loop(tree);

}

}

**file.h**

#pragma once

#include <iostream>

#include "../array/array.h"

#include "../list/tree.h"

struct Writer

{

FILE\* m\_pFile;

Writer();

void Print(array\_tree::Tree\* tree);

void Print(linked\_tree::Tree\* tree);

template <class T>

void Write(T tree);

~Writer()

{

fclose(m\_pFile);

}

};

**file.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "file.h"

Writer::Writer()

{

m\_pFile = fopen("src\\information\\log.txt", "a+");

}

void Writer::Print(linked\_tree::Tree\* tree)

{

if (tree->CheckFirst())

return;

tree->Root();

Write(tree);

if (tree->CheckIfEmpty())

return;

while (true) {

while (tree->CheckLeft() != nullptr && !tree->CheckLeft()->m\_Printed)

{

tree->Left();

Write(tree);

}

if (tree->CheckLeft() == nullptr && (tree->CheckRight() == nullptr || !tree->CheckRight()->m\_Printed))

fwrite("\*\*\n", sizeof(char), 3, m\_pFile);

if (tree->CheckRight() != nullptr && !tree->CheckRight()->m\_Printed)

{

tree->Right();

Write(tree);

continue;

}

if (tree->CheckRight() == nullptr)

fwrite("\*\*\n", sizeof(char), 3,m\_pFile);

tree->Parent();

if (tree->m\_Current == tree->m\_First && tree->CheckRight() == nullptr && tree->CheckLeft() == nullptr)

break;

else if (tree->m\_Current == tree->m\_First && tree->CheckRight() == nullptr && tree->CheckLeft() != nullptr)

{

if (tree->CheckLeft()->m\_Printed)

break;

else

continue;

}

else if (tree->m\_Current == tree->m\_First && tree->CheckRight() != nullptr && tree->CheckLeft() == nullptr)

{

if (tree->CheckRight()->m\_Printed)

break;

}

else if (tree->m\_Current == tree->m\_First && tree->CheckRight() != nullptr && tree->CheckLeft() != nullptr)

{

if (tree->CheckRight()->m\_Printed && tree->CheckLeft()->m\_Printed)

break;

}

}

tree->ResetPrint();

}

void Writer::Print(linked\_tree::Tree\* tree)

{

if (tree->CheckFirst())

return;

tree->Root();

Write(tree);

if (tree->CheckIfEmpty())

return;

while (true) {

while (tree->CheckLeft() != nullptr && !tree->CheckLeft()->m\_Printed)

{

tree->Left();

Write(tree);

}

if (tree->CheckLeft() == nullptr && (tree->CheckRight() == nullptr || !tree->CheckRight()->m\_Printed))

fwrite("\*\*\n", sizeof(char), 3, m\_pFile);

if (tree->CheckRight() != nullptr && !tree->CheckRight()->m\_Printed)

{

tree->Right();

Write(tree);

continue;

}

if (tree->CheckRight() == nullptr)

fwrite("\*\*\n", sizeof(char), 3,m\_pFile);

tree->Parent();

if (tree->m\_Current == tree->m\_First && tree->CheckRight() == nullptr && tree->CheckLeft() == nullptr)

break;

else if (tree->m\_Current == tree->m\_First && tree->CheckRight() == nullptr && tree->CheckLeft() != nullptr)

{

if (tree->CheckLeft()->m\_Printed)

break;

else

continue;

}

else if (tree->m\_Current == tree->m\_First && tree->CheckRight() != nullptr && tree->CheckLeft() == nullptr)

{

if (tree->CheckRight()->m\_Printed)

break;

}

else if (tree->m\_Current == tree->m\_First && tree->CheckRight() != nullptr && tree->CheckLeft() != nullptr)

{

if (tree->CheckRight()->m\_Printed && tree->CheckLeft()->m\_Printed)

break;

}

}

tree->ResetPrint();

}

template <class T>

void Writer::Write(T tree)

{

fprintf(m\_pFile, "%s\n", tree->CheckCurrent()->m\_Name.c\_str());

fprintf(m\_pFile, "%s\n", tree->CheckCurrent()->m\_LastName.c\_str());

fprintf(m\_pFile, "%s\n", tree->CheckCurrent()->m\_Date.c\_str());

fprintf(m\_pFile, "%s\n", tree->CheckCurrent()->m\_Death.c\_str());

fprintf(m\_pFile, "%s\n", tree->CheckCurrent()->m\_Place.c\_str());

tree->CheckCurrent()->m\_Printed = true;

}

**main.cpp**

#include "logging/console.h"

#include "additional/filereader.h"

#include <windows.h>

void MainHeader() {

printf("=====================================================\n"

"Здравствуйте! Пожалуйста выберите задание!\n"

"«Бинарное дерево»:\n"

"1) на основе ссылочного типа «Нелинейный связный список»\n"

"2) на основе динамического массива\n"

"E) чтобы закончит программу \n"

"=====================================================\n");

}

int main()

{

setlocale(LC\_ALL, "Russian");

MainHeader();

char task;

std::cin >> task;

while (task != 'E') {

if (task == '1')

ProgramLoop(&task);

else if (task == '2')

ProgramLoop(&task);

else

printf("Неверный ввод! \n");

MainHeader();

std::cin >> task;

}

}

Приложение 4 (воспитательные модули)

**values.h**

#pragma once

#define \_CRT\_SECURE\_NO\_WARNINGS

#include <iostream>

struct Values

{

std::string m\_MiddleName;

std::string m\_Name;

std::string m\_LastName;

std::string m\_Date;

std::string m\_Death;

std::string m\_Place;

bool m\_Printed = false;

Values() {}

Values(Values\* values)

{

m\_MiddleName = values->m\_MiddleName;

m\_Name = values->m\_Name;

m\_LastName = values->m\_LastName;

m\_Date = values->m\_Date;

if (!values->m\_Death.empty())

m\_Death = values->m\_Death;

if (!values->m\_Place.empty())

m\_Place = values->m\_Place;

}

};

**filereader.h**

#pragma once

#include "../logging/console.h"

#include <iostream>

struct File

{

FILE\* m\_pFile;

std::string m\_Result;

unsigned long m\_Size;

unsigned long m\_Position;

char\* m\_Value;

File(FILE\* pFile);

void ToObject(Values\* values);

char\* ReadLine();

bool GenTree(Values\* values);

~File()

{

fclose(m\_pFile);

}

};

**filereader.cpp**

#define \_CRT\_SECURE\_NO\_WARNINGS

#include "filereader.h"

File::File(FILE\* pFile)

: m\_pFile(pFile)

{

fseek(pFile, 0, SEEK\_END);

m\_Size = ftell(pFile);

m\_Position = 0;

fseek(pFile, 0, SEEK\_SET);

m\_Value = new char[m\_Size];

memset(m\_Value, 0, m\_Size \* (sizeof m\_Value[0]));

fread(m\_Value, 1, m\_Size, pFile);

}

void File::ToObject(Values\* values)

{

values->m\_Name = ReadLine();

values->m\_MiddleName = ReadLine();

values->m\_LastName = ReadLine();

values->m\_Date = ReadLine();

values->m\_Death = ReadLine();

values->m\_Place = ReadLine();

}

char\* File::ReadLine()

{

unsigned int size = 0;

while (m\_Value[m\_Position + size] != '\n' && feof(m\_pFile) != 0)

{

size++;

}

char\* line = new char[size];

unsigned int k = 0;

for (unsigned int i = m\_Position; i < m\_Position + size; i++) {

line[k] = m\_Value[i];

k++;

}

m\_Position += size + 1;

line[size] = 0;

return line;

}

bool File::GenTree(Values\* values)

{

values->m\_Name = ReadLine();

if (values->m\_Name == "\*\*" || feof(m\_pFile) == 0)

{

return false;

}

values->m\_MiddleName = ReadLine();

values->m\_LastName = ReadLine();

values->m\_Date = ReadLine();

values->m\_Death = ReadLine();

values->m\_Place = ReadLine();

return true;

}

**dates.h**

#pragma once

#include <iostream>

#include <string>

bool CheckDate(const std::string\* date);

**dates.cpp**

#include "date.h"

bool CheckDate(const std::string\* date)

{

if (date->size() < 10 || date->size() > 11)

return false;

try

{

if (std::stoi(&date->at(0)) > 31 || std::stoi(&date->at(3)) > 12 || std::stoi(&date->at(6)) > CURRENT\_YEAR)

return false;

}

catch (std::invalid\_argument& e) { return false; }

return true;

}

**calculator.h**

#include <string>

#define TODAY "30.05.2020."

bool Calculator(const std::string\* date\_root, const std::string\* date\_new);

bool IsMoscow(const std::string\* place);

**calculator.cpp**

#include "calculator.h"

bool Calculator(const std::string\* date\_root, const std::string\* date\_new)

{

char yyyy\_root[4];

char yyyy\_new[4];

for (int i = 0; i < 4; i++)

{

yyyy\_root[i] = date\_root->at(i + 6);

if (date\_new->size() < 10)

yyyy\_new[i] = TODAY[i + 6];

else

yyyy\_new[i] = date\_new->at(i + 6);

}

if (std::stoi(yyyy\_new) - std::stoi(yyyy\_root) >= 70)

return true;

return false;

}

bool IsMoscow(const std::string\* place)

{

if (place->compare("Москва") == 0 || place->compare("Moskva") == 0 || place->compare("Moscow") == 0)

return true;

return false;

}

Приложение 5

Пример получившегося лог-файла

Дарья

Клубничкина

18 09 1998.

-

-

Андрей

Мамонтов

05.04.1978.

-

-

Александр

Кунцевич

05.06.1942.

-

Москва

Пётр

Игоревич

05.08.1902.

06.05.2000.

-

Алексей

Есин

05.05.1850.

05.05.1950.

Москва

Пётр

Петров

05.05.1000.

-

-

\*\*

\*\*

\*\*

\*\*

Роман

Востриков

05.01.1950.

x

x

\*\*

\*\*

Елена

Кунцевич

05.06.1980.

-

-

\*\*

\*\*

Иван

Гусаев

05.02.2003.

-

-

\*\*

Егор

Ходунов

18.09.2004.

-

-

\*\*

\*\*

Daria

Aleksandovna

05-08-2005

-

-

Андрей

Мамонтов

05.04.1978.

-

-

Александр

Кунцевич

05.06.1942.

-

Москва

Пётр

Игоревич

05.08.1902.

06.05.2000.

-

Алексей

Есин

05.05.1850.

05.05.1950.

Москва

Пётр

Петров

05.05.1000.

-

-

\*\*

\*\*

\*\*

\*\*

Роман

Востриков

05.01.1950.

x

x

\*\*

\*\*

Елена

Кунцевич

05.06.1980.

-

-

\*\*

\*\*

Иван

Гусаев

05.02.2003.

-

-

\*\*

Егор

Ходунов

18.09.2004.

-

-

\*\*

\*\*

Daria

Aleksandovna

05-08-2005

-

-

Андрей

Мамонтов

05.04.1978.

-

-

Александр

Кунцевич

05.06.1942.

-

Москва

Пётр

Игоревич

05.08.1902.

06.05.2000.

-

Алексей

Есин

05.05.1850.

05.05.1950.

Москва

Пётр

Петров

05.05.1000.

-

-

\*\*

\*\*

\*\*

\*\*

Роман

Востриков

05.01.1950.

x

x

\*\*

\*\*

Елена

Кунцевич

05.06.1980.

-

-

\*\*

\*\*

Иван

Гусаев

05.02.2003.

-

-

\*\*

Егор

Ходунов

18.09.2004.

-

-

\*\*

\*\*

Daria

Aleksandovna

05-08-2005

-

-

Андрей

Мамонтов

05.04.1978.

-

-

Александр

Кунцевич

05.06.1942.

-

Москва

Пётр

Игоревич

05.08.1902.

06.05.2000.

-

Алексей

Есин

05.05.1850.

05.05.1950.

Москва

Пётр

Петров

05.05.1000.

-

-

\*\*

\*\*

\*\*

\*\*

Роман

Востриков

05.01.1950.

x

x

\*\*

\*\*

Елена

Кунцевич

05.06.1980.

-

-

\*\*

\*\*

Иван

Гусаев

05.02.2003.

-

-

\*\*

Егор

Ходунов

18.09.2004.

-

-

\*\*

\*\*

Daria

Aleksandovna

05-08-2005

-

-

Андрей

Мамонтов

05.04.1978.

-

-

Александр

Кунцевич

05.06.1942.

-

Москва

Пётр

Игоревич

05.08.1902.

06.05.2000.

-

Алексей

Есин

05.05.1850.

05.05.1950.

Москва

Пётр

Петров

05.05.1000.

-

-

\*\*

\*\*

\*\*

\*\*

Роман

Востриков

05.01.1950.

x

x

\*\*

\*\*

Елена

Кунцевич

05.06.1980.

-

-

\*\*

\*\*

Иван

Гусаев

05.02.2003.

-

-

\*\*

Егор

Ходунов

18.09.2004.

-

-

\*\*

\*\*