Федеральное государственное бюджетное образовательное учреждение высшего профессионального образования
Национальный исследовательский университет «МЭИ»

Типовой расчет «Моделирование типов»
Вариант 24

Задание выполнил:
Черкасов Александр Андреевич
Студент группы А-06-19

Проверил(а):_____
Оценка:_____
Замечания:_____
_____
_____
_____
_____
_____
_____

Москва 2020 НИУ «МЭИ»

Часть 1:
Задание:
Смоделировать двумя способами новый тип «Бинарное дерево»:
1) на основе ссылочного типа «Нелинейный связный список»;

2) на основе динамического массива или типизированного/двоичного файла (на выбор студента).

Для этого создать модуль и описать новый тип данных и его структуру (на русском языке и на Delphi/C, без классов! Только записи/структуры и массив/файл) и базовые операции:
- проверка на пустоту дерева (поддерева, начинающегося с текущей вершины);
- создание пустого дерева или дерева из одного элемента (как удобнее);
- сделать текущим (по адресу в списке или индексу в массиве) корень дерева;
- сделать текущим левое поддерево (от текущей вершины);
- сделать текущим правое поддерево (от текущей вершины);
- 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);
- узнать значение текущей вершины;
- изменить значение текущей вершины;
- создать левое поддерево для текущей вершины;
- создать правое поддерево для текущей вершины;

Выбранный 2й способ: бинарный файл

Использованные структуры:

```
struct date       //Дата
{
  int day;      //День
  int month;    //Месяц
  int year;     //Год
};
struct data       //Данные
{
  char*  FName;  //Имя
  char*  SName;  //Фамилия
  char*  LName;  //Отчество
  date  dBirth;  //Дата рождения
  date  dDeath;  //Смерти (опционально)
  char* BPlace;  //Место рождения(опционально)
};
```

Специфическая структура для дерева на основе списка

```
 struct Node        //Структура одной ветви
{
   data info;     //Данные
   Node *nextL;    //Указатель на левый элемент
   Node *nextR;    //Указатель на правый элемент
   Node *prev;     //Указатель на предыдущий элемент
};
```

Основные функции:

Названия:

Список:

```
//Указатель на ветвь называем NodePtr
typedef Node* NodePtr;
```

Файл:

```
//"Указатель" на ветвь содержит текущую позицию и указатель на файл, к которому принадлежит
typedef std::pair<int,FILE*> NodePtr;
```

Инициализация:

Список:

```
void Init()
{
   //Создаем на корне пустые элементы и присваиваем неотсортированному стоп-знак
   head = NewNode(nullptr);
   headKey = NewNode(nullptr);
   head->info={(char*)"**"};
}
```

Файл:

```
void Init()
{
   //Открываем соответствующие файлы
   pTreeUnsorted = fopen("C:\\Users\\Public\\Documents\\treeUnsorted.bin","wb+");
   pTreeSorted = fopen("C:\\Users\\Public\\Documents\\treeSorted.bin","wb+");
   //Инициализируем корни
   head = std::make_pair(1,pTreeUnsorted);
   headKey = std::make_pair(1,pTreeSorted);
   //Записываем в корни пустые данные
```

```
   replDataM(&head,nullData);
   replDataM(&headKey,nullData);
}
```

Новый элемент:

NodePtr (Node **CurrentPosition, char Pos='0')

Node ** CurrentPosition – «указатель на указатель» текущего положения в дереве

char Pos – символ, определяющий положение, на которой будет стоять новый элемент, относительно текущего

Возвращает указатель на созданный элемент

Список:

```
//Создание новой ветви, в зависимости от символа будет правым или левым или же "основой"
NodePtr NewNode(Node **CurrentPosition, char Pos='0')
{
   Node *newNode = new Node;

   newNode->nextL=nullptr;
   newNode->nextR=nullptr;
   //Создание пустых связей
   //Проверка на наличие "предшественника"
   newNode->prev = (CurrentPosition == nullptr ? nullptr : (*CurrentPosition));
   //Добавление связи в зависимости от того, куда необходимо добавить новую ветвь
   if (Pos=='L') (*CurrentPosition)->nextL=newNode;
   if (Pos=='R') (*CurrentPosition)->nextR=newNode;
   //Записываем пустые данные
   newNode->info=nullData;
   return newNode;
}
```

Файл:

```
//Создание новой ветви, в зависимости от символа будет правым или левым или же "основой"
NodePtr NewNode(NodePtr *CurrentPosition, char Pos='0')
{
//Пользуясь тем, что в файле не надо дополнительно выделять память, возвращаем нужную позицию
   if (Pos=='L') return curL(*CurrentPosition); //Возвращение «левой» позиции
   else if (Pos=='R') return curR(*CurrentPosition); //Возвращение «правой» позиции
```

```
  else return (*CurrentPosition); //Возвращение самого себя
}
```

Замена данных:

Все три функции имеют вид:

void (NodePtr *cur,data inf)

NodePtr *cur – «указатель на указатель» текущего положения

data inf – данные для записи

Текущая ветвь

Список:

```
void replDataM(NodePtr *cur,data inf){
  (*cur)->info=inf;
}
```

Файл:

```
void replDataM(NodePtr *cur,data inf) {
  fseek((*cur).second, (*cur).first * sizeof(data), SEEK_SET); //Встаем на позицию ветви
  fwrite(&inf, sizeof(data), 1, (*cur).second); //Записывам
}
```

Левая ветвь

Список:

```
void replDataL(NodePtr *cur,data inf){
  (*cur)->nextL = NewNode(cur); //Создание элемента слева, заодно и «переход»
  (*cur)->nextL->info=inf;
}
```

Файл:

```
void replDataL(NodePtr *cur,data inf) {
  fseek((*cur).second, (*cur).first * 2 * sizeof(data), SEEK_SET); //Встаем на позицию ветви
  fwrite(&inf, sizeof(data), 1, (*cur).second); //Запись
}
```

Правая ветвь

Список:

```
void replDataR(NodePtr *cur,data inf){
  (*cur)->nextR = NewNode(cur); //Создание элемента справа, заодно и «переход»
```

```
  (*cur)->nextR->info=inf;
}
```

Файл:

```
void replDataR(NodePtr *cur,data inf) {
  fseek((*cur).second, ((*cur).first * 2 + 1) * sizeof(data), SEEK_SET); //Встаем на позицию ветви
  fwrite(&inf, sizeof(data), 1, (*cur).second); //Запись
}
```

Сделать текущим левое поддерево

NodePtr (NodePtr cur)

NodePtr cur – текущее положение

Возвращает указатель на левое поддерево

Список:

```
NodePtr curL(NodePtr cur){
  return cur->nextL; //Возврат указателя на левое поддерево
}
```

Файл:

```
NodePtr curL(NodePtr cur){
  return std::make_pair(cur.first*2,cur.second); //Возврат пары левая позиция, указатель на файл
}
```

Сделать текущим правое поддерево

NodePtr (NodePtr cur)

NodePtr cur – текущее положение

Возвращает указатель на правое поддерево

Список:

```
NodePtr curR(NodePtr cur){
  return cur->nextR; //Возврат указателя на правое поддерево
}
```

Файл:

```
NodePtr curR(NodePtr cur){
  return std::make_pair(cur.first*2+1,cur.second); //Возврат пары правая позиция, указатель на файл
}
```

Сделать текущим родительскую вершину:

Обе функции имеют вид

NodePtr ()

Возвращает указатель на корень дерева

Генеалогическое дерево

Список:

```
//Указатель на корень Геологического дерева
NodePtr getStartUnsorted(){
  return head;
}
```

Файл:

```
//Указатель на корень Геологического дерева
NodePtr getStartUnsorted(){
  return head;
}
```

Отсортированное дерево

Список:

```
//Указатель на корень отсортированного дерева
NodePtr getStartSorted(){
  return headKey;
}
```

Файл:

```
//Указатель на корень отсортированного дерева
NodePtr getStartSorted(){
  return headKey;
}
```

Удаление ветви

void (NodePtr *cur)

NodePtr* cur – текущее положение в дереве

Список:

```
//Уничтожение ветви или целого дерева
void destroy(NodePtr *cur)
{
  if (!(*cur)->info.FName) { //Проверка, удалили ли данные раньше
    /*
```

```
        * Нужно в основном для отсортированного дерева
        * т.к. есть элемнты добавленные вручную, а остальные
        * удалены ранее в генеологическом дереве
        */
       delete[]((*cur)->info.FName);
       delete[]((*cur)->info.SName);
       delete[]((*cur)->info.LName);
       delete[]((*cur)->info.BPlace);
       (*cur)->info.FName=nullptr; //для возможности определить был ли удален элемент ранее
    }
    //Если есть что-то слева или справа - уничтожаем
    if ((*cur)->nextL!= nullptr)
       destroy(&((*cur)->nextL));
    if ((*cur)->nextR!= nullptr)
       destroy(&((*cur)->nextR));
    //Если не голова - удаляем текущий элемент и соответствующую связь
    if (*cur!=head and *cur!=headKey){
       Node *temp = (*cur)->prev;
       if (temp->nextL==(*cur)) temp->nextL=nullptr;
       if (temp->nextR==(*cur)) temp->nextR=nullptr;
       delete [](*cur);
       (*cur) = temp;
    }
    else {
       //Очищаем голову и отдаем nullptr
       delete [](*cur);
       (*cur) = nullptr;
    }
}
```

Файл:

```
void destroy(NodePtr *cur) {
    //Если уничтожаем корень, то просто закрываем файл и уничтожаем его
    if (*cur == head) {
       fclose(pTreeUnsorted);
```

```
        remove("C:\\Users\\Public\\Documents\\treeUnsorted.bin");
    } else if (*cur == headKey) {
        fclose(pTreeSorted);
        remove("C:\\Users\\Public\\Documents\\treeSorted.bin");
    } else {
        //Иначе просто записываем в нужный файл на соответствующее место пустые данные
        fseek((*cur).second, (*cur).first * sizeof(data), SEEK_SET);
        fwrite(&nullData, sizeof(data), 1, (*cur).second);
        NodePtr nextL = std::make_pair((*cur).first * 2, (*cur).second);
        NodePtr nextR = std::make_pair((*cur).first * 2 + 1, (*cur).second);
        //и переходим влево и вправо
        if (!IsEmpty(nextL)) destroy(&nextL);
        if (!IsEmpty(nextR)) destroy(&nextR);
    }
}
```

Часть 2:

Задание:

Создать приложение (консольное, на C или Delphi), позволяющее выбрать и выполнить следующие действия над деревом (только с помощью выше перечисленных базовых операций! Без прямого обращения к элементам списка/массива/файла):
- удалить дерево (освободить память);
- создать упорядоченное по ключу (по возрастанию/убыванию одного или нескольких полей записи/структуры) бинарное дерево из текстового файла;
- создать неупорядоченное бинарное дерево (генеалогическое дерево) из текстового файла (Корень – Левое_поддерево – Правое_поддерево, ** – признак пустого поддерева);
- добавить один элемент, со значениями введенными в интерактивном режиме (с клавиатуры) в упорядоченное дерево по ключу (по возрастанию/убыванию одного или нескольких полей записи/структуры) для построения упорядоченного дерева; например, в ниже изображенное дерево добавляются числа 6,3,4,2,6,9 по возрастанию (неубыванию в случае добавления одинаковых значений);
- найти решение задачи из списка заданий по вариантам в конце этого файла (стр.16);
- вывести дерево 2-4мя разными способами: (Корень – Левое_поддерево – Правое_поддерево, Левое_поддерево – Корень – Правое_поддерево, Левое_поддерево – Правое_поддерево – Корень; графически(по желанию). ** – признак пустого поддерева);

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 или последним из вышеуказанных 4-х способов.

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**

24. а) Создав соответствующее упорядоченное дерево, создать на год (с января по декабрь) план поминок усопших родственников. б) По генеалогическому дереву определить, есть ли в Вашем роду кто-нибудь, у кого обоих дедушек звали Иванами

Формат входного файла:

Данные

<FName>

<SName>

<LName>

<dBitrh.day>

<dBitrh.month>

<dBitrh.year>

<dDeath.day>

<dDeath.month>

<dDeath.year>

<BPlace>

При желании перейти на другую ветвь ставить стоп-знак

<**>

Описание меню

'v' - вывести деревья на экран

Предлагает на выбор 3 возможных способа вывода дерева; функции имеют вид:

void (NodePtr TreeHead,FILE* pfile,int depth)

NodePtr TreeHead – указатель на текущее положение в дереве;

FILE *pfile – указатель на поток, в который записывается инфомация

int depth – глубина текущего положения в дереве

'p' - вывести план поминок

Функция имеет вид:

void (NodePtr TreeHead, FILE *pfile)

NodePtr TreeHead – указатель на текущее положение в дереве;

FILE *pfile – указатель на поток, в который записывается инфомация

'a' - добавить к дереву по ключу

Сперва вызывается

data readData(char* FName, FILE* FileIn)

char* FName – Имя элемента. Вынесена в параметр, из-за использования при считывании данных с текстового файла

FILE* FileIn – поток, с которого считываются данные

Возвращает данные в формате структуры data

Если будут ошибки с форматом даты или другая надпись вместо даты – вызов кода ошибки 2

Потом

void AddToTreeKey(data currentData)

data CurrentData -данные, которые нужно записать в дерево

't' - выполнить поиск человека, у которого обоих дедушек зовут Иванами

Функция имеет вид:

int (NodePtr CurrentPosition)

NodePtr CurrentPosition – текущая позиция в дереве.

Возвращает 1, если найден человек, у которого обоих дедушек зовут Иванами

Иначе 0

'r' - пересоздать деревья

Сперва вызывается для обоих деревье

void remove(NodePtr *cur)

NodePtr *cur – «указатель на указатель» текущего положения дерева, которого нужно очистить

Снова инициализируются деревья

void Init()

Вызываются создания деревьев

void MakeTree(char* filename)

char* filename – имя входного файла

void MakeTreeKey()

'x' - завершение программы

Переключает флаг end основного цикла на 1 (цикл работает на !end)

Функциональные тесты:

Ремарка для удобства восприятия, все деревья выводятся в режиме Лево-Корень-Право

№	Входные данные	Ожидаемый результат	Смысл теста

Аномалии			
0	В параметрах при запуске указать неправильную папку логов	Сообщение «Невозможно создать Лог-файл» Завершение программы	Проверка поведении программы на невозможность создать лог-файл
1	В параметрах при запуске указать не существующий входной файл	Сообщение «Невозможно прочитать входной файл» Завершение программы	Проверка поведении программы на невозможность прочитать входные данные
2	Человек Случайный Корень Точно 11 2000 1 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных День в дате рождения
2.1	Человек Случайный Корень -10 11 2000 1 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Отрицательное число в дне даты рождения
2.2	Человек Случайный Корень 35 11 2000 1 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Число больше 31 в дне даты рождения
3	Человек Случайный Корень 11 Точно 2000 1	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Месяц в дате рождения

	2 2060 Doma ** **		
3.1	Человек Случайный Корень 11 -1 2000 1 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Отрицательное число в месяце даты рождения
3.2	Человек Случайный Корень 11 30 2000 1 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Число больше 12 в месяце даты рождения
4	Человек Случайный Корень 11 2 Точно 1 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Год в дате рождения
4.1	Человек Случайный Корень 1 2 -2000 11 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Отрицательный год в дате рождения

5	Человек Случайный Корень 1 2 2000 Точно 11 2000 Doma ** **	Генеалогическое дерево -- Случайный Человек Корень Родился 1.2.2000 Родился в Doma ---------- Упорядоченное дерево **	Проверка поведении программы на неверный формат входных данных День в дате смерти
5.1	Человек Случайный Корень 10 11 2000 -1 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Отрицательное число в дне даты смерти
5.2	Человек Случайный Корень 10 11 2000 45 2 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Число больше 31 в дне даты смерти
6	Человек Случайный Корень 1 2 2000 11 Точно 2000 Doma ** **	Генеалогическое дерево -- Случайный Человек Корень Родился 1.2.2000 Родился в Doma ---------- Упорядоченное дерево **	Проверка поведении программы на неверный формат входных данных Месяц в дате смерти
6.1	Человек Случайный Корень 11 1	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных

	2000 1 -2 2060 Doma ** **		Отрицательное число в месяце даты смерти
6.2	Человек Случайный Корень 11 12 2000 1 20 2060 Doma ** **	Сообщение «Неверный формат даты» Завершение программы	Проверка поведении программы на неверный формат входных данных Число больше 12 в месяце даты смерти
7	Человек Случайный Корень 1 2 2000 11 2 Точно Doma ** **	Генеалогическое дерево -- Случайный Человек Корень Родился 1.2.2000 Родился в Doma ---------- Упорядоченное дерево **	Проверка поведении программы на неверный формат входных данных Год в дате смерти
7.1	Человек Случайный Корень 1 2 2000 11 2 -200 Doma ** **	Генеалогическое дерево -- Случайный Человек Корень Родился 1.2.2000 Родился в Doma ---------- Упорядоченное дерево **	Проверка поведении программы на неверный формат входных данных Отрицательный год в дате смерти
8	Человек Корневой Первый 1 3 1990 11 12 2040 --	Сообщение «Попытка записи в существующий блок» Завершение программы	Попытка записать в уже созданное генеалогическое дерево

	** ** Человек Корневой Первый 1 3 1990 11 12 2040 -- ** **		
Построение дерева			
9	**	Генеалогическое дерево ** ---------- Упорядоченное дерево ** Task Невозможно выполнить поиск на пустом дереве Нет такого человека, у которого обоих дедушек зовут Иванами	Пустые данные
10	Человек Случайный Корень 1 2 2000 11 2 2060 Doma ** **	- Генеалогическое дерево -- Случайный Человек Корень Родился 1.2.2000 Умер 11.2.2060 Родился в Doma ---------- Упорядоченное дерево -- Случайный Человек Корень Родился 1.2.2000 Умер 11.2.2060 Родился в Doma	Один элемент с датой смерти
11	Человек Случайный Корень 1 2 2000 0 0 0 Doma ** **	Генеалогическое дерево -- Случайный Человек Корень Родился 1.2.2000 Умер 11.2.2060 Родился в Doma ---------- Упорядоченное дерево **	Один элемент без даты смерти
12	Человек Тестовый Корневой 11	Генеалогическое дерево -- Тестовый Человек Слева Родился 16.11.2010 Умер 1.1.12 Родился в Городе	Построение левой ветки

	11 2000 10 12 2060 Городе Человек Тестовый Слева 16 11 2010 1 1 12 Городе ** ** **	-- Тестовый Человек Корневой Родился 11.11.2000 Умер 10.12.2060 Родился в Городе ---------- Упорядоченное дерево -- Тестовый Человек Слева Родился 16.11.2010 Умер 1.1.12 Родился в Городе -- Тестовый Человек Корневой Родился 11.11.2000 Умер 10.12.2060 Родился в Городе	
13	Человек Тестовый Корневой 11 11 2000 1 2 2060 Городе ** Человек Тестовый Справа 16 11 2010 10 11 12 Городе ** **	Генеалогическое дерево -- Тестовый Человек Корневой Родился 11.11.2000 Умер 1.2.2060 Родился в Городе -- Тестовый Человек Справа Родился 16.11.2010 Умер 10.11.12 Родился в Городе ---------- Упорядоченное дерево -- Тестовый Человек Корневой Родился 11.11.2000 Умер 1.2.2060 Родился в Городе -- Тестовый Человек Справа Родился 16.11.2010 Умер 10.11.12 Родился в Городе	Построение правой ветки
Добавление элемента			
14	Входной файл ** Консоль: Человек Корневой Первый 1 3 1990 11	Генеалогическое дерево ** ---------- Упорядоченное дерево -- Человек Корневой Первый Родился 1.3.1990 Умер 11.12.2040 План поминок: Человек Корневой Первый | Умер 11.12.2040	Добавление элемента к пустому дереву

	12 2040 --		
15	Входной файл Консоль: Человек Корневой Первый 1 3 1990 11 12 2040 -- ** ** Консоль: Человек Добавленный Второй 1 3 1990 11 11 2040	Генеалогическое дерево ** ---------- Упорядоченное дерево -- Человек Добавленный Второй Родился 1.3.1990 Умер 11.11.2040 -- Человек Корневой Первый Родился 1.3.1990 Умер 11.12.2040 План поминок: Человек Добавленный Второй | Умер 11.11.2040 Человек Корневой Первый | Умер 11.12.2040	Более ранняя дата смерти Добавлен в левую подветвь
16	Входной файл Консоль: Человек Корневой Первый 1 3 1990 11 12 2040 -- ** ** Консоль: Человек Добавленный Второй 1 3 1990 11 12 2040	Генеалогическое дерево ** ---------- Упорядоченное дерево -- Человек Корневой Первый Родился 1.3.1990 Умер 11.12.2040 -- Человек Добавленный Второй Родился 1.3.1990 Умер 11.12.2040 План поминок: Человек Корневой Первый | Умер 11.12.2040 Человек Добавленный Второй | Умер 11.12.2040	Такая же дата смерти Добавлен в правую подветвь
17	Входной файл Консоль:	Генеалогическое дерево **	Более поздняя дата смерти

	Человек Корневой Первый 1 3 1990 11 12 2040 -- ** ** Консоль: Человек Добавленный Второй 1 3 1990 30 12 2040	---------- Упорядоченное дерево -- Человек Корневой Первый Родился 1.3.1990 Умер 11.12.2040 -- Человек Добавленный Второй Родился 1.3.1990 Умер 30.12.2040 План поминок: Человек Корневой Первый | Умер 11.12.2040 Человек Добавленный Второй | Умер 30.12.2040	Добавлен в правую подветвь
Поиск человека, у которого обоих дедушек зовут Иванами			
18	Человек Случайный Корень 1 2 2000 11 2 2060 Doma ** **	Генеалогическое дерево -- Человек Случайный Корень Родился 1.2.2000 Умер 11.2.2060 Task: Нет такого человека, у которого обоих дедушек зовут Иванами	Поиск с недостающим количеством элементов
19	Человек Тестовый Корневой 11 11 2000 0 0 0 -- Человек Тестовый Слева 16 11 1980 0 0	Генеалогическое дерево -- Тестовый Олег СлеваЛева Родился 12.5.1950 -- Тестовый Человек Слева Родился 16.11.1980 -- Тестовый Человек Корневой Родился 11.11.2000 -- Тестовый Олег СлеваПрава Родился 12.7.1951 -- Тестовый Человек Справа Родился 16.10.1980 Task Нет такого человека, у которого обоих дедушек зовут Иванами	Нет нужного элемента

	0 -- Олег Тестовый СлеваЛева 12 5 1950 0 0 0 -- ** ** ** Человек Тестовый Справа 16 10 1980 0 0 0 -- Олег Тестовый СлеваПрава 12 7 1951 0 0 0 -- ** ** **		
20	Человек Тестовый Корневой 11 11 2000 0 0 0 -- Человек Тестовый Слева 16 11	Генеалогическое дерево -- Тестовый Иван СлеваЛева Родился 12.5.1950 -- Тестовый Человек Слева Родился 16.11.1980 -- Тестовый Человек Корневой Родился 11.11.2000 -- Тестовый Иван СлеваПрава Родился 12.7.1951 -- Тестовый Человек Справа Родился 16.10.1980 Task Найден человек, у которого обоих дедушек зовут Иванами	Есть нужный элемент Успешно найден

	1980 0 0 0 -- Иван Тестовый СлеваЛева 12 5 1950 0 0 0 -- ** ** ** Человек Тестовый Справа 16 10 1980 0 0 0 -- Иван Тестовый СлеваПрава 12 7 1951 0 0 0 -- ** ** **		
21	Человек Тестовый Корневой 11 11 2000 0 0 0 -- Человек Тестовый	Генеалогическое дерево -- Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000 -- Тестовый Иван СлеваЛева Родился 12.5.1950 -- Тестовый Человек Слева Родился 16.11.1980 -- Тестовый Иван ЛСЛ Родился 12.5.1920 Умер 14.1.2000 -- Тестовый Олег СправаЛева Родился 12.5.1950	Два нужных элемента В логе (см. приложение 4) видно, что дальше первого элемента не шли

	Слева 16 11 1980 0 0 0 -- Иван Тестовый СлеваЛева 12 5 1950 0 0 0 -- Иван Тестовый ЛЛЛ 12 5 1920 14 1 2000 -- ** ** ** Олег Тестовый СправаЛева 12 5 1950 0 0 0 -- Иван Тестовый ЛСЛ 12 5 1920 14 1 2000 -- ** **	-- Тестовый Человек Корневой Родился 11.11.2000 -- Тестовый Иван СлеваПрава Родился 12.5.1950 -- Тестовый Человек Справа Родился 16.11.1980 Task Найден человек, у которого обоих дедушек зовут Иванами	

	** Человек Тестовый Справа 16 11 1980 0 0 0 -- Иван Тестовый СлеваПрава 12 5 1950 0 0 0 -- ** ** **		
22	Человек Тестовый Корневой 11 11 2000 0 0 0 -- Человек Тестовый Слева 16 11 1980 0 0 0 -- Олегжа Тестовый СлеваЛева 12 5 1950 0 0 0	Генеалогическое дерево -- Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000 -- Тестовый Олегжа СлеваЛева Родился 12.5.1950 -- Тестовый Человек Слева Родился 16.11.1980 -- Тестовый Иван ЛСЛ Родился 12.5.1920 Умер 14.1.2000 -- Тестовый Олег СправаЛева Родился 12.5.1950 -- Тестовый Человек Корневой Родился 11.11.2000 -- Тестовый Иван СлеваПрава Родился 12.5.1950 -- Тестовый Человек Справа Родился 16.11.1980 Task Найден человек, у которого обоих дедушек зовут Иванами	Нужный элемент не первый Успешно найден

	-- Иван Тестовый ЛЛЛ 12 5 1920 14 1 2000 -- ** ** ** Олег Тестовый СправаЛева 12 5 1950 0 0 0 -- Иван Тестовый ЛСЛ 12 5 1920 14 1 2000 -- ** ** ** Человек Тестовый Справа 16 11 1980 0 0 0 -- Иван Тестовый СлеваПрава 12 5 1950		

	0 0 0 -- ** ** **		

Код для решения задачи

```
/*
Ремарка. PutInLog(char* info) определен в доп. заголовочном файле. Просто выводит информацию в лог-файл. Код в приложении
*/
//Поиск человека, у которого обоих дедушек зовут Иванами
int Task(NodePtr CurrentPosition)
{
  //Проверка на пустоту текущей позиции
  if (IsEmpty(CurrentPosition) or strcmp(getData(CurrentPosition).FName,"**")==0)
  {
    printf("Невозможно выполнить поиск на пустом дереве\n");
    PutInLog((char*)"Функция вызвана на путом дереве\n");
  } else {
    PutInLog((char *) "Проверка элемента ");
    ShowData(getData(CurrentPosition), pLog); //Показ элемента в лог
    //Проверка на наличие родителей и дедушек
    if ( !IsEmpty(curL(CurrentPosition)) and !IsEmpty(curR(CurrentPosition)) and
      strcmp(getData(curL(CurrentPosition)).FName, "**") != 0 and
      strcmp(getData(curR(CurrentPosition)).FName, "**") != 0 and
      !IsEmpty(curL(curL(CurrentPosition))) and !IsEmpty(curR(curL(CurrentPosition)))
        )
    {
      PutInLog((char *) "У элемента найдены дедушки:\n");
      //Считываение дедушек
      data grandL = getData(curL(curL(CurrentPosition)));
      data grandR = getData(curL(curR(CurrentPosition)));
      ShowData(grandL,pLog); //Показ элемента в лог
      ShowData(grandR,pLog);
```

```
        //Проверка имени
        if (strcmp(grandL.FName, "Иван") == 0 and strcmp(grandR.FName, "Иван") == 0)
        {
          PutInLog((char *) "Элемент соответсвует поиску\n");
          return 1;
        }
        PutInLog((char *) "Не соответствует поиску\n");
      }
      else {
        PutInLog((char *) "У элемента нет дедушек\n");
      }
      //Если элемент не прошел проверку, пускаем по следующим элементам
      return ((
            (!IsEmpty(curL(CurrentPosition))) and (Task(curL(CurrentPosition))))
        or ((!IsEmpty(curR(CurrentPosition))) and (Task(curR(CurrentPosition))))
      );
    }
    return 0;
}
//Инициализатор создания отсортированного дерева
void MakeTreeKey()
{
    PutInLog((char*)"Построение отсортированного дерева\n");
    TreeSortedBranches(getStartUnsorted()); //Создание ветвей
    PutInLog((char*)"Создано дерево:\n");
    ShowTree1(getStartSorted(), pLog);//вывод в лог
    PutInLog((char*)"План поминок выглядит:\n");
    ShowTree(getStartSorted(), pLog);  //Вывод в лог дерева в виде «плана поминок»
}

// Рекурсия для построения отсортированного дерева
void TreeSortedBranches(NodePtr CurrentPositionOfUnsorted)
{
    data currentData = getData(CurrentPositionOfUnsorted);
```

```
  if (strcmp(currentData.FName,"**")!=0 and !IsEmpty(CurrentPositionOfUnsorted) and
currentData.dDeath.day) { //Проверка на пустоту и наличие даты смерти
    AddToTreeKey(currentData); //Добавление элемента
  }
  if (!IsEmpty(curL(CurrentPositionOfUnsorted)) and
    strcmp(getData(curL(CurrentPositionOfUnsorted)).FName, "**") != 0)
    //Если слева не пусто - вызываем для левой части
    TreeSortedBranches(curL(CurrentPositionOfUnsorted));
  if (!IsEmpty(curR(CurrentPositionOfUnsorted)) and
    strcmp(getData(curR(CurrentPositionOfUnsorted)).FName, "**") != 0)
    //Если справа не пусто - вызываем для правой части
    TreeSortedBranches(curR(CurrentPositionOfUnsorted));
}

/*
 * Добавление элемента в отсортированное дерево
 * Входной параметр - данные для записи
 */
void AddToTreeKey(data currentData)
{
  PutInLog((char*)"Элемент: ");
  ShowData(currentData,pLog);
  NodePtr CurrentPosition=getStartSorted(); //Начинаем с корня
  while (!IsEmpty(CurrentPosition) and strcmp(getData(CurrentPosition).FName,"**")!=0) //Пока не пустой
элемент
  {
    PutInLog((char*)"Сравнение текущего элемента с ");
    data prevData = getData(CurrentPosition);
    ShowData(prevData,pLog);
    //Сравнение текущего элемента с новым и определение его влево или вправо
    if ((prevData.dDeath.day-currentData.dDeath.day)+
      (prevData.dDeath.month-currentData.dDeath.month)*31>0)
    {
      CurrentPosition = curL(CurrentPosition);
      PutInLog((char*)"Переход влево\n");
```

```
    }
    else {
      CurrentPosition = curR(CurrentPosition);
      PutInLog((char*)"Переход вправо\n");
    }
  }
  //Запись
  replDataM(&CurrentPosition,currentData);
  PutInLog((char*)"Элемент записан\n");
  //Предопределение, что следующие пустые, потом по необходимости заменяются
  replDataR(&CurrentPosition,nullData);
  replDataL(&CurrentPosition,nullData);
}
```

Заключение:

Выполняя данную работу, использовалось два способа создания дерева: с помощью списка и с помощью файла. Обнаруженные отличия:

1.Различные способы хранения информации в памяти. Первый способ хранит информацию в оперативной памяти, второй на диске.

2.Информация, передаваемая в «NodePtr». Если в списке передается указатель на элемент списка, то при реализации через файл требуется передать номер текущего положения и указатель на файл, в который записывается информация. Из-за этого пришлось использовать typedef, чтобы в основной программе не было отличий для разных реализаций

3.Разница в логике реализации, если бинарный файл можно представить как своеобразный массив, с которым все привыкли работать, то способ работы со списком не всем понятен, особенно трехсвязным

Так же у меня были попытки работать с динамическим массивом. Не очень удобно и эффективно с ним работать, т.к. постоянно приходится пере выделять память. Нельзя выделить память через new, можно выделить слишком много или недостаточно – оба случая не приятны

В итоге, по удобности лично для меня на первом месте стоит бинарный файл, потом стоит массив (если бы не пере выделение памяти, то можно было на одном уровне поставить),и на последнем список (т.к. его реализация требует больше действий). Однако для программы файл не самый лучший вариант из-за способа хранения информации, ей было бы проще обращаться напрямую к оперативной памяти, а не постоянно считывать информацию с диска.

Приложения:

```
/*
Ремарка "add_Structs.h" содержит структуры data и date, описанные ранее
*/
```

Приложение 1:

```
#include "add_Structs.h"
struct Node       //Структура одной ветыи
{
   data info;     //Данные
   Node *nextL;   //Указатель на левый элемент
   Node *nextR;   //Указатель на правый элемент
   Node *prev;    //Указатель на предыдущий элемент
};

//Указатель на ветвь называем NodePtr
typedef Node* NodePtr;
//Ссылки на деревья
NodePtr head=nullptr; //Неупорядоченное
NodePtr headKey=nullptr; //Упорядоченное

//Создание новой ветви, в зависимости от символа будет правым или левым или же "основой"
NodePtr NewNode(Node **CurrentPosition, char Pos='0')
{
   Node *newNode = new Node;

   newNode->nextL=nullptr;
   newNode->nextR=nullptr;
   //Создание пустых связей
   //Проверка на наличие "предшественника"
   newNode->prev = (CurrentPosition == nullptr ? nullptr : (*CurrentPosition));
   //Добавление связи в зависимости от того, куда неообходимо добавить новую ветвь
   if (Pos=='L') (*CurrentPosition)->nextL=newNode;
   if (Pos=='R') (*CurrentPosition)->nextR=newNode;
   //Записываем пустые данные
```

```
    newNode->info=nullData;
    return newNode;
}

//Уничтожение ветви или целого дерева
void destroy(NodePtr *cur)
{
    if (!(*cur)->info.FName) { //Проверка, удалили ли данные раньше
        /*
         * Нужно в основном для отсортированного дерева
         * т.к. есть элемнты добавленные вручную, а остальные
         * удалены ранее в генеологическом дереве
         */
        delete[]((*cur)->info.FName);
        delete[]((*cur)->info.SName);
        delete[]((*cur)->info.LName);
        delete[]((*cur)->info.BPlace);
        (*cur)->info.FName=nullptr; //для возможности определить был ли удален элемент ранее
    }
    //Если есть что-то слева или справа - уничтожаем
    if ((*cur)->nextL!= nullptr)
        destroy(&((*cur)->nextL));
    if ((*cur)->nextR!= nullptr)
        destroy(&((*cur)->nextR));
    //Если не голова - удаляем текущий элемент и соответствующую связь
    if (*cur!=head and *cur!=headKey){
        Node *temp = (*cur)->prev;
        if (temp->nextL==(*cur)) temp->nextL=nullptr;
        if (temp->nextR==(*cur)) temp->nextR=nullptr;
        delete [](*cur);
        (*cur) = temp;
    }
    else {
        //Очищаем голову и отдаем nullptr
```

```
      delete [](*cur);
      (*cur) = nullptr;
    }
}

//Инициализатор
void Init()
{
   //Создаем на корне пустые элементы и присваиваем неотсортированному стоп-знак
   head = NewNode(nullptr);
   headKey = NewNode(nullptr);
   head->info={(char*)"**"};
}

//Проверка на пустоту
bool IsEmpty(NodePtr nd)
{
   return (nd->info.FName==nullptr) or (nd==nullptr);
}

//Указатель на корень Геологического дерева
NodePtr getStartUnsorted(){
   return head;
}

//Указатель на корень отсортированного дерева
NodePtr getStartSorted(){
   return headKey;
}

//Возврат назад
NodePtr back(NodePtr *cur) {
   return (*cur)->prev;
}
```

```
//Переход влево
NodePtr curL(NodePtr cur){
   return cur->nextL;
}

//Переход вправо
NodePtr curR(NodePtr cur){
   return cur->nextR;
}

//Получение данных
data getData(NodePtr cur){
   return cur->info;
}
//Замена данных:
//        в текущей ветви
void replDataM(NodePtr *cur,data inf){
   (*cur)->info=inf;
}
//        слева
void replDataL(NodePtr *cur,data inf){
   (*cur)->nextL = NewNode(cur); //Создание элемента слева
   (*cur)->nextL->info=inf;
}
//        справа
void replDataR(NodePtr *cur,data inf){
   (*cur)->nextR = NewNode(cur); //Создание элемента справа
   (*cur)->nextR->info=inf;
}
```

Приложение 2:

```
#include <utility>
#include "add_Structs.h"
```

```
//"Указатель" на ветвь содержит текущую позицию и указатель на файл, к которому принадлежит
typedef std::pair<int,FILE*> NodePtr;

FILE *pTreeUnsorted;NodePtr head;
FILE *pTreeSorted;  NodePtr headKey;

//Замена данных:
//        в текущей ветви
void replDataM(NodePtr *cur,data inf) {
   fseek((*cur).second, (*cur).first * sizeof(data), SEEK_SET);
   fwrite(&inf, sizeof(data), 1, (*cur).second);
}
//        слева
void replDataL(NodePtr *cur,data inf) {
   fseek((*cur).second, (*cur).first * 2 * sizeof(data), SEEK_SET);
   fwrite(&inf, sizeof(data), 1, (*cur).second);
}
//        справа
void replDataR(NodePtr *cur,data inf) {
   fseek((*cur).second, ((*cur).first * 2 + 1) * sizeof(data), SEEK_SET);
   fwrite(&inf, sizeof(data), 1, (*cur).second);
}

//Инициализатор
void Init()
{
   //Открываем соответствующие файлы
   pTreeUnsorted = fopen("C:\\Users\\Public\\Documents\\treeUnsorted.bin","wb+");
   pTreeSorted = fopen("C:\\Users\\Public\\Documents\\treeSorted.bin","wb+");
   if (pTreeUnsorted==nullptr or pTreeSorted== nullptr) throw 2;
   //Инициализируем корни
   head = std::make_pair(1,pTreeUnsorted);
   headKey = std::make_pair(1,pTreeSorted);
```

```
   //Записываем в корни пустые данные
   replDataM(&head,nullData);
   replDataM(&headKey,nullData);
}

//Переход влево
NodePtr curL(NodePtr cur){
   return std::make_pair(cur.first*2,cur.second);
}

//Переход вправо
NodePtr curR(NodePtr cur){
   return std::make_pair(cur.first*2+1,cur.second);
}
//Создание новой ветви, в зависимости от символа будет правым или левым или же "основой"
NodePtr NewNode(NodePtr *CurrentPosition, char Pos='0')
{
   if (Pos=='L') return curL(*CurrentPosition);
   else if (Pos=='R') return curR(*CurrentPosition);
   else return (*CurrentPosition);
}

//Получение данных
data getData(NodePtr cur) {
   data currentData;
   //Встаем на нужную позицию и считываем данные
   fseek(cur.second, cur.first * sizeof(data), SEEK_SET);
   fread(&currentData, sizeof(data), 1, cur.second);
   return currentData;
}

//Проверка на пустоту
bool IsEmpty(NodePtr nd)
{
```

```
    return (getData(nd).FName==nullptr);
}

//Уничтожение ветви или целого дерева
void destroy(NodePtr *cur) {
    //Если уничтожаем корень, то просто закрываем файл и уничтожаем его
    if (*cur == head) {
        fclose(pTreeUnsorted);
        remove("C:\\Users\\Public\\Documents\\treeUnsorted.bin");
    } else if (*cur == headKey) {
        fclose(pTreeSorted);
        remove("C:\\Users\\Public\\Documents\\treeSorted.bin");
    } else {
        //Иначе просто записываем в нужный файл на соответствующее место пустые данные
        fseek((*cur).second, (*cur).first * sizeof(data), SEEK_SET);
        fwrite(&nullData, sizeof(data), 1, (*cur).second);
        NodePtr nextL = std::make_pair((*cur).first * 2, (*cur).second);
        NodePtr nextR = std::make_pair((*cur).first * 2 + 1, (*cur).second);
        //и переходим влево и вправо
        if (!IsEmpty(nextL)) destroy(&nextL);
        if (!IsEmpty(nextR)) destroy(&nextR);
    }
}

//Указатель на корень Геологического дерева
NodePtr getStartUnsorted(){
    return head;
}
//Указатель на корень отсортированного дерева
NodePtr getStartSorted(){
    return headKey;
}

//Возврат назад
```

```
NodePtr back(NodePtr *cur) {
   return std::make_pair((*cur).first/2,(*cur).second);
}
```

Приложение 3:

Logger.h

```
#include <ctime>
//#include <string>
/*
 * Собственноручно созданная "библиотека" для логов
 */

FILE *pLog; //Указатель на файл лога, доступен всем

//Открытие файла, передается папка для логоа
void InitLogger(char* LogFolder)
{
   char* logName = new char[255];
   time_t rawtime;
   struct tm * timeinfo;
   char buffer [255];

   time (&rawtime);
   timeinfo = localtime (&rawtime);
   strftime (buffer,255," %d-%b-%Y %H.%M.%S",timeinfo);
   snprintf(logName,255,"%sLog%s.txt",LogFolder,buffer);
   //Имя файла - пака\Log - день-месяц-год час.минута.секунда.txt на момент создания
   pLog = fopen(logName,"w");
   if (pLog==nullptr) throw 0; //Возможность отправить код ошибки, если файл не открыт
   delete []logName; //Сразу же очищаем лишнюю память
}

//Просто отправляет в лог сообщение (char*)
void PutInLog(char* info)
```

```
{
   fprintf(pLog,"%s",info);
   fflush(pLog);
}

//Закрытие файла
void EndLogger()
{
   fclose(pLog);
}
```

Main.cpp:

```
#include <cstdio>
#include <windows.h>
#include <cctype>
#include "tree.h"    //Реализация дерева через список
//#include "treeR3.h"  //Реализация дерева через файл
#include "Func.h"



using function = void(*)(NodePtr,FILE*,int); //псевдоним для функций вывода дерева в поток
//Параметры - входной файл для безключевого дерева
//файл входных данных, Папка Логов
int main(int argc, char** argv) {

   system("chcp 65001");
   Init();
   char* filename = new char[255]; //Имя входного файла
   char* buffer = new char[10]; //Буффер для считывания комманд при вывода дерева
   //Начало обработки аномалий
   try {
      InitLogger(argv[2]); //Подключаем лог-файл
      PutInLog((char *) "Инициализированы модули\n");
      //Создание деревьев
      MakeTree(argv[1]);
```

```
    MakeTreeKey();
    //Получение "указателей" на "корень"
    NodePtr headUnsorted = getStartUnsorted();
    NodePtr headSorted = getStartSorted();
    //Меню
    int end = 0;
    while (!end) {
      char act;
      printf("\'v\' - вывести деревья на экран\n"
           "\'p\' - вывести план поминок\n"
           "\'a\' - добавить к дереву по ключу\n"
           "\'t\' - выполнить поиск человека, у которого обоих дедушек зовут Иванами\n"
           "\'r\' - пересоздать деревья\n"
           "\'x\' - завершение программы\n");
      scanf("%c", &act);
      fflush(stdin);
      act = (char) tolower(act);
      //Обработка  комманд
      switch (act) {
        default: //Если пользователь ввел что-то невнятное
          printf("Неизвестная команда\n");
          break;
        case 'v': //Вывод деревьев на экран
          PutInLog((char *) "Вывод деревьев на экран ");
          int type;
          function fShowTree; //переменная, в которую определяется способ вывода дерева
          printf("Выберите способ: \'1\' - Корень-Лево-Право \'2\' - Лево-Корень-Право  \'3\' - Лево-
Право-Корень\n");
          scanf("%s", buffer); type = strtol(buffer, nullptr, 10);
          fflush(stdin);
          //Выбираем на основе введенных данных
          switch (type) {
            default: //Если пользователь ввел что-то невнятное
              printf("Неверная команда\n");
```

```
            fShowTree = nullptr;
            PutInLog((char *) "\nНе удалось т.к. пользователь ввел неверную команду\n");
            break;
        case (1): //Корень-Лево-Право
            PutInLog((char *) "типом Корень-Лево-Право\n");
            fShowTree = &ShowTree1;
            break;
        case (2): //Лево-Корень-Право
            PutInLog((char *) "типом Лево-Корень-Право\n");
            fShowTree = &ShowTree2;
            break;
        case (3): //Лево-Право-Корень
            PutInLog((char *) "типом Лево-Право-Корень\n");
            fShowTree = &ShowTree3;
            break;
    }
    if (fShowTree!=nullptr) { //Если успешно выбран способ
        printf("Генеалогическое дерево\n");
        //Выводим в консоль деревья
        fShowTree(headUnsorted, stdout, 0);
        printf("----------------\nУпорядоченное дерево\n");
        fShowTree(headSorted, stdout, 0);
        printf("\n");
    }
    break;
case 't': //Поиск человека, у которого обоих дедушек зовут Иванами
    PutInLog((char *) "Производится поиск элемнта по ключу\n");
    if (Task(headUnsorted))
        printf("Найден человек, у которого обоих дедушек зовут Иванами\n");
    else printf("Нет такого человека, у которого обоих дедушек зовут Иванами\n");
    break;
case 'r': //Пересоздание деревьев
    PutInLog((char *) "Вызвано пересоздание деревьев\n");
    //уничтожаем текущие деревья
```

```
    remove(&headUnsorted);
    remove(&headSorted);
    //Переинициализируем
    Init();
    //Считываем имя файла
    printf("Введите полное имя файла, с которого нужно получить входные файлы\n"
        "Если из того же \"--\"\n");
    scanf("%s",filename);
    //Если введено --  - вызываем из того же
    if (strcmp(filename,"--")==0) {
      filename = argv[1];
      PutInLog((char *) "Создание из того же входного файла\n");
    }
    else {
      PutInLog((char *) "Создание из ");
      PutInLog(filename);
      PutInLog((char*)"\n");
    }
    //Создаем снова деревья
    MakeTree(filename);
    MakeTreeKey();
    //Получаеи указатели на корни
    headUnsorted = getStartUnsorted();
    headSorted = getStartSorted();
    break;
  case 'x': //Завершение программы
    PutInLog((char *) "Вызвано завершение программы\n");
    end = 1;
    break;
  case 'p': //Вывод плана поминок
    PutInLog((char *) "Вывод плана поминок\n");
    ShowTree(headSorted);
    printf("\n");
    break;
```

```
            case 'a': //Добавление элемента в интеракитвном режиме
              PutInLog((char *) "Вызвано добавление элемента вручную\n");
              printf("Введите ФИО\n");
              //Считываем отдельно имя
              char *FName = new char[10];
              scanf("%s", FName);
              //Отправляем в функцию, продолжающую считывать данные, и получеенные данные в функцию добвления элемента
              AddToTreeKey(readData(FName));
              PutInLog((char *) "Отсортированное дерево:\n");
              ShowTree1(headSorted, pLog);
              break;
          }
        }
        // под конец уничтожаем деревья и закрываем в лог
        remove(&headUnsorted);
        remove(&headSorted);
        PutInLog((char *) "Завершение программы\n");
        EndLogger();
      }
      //Если во время выполнения программы пошло что-то не так
      catch(int errorCode)
      { //На основе выданного кода ошибки сообщаем какая именно она была
        switch(errorCode){
          default: //На случай, если выбросится что-то непонятное
            printf("Неизвестный код ошибки\n");
            break;
          case (0):
            printf("Невозможно создать Лог-файл\n");
            break;
          case (1):
            printf("Невозможно прочитать входной файл\n");
            PutInLog((char*)"Завершение программы из-за невозможности прочитать входной файл\n");
            EndLogger();
```

```
            break;
        case (2):
            printf("Невозможно создать бинарные файлы\n");
            PutInLog((char*)"Завершение программы из-за невозможнсти создать бинарные файлы\n");
            EndLogger();
            break;
        case (3):
            printf("Неверный формат в поле даты\n");
            PutInLog((char*)"Завершение программы из-за неверного формата даты\n");
            EndLogger();
            break;
        case (4):
            printf("Попытка записи в существующий блок\n");
            PutInLog((char*)"Завершение программы из-за неверных входных данных\n");
            EndLogger();
            break;
      }
   }
   //Возвращаем память
   delete []filename;
   delete []buffer;
   printf("PRESS ENTER");
   getchar();
   return 0;
}
```

Func.h:

```
#include "Logger.h"

/* Функция вывода полной информации.
 * Может выводить в консоль по умолчанию или в в другой поток
 * В частности - файл лога
 * Чтобы не писать нескольно раз, вынесено в отдельную функцию
 * входные параметры: данные, поток для вывода
*/
```

```
void ShowData(data info,FILE *pfile=stdout)
{
  //Основные данные
  fprintf(pfile, "%s %s %s Родился %d.%d.%d ", info.SName, info.FName, info.LName, info.dBirth.day,
info.dBirth.month, info.dBirth.year);
  //Если есть дата смерти - выводим и ее
  if (info.dDeath.day) fprintf(pfile, "Умер %d.%d.%d ", info.dDeath.day, info.dDeath.month, info.dDeath.year);
  //Так же и место рождения
  if (strcmp(info.BPlace,"--")!=0) fprintf(pfile, "Родился в %s", info.BPlace);
  fprintf(pfile, "\n");
  fflush(pfile);
}

/* Первый способ выведения дерева Корень-лево-право
 * Может выводить по умолчанию в консоль, или в другой поток
 * В частности - лог
 * входные параметры: данные, поток для вывода, глубина текущаего элемента в дереве
 */
void ShowTree1(NodePtr TreeHead, FILE *pfile= stdout, int depth= 0){
  //Проверка на стоп-знак, если вызвана вне рекурсии
  if (strcmp(getData(TreeHead).FName,"**")==0 or IsEmpty(TreeHead)) fprintf(pfile,"**\n");
  else {
    fprintf(pfile, "%*s", depth * 2 + 3, "-- ");
    fflush(pfile);
    //выводим данные
    ShowData(getData(TreeHead), pfile);
    //проверяем на пустоту лево, право и выводим данные оттуда, если есть
    if (!IsEmpty(curL(TreeHead)) and strcmp(getData(curL(TreeHead)).FName, "**") != 0)
      ShowTree1(curL(TreeHead), pfile, depth + 1);
    if (!IsEmpty(curR(TreeHead)) and strcmp(getData(curR(TreeHead)).FName, "**") != 0)
      ShowTree1(curR(TreeHead), pfile, depth + 1);
  }
}
```

```
/* Второй способ выведения дерева лево-корень-право
 * Может выводить по умолчанию в консоль, или в другой поток
 * В частности - лог
 * входные параметры: данные, поток для вывода, глубина текущаего элемента в дереве
 */
void ShowTree2(NodePtr TreeHead,FILE *pfile=stdout, int depth=0) {
   //Проверка на стоп-знак, если вызвана вне рекурсии
   if (IsEmpty(TreeHead) or strcmp(getData(TreeHead).FName,"**")==0) fprintf(pfile,"**\n");
   else {
      //проверяем на пустоту лево и выводим данные оттуда, если есть
      if (!IsEmpty(curL(TreeHead)) and strcmp(getData(curL(TreeHead)).FName,"**")!=0)
ShowTree2(curL(TreeHead), pfile,depth+1);
      fprintf(pfile,"%*s",depth*2+3,"-- "); fflush(pfile);
      //Нынещние данные
      ShowData(getData(TreeHead),pfile);
      //проверяем на пустоту  право и выводим данные оттуда, если есть
      if (!IsEmpty(curR(TreeHead)) and
strcmp(getData(curR(TreeHead)).FName,"**")!=0)ShowTree2(curR(TreeHead), pfile,depth+1);
   }
}

/* Третий способ выведения дерева лево-право-корень
 * Может выводить по умолчанию в консоль, или в другой поток
 * В частности - лог
 * входные параметры: данные, поток для вывода, глубина текущаего элемента в дереве
 */
void ShowTree3(NodePtr TreeHead,FILE *pfile=stdout, int depth=0) {
   //Проверка на стоп-знак, если вызвана вне рекурсии
   if (IsEmpty(TreeHead) or strcmp(getData(TreeHead).FName,"**")==0) fprintf(pfile,"**\n");
   else {
      //проверяем на пустоту лево, право и выводим данные оттуда, если есть
      if (!IsEmpty(curL(TreeHead)) and strcmp(getData(curL(TreeHead)).FName,"**")!=0)
ShowTree2(curL(TreeHead), pfile,depth+1);
      if (!IsEmpty(curR(TreeHead)) and strcmp(getData(curR(TreeHead)).FName,"**")!=0)
ShowTree2(curR(TreeHead), pfile,depth+1);
```

```
      fprintf(pfile,"%*s",depth*2+3,"-- "); fflush(pfile);
      //Текущие данные
      ShowData(getData(TreeHead),pfile);
    }
}



/*
 * Способ вывода дерева только для сортированного дерева
 * Или же вывод плана поминок
 * Входные параметры: указатель на текущее положение, поток вывода
 */
void ShowTree(NodePtr TreeHead, FILE *pfile= stdout) {
   //Проверка на стоп-знак, если вызвана вне рекурсии
   if (IsEmpty(TreeHead) or strcmp(getData(TreeHead).FName,"**")==0) fprintf(pfile,"**\n");
   else {
      //По сортировке, что меньше - лево, поэтому идем Лево-Корень-Право
      if (!IsEmpty(curL(TreeHead)) and strcmp(getData(curL(TreeHead)).FName,"**")!=0)
         ShowTree(curL(TreeHead), pfile);
      data info = getData(TreeHead);
      fprintf(pfile, "%10s %10s %10s | Умер %d.%d.%d\n", info.SName, info.FName, info.LName,
info.dDeath.day, info.dDeath.month, info.dDeath.year);
      fflush(pfile);
      if (!IsEmpty(curR(TreeHead)) and strcmp(getData(curR(TreeHead)).FName,"**")!=0)
         ShowTree(curR(TreeHead), pfile);
   }
}
/*
 * Добавление элемента в отсортированное дерево
 * Входной параметр - данные для записи
 */
void AddToTreeKey(data currentData)
{
   PutInLog((char*)"Элемент: ");
```

```
  ShowData(currentData,pLog);
  NodePtr CurrentPosition=getStartSorted(); //Начинаем с корня
  while (!IsEmpty(CurrentPosition) and strcmp(getData(CurrentPosition).FName,"**")!=0) //Пока не пустой элемент
  {
    PutInLog((char*)"Сравнение текущего элемента с ");
    data prevData = getData(CurrentPosition);
    ShowData(prevData,pLog);
    //Сравнение текущего элемента с новым и определение его влево или вправо
    if ((prevData.dDeath.day-currentData.dDeath.day)+
      (prevData.dDeath.month-currentData.dDeath.month)*31>0)
    {
      CurrentPosition = curL(CurrentPosition);
      PutInLog((char*)"Переход влево\n");
    }
    else {
      CurrentPosition = curR(CurrentPosition);
      PutInLog((char*)"Переход вправо\n");
    }
  }
  //Запись
  replDataM(&CurrentPosition,currentData);
  PutInLog((char*)"Элемент записан\n");
  //Предопределение, что следующие пустые, потом по необходимости заменяются
  replDataR(&CurrentPosition,nullData);
  replDataL(&CurrentPosition,nullData);
}

// Рекурсия для построения отсортированного дерева
void TreeSortedBranches(NodePtr CurrentPositionOfUnsorted)
{
  data currentData = getData(CurrentPositionOfUnsorted);
  if (strcmp(currentData.FName,"**")!=0 and !IsEmpty(CurrentPositionOfUnsorted) and currentData.dDeath.day) { //Проверка на пустоту
    AddToTreeKey(currentData);
```

```
    }
    if (!IsEmpty(curL(CurrentPositionOfUnsorted)) and
        strcmp(getData(curL(CurrentPositionOfUnsorted)).FName, "**") != 0)
        //Если слева не пусто - вызываем для левой части
        TreeSortedBranches(curL(CurrentPositionOfUnsorted));
    if (!IsEmpty(curR(CurrentPositionOfUnsorted)) and
        strcmp(getData(curR(CurrentPositionOfUnsorted)).FName, "**") != 0)
        //Если справа не пусто - вызываем для правой части
        TreeSortedBranches(curR(CurrentPositionOfUnsorted));
}


//Инициализатор создания отсортированного дерева
void MakeTreeKey()
{
    PutInLog((char*)"Построение отсортированного дерева\n");
    TreeSortedBranches(getStartUnsorted()); //Создание ветвей
    PutInLog((char*)"Создано дерево:\n");
    ShowTree1(getStartSorted(), pLog);//вывод в лог
    PutInLog((char*)"План поминок выглядит:\n");
    ShowTree(getStartSorted(), pLog);
}

//Поиск человека, у которого обоих дедушек зовут Иванами
int Task(NodePtr CurrentPosition)
{
    //Проверка на пустоту текущей позиции
    if (IsEmpty(CurrentPosition) or strcmp(getData(CurrentPosition).FName,"**")==0)
    {
        printf("Невозможно выполнить поиск на пустом дереве\n");
        PutInLog((char*)"Функция вызвана на путом дереве\n");
    } else {
        PutInLog((char *) "Проверка элемента ");
        ShowData(getData(CurrentPosition), pLog);
```

```
    //Проверка на наличие родителей и дедушек
    if ( !IsEmpty(curL(CurrentPosition)) and !IsEmpty(curR(CurrentPosition)) and
      strcmp(getData(curL(CurrentPosition)).FName, "**") != 0 and
      strcmp(getData(curR(CurrentPosition)).FName, "**") != 0 and
      !IsEmpty(curL(curL(CurrentPosition))) and !IsEmpty(curR(curL(CurrentPosition)))
        )
    {
      PutInLog((char *) "У элемента найдены дедушки:\n");
      //Считываение дедушек
      data grandL = getData(curL(curL(CurrentPosition)));
      data grandR = getData(curL(curR(CurrentPosition)));
      ShowData(grandL,pLog);
      ShowData(grandR,pLog);
      //Проверка имени
      if (strcmp(grandL.FName, "Иван") == 0 and strcmp(grandR.FName, "Иван") == 0)
      {
        PutInLog((char *) "Элемент соответсвует поиску\n");
        return 1;
      }
      PutInLog((char *) "Не соответствует поиску\n");
    }
    else {
      PutInLog((char *) "У элемента нет дедушек\n");
    }
    //Если элемент не прошел проверку, пускаем по следующим элементам
    return ((
          (!IsEmpty(curL(CurrentPosition))) and (Task(curL(CurrentPosition))))
      or ((!IsEmpty(curR(CurrentPosition))) and (Task(curR(CurrentPosition))))
    );
  }
  return 0;
}

/*
```

```
 * Чтение данных их потока(второй параметр).
 * Если считывается из стандартного текстового файла(консоль) -
 * выводятся подсказки для пользователя
 * Чтобы не писать нескольно раз, вынесено в отдельную функцию
 */
data readData(char* FName, FILE* FileIn=stdin)
{
   //FName находится в параметрах т.к. при считывании из файла проверяется на стоп-знак или eof
   char *SName = new char[10];
   char *LName = new char[10];
   char *BPlace = new char[50];

   char* buffer = new char[10];
   fscanf(FileIn, "%s", SName);
   fscanf(FileIn, "%s", LName);

   int day, month, year;
   if (FileIn==stdin) printf("Введите дату рождения\n");
   //Чтобы предотвратить вылет из-за неправильных данных, записываем сначала в буффер,
   //потом пытаемся конвертировать в число
   fscanf(FileIn, "%s", buffer); day   = strtol(buffer,nullptr,10);
   fscanf(FileIn, "%s", buffer); month = strtol(buffer,nullptr,10);
   fscanf(FileIn, "%s", buffer); year  = strtol(buffer,nullptr,10);
   date dBirth = {day, month, year};
   //Поверка на правильность даты, иначе - выдаем код ошибки
   if (day<1 or day>31 or month<1 or month>12 or year<1) throw 3;

   if (FileIn==stdin) printf("Введите дату смерти.\n");
   //Чтобы предотвратить вылет из-за неправильных данных, записываем сначала в буффер,
   //потом пытаемся конвертировать в число (в данном случае ошибка в формате даты распознается как отсутствие даты)
   fscanf(FileIn, "%s", buffer); day   = strtol(buffer,nullptr,10);
   fscanf(FileIn, "%s", buffer); month = strtol(buffer,nullptr,10);
   fscanf(FileIn, "%s", buffer); year  = strtol(buffer,nullptr,10);
```

```
  if (month==0 or year==0) day=0; //т.к. вся проверка даты смерти ведется через день, то если хоть где-то есть 0, то в дне тоже обязательно должен быть 0
  date dDeath = {day, month, year};
  //Поверка на правильность даты, иначе - выдаем код ошибки. Расширен до 0, т.к. может не быть даты смерти
  if (day<0 or day>31 or month<0 or month>12 or year<0) throw 3;

  if (FileIn==stdin) printf("Введите место рождения или \"--\"\n");
  fscanf(FileIn, "%s", BPlace);
  delete []buffer;
  return {FName,SName,LName,dBirth,dDeath,BPlace};
}

//Формирование ветвей генеалогического древа.
void TreeBranches(FILE *FileIn,NodePtr* CurrentPosition)
{
  //Считываем первый символ
  char *buffer = new char[50];
  fscanf(FileIn,"%s",buffer);
  data newinfo{};
  //Если не конец файла - смотрим является ли стоп-знаком
  if (!feof(FileIn)) {
    NodePtr next;
    if (strcmp(buffer,"**")!=0) {
      PutInLog((char*)"Найден новый элемент\n");
      //Считываем из входного файла
      newinfo = readData(buffer,FileIn);
      PutInLog((char*)"Прочитан новый элемент\n");
      //Дальше - лево, создаем пустой элемент
      next = NewNode(CurrentPosition,'L');
      replDataM(CurrentPosition,newinfo);
      replDataR(CurrentPosition,nullData); //Так как бинарный файл не забивает пустые позиции nullptr-м, предопределяем их нулевыми данными
      PutInLog((char*)"Создана ветвь "); ShowData(getData(*CurrentPosition),pLog);
      PutInLog((char*)"Переход влево\n");
```

```
    }
    else
    {
      PutInLog((char*)"Прочитан стоп-знак\n");
      newinfo = {(char*)"**"};
      replDataM(CurrentPosition,newinfo);
      //Если прочитали стоп-знак, возвращаемся назад, пока не найдем справа пустое место
      do
      {
        *CurrentPosition = back(CurrentPosition);
        PutInLog((char*)"Возврат на предыдущую позицию\n");
      }while(!IsEmpty(curR(*CurrentPosition)) and (*CurrentPosition)!=getStartUnsorted());
      if (!IsEmpty(curR(*CurrentPosition))) throw 4;
      //Нашли - идем вправо
      next = NewNode(CurrentPosition,'R');
      PutInLog((char*)"Переход вправо\n");
    }
    TreeBranches(FileIn, &next);
  }
  else{ //Так как последний ** считывается, но не обрабатывается - делаем сами
    PutInLog((char*)"Конец файла\n");
    newinfo = {(char*)"**"};
    replDataM(CurrentPosition,newinfo);
  }
}
//Инициализатор создания неупорядоченного дерева
void MakeTree(char* filename)
{
  PutInLog((char*)"Построение обыкновенного дерева\n");
  FILE* in = fopen(filename,"r");
  if (in==nullptr) throw 1; //Если файл не удалось открыть - сообщаем
  NodePtr headUnsorted = getStartUnsorted();
  TreeBranches(in,&headUnsorted); //Запускаем создание дерева
  fclose(in);
```

```
   PutInLog((char *)"Создано дерево:\n");
   ShowTree1(headUnsorted, pLog);

}
//Уничтожение ветви или целого дерева
void remove(NodePtr *cur)
{
   //Проверяем какое дерево уничтожаем, соответствующе отчитываемся
   if ((*cur)==getStartUnsorted()) {
      PutInLog((char*)"Уничтожено обыкновенное дерево\n");
      destroy(cur);
   }
   else if ((*cur)==getStartSorted()) {
      PutInLog((char*)"Уничтожено отсортированное дерево\n");
      destroy(cur);
   }
   //Есть возможность "обрубания" одной из ветвей, но не используется
   else{
      PutInLog((char*)"Уничтожена ветвь "); ShowData(getData(*cur),pLog);
      destroy(cur);
   }
}
```

Приложение 4:

/* Лог файл полной проработки программы с входными данными теста 21**/**

Инициализированы модули

Построение обыкновенного дерева

Найден новый элемент

Прочитан новый элемент

Создана ветвь Тестовый Человек Корневой Родился 11.11.2000

Переход влево

Найден новый элемент

Прочитан новый элемент

Создана ветвь Тестовый Человек Слева Родился 16.11.1980

Переход влево

Найден новый элемент

Прочитан новый элемент

Создана ветвь Тестовый Иван СлеваСлева Родился 12.5.1950

Переход влево

Найден новый элемент

Прочитан новый элемент

Создана ветвь Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000

Переход влево

Прочитан стоп-знак

Возврат на предыдущую позицию

Переход вправо

Прочитан стоп-знак

Возврат на предыдущую позицию

Возврат на предыдущую позицию

Переход вправо

Прочитан стоп-знак

Возврат на предыдущую позицию

Возврат на предыдущую позицию

Переход вправо

Найден новый элемент

Прочитан новый элемент

Создана ветвь Тестовый Олег СправаСлева Родился 12.5.1950

Переход влево

Найден новый элемент

Прочитан новый элемент

Создана ветвь Тестовый Иван ЛПЛ Родился 12.5.1920 Умер 14.1.2000

Переход влево

Прочитан стоп-знак

Возврат на предыдущую позицию

Переход вправо

Прочитан стоп-знак

Возврат на предыдущую позицию

Возврат на предыдущую позицию

Переход вправо

Прочитан стоп-знак

Возврат на предыдущую позицию

Возврат на предыдущую позицию

Возврат на предыдущую позицию

Переход вправо

Найден новый элемент

Прочитан новый элемент

Создана ветвь Тестовый Человек Справа Родился 16.11.1980

Переход влево

Найден новый элемент

Прочитан новый элемент

Создана ветвь Тестовый Иван СлеваСправа Родился 12.5.1950

Переход влево

Прочитан стоп-знак

Возврат на предыдущую позицию

Переход вправо

Прочитан стоп-знак

Возврат на предыдущую позицию

Возврат на предыдущую позицию

Переход вправо

Конец файла

Создано дерево:

-- Тестовый Человек Корневой Родился 11.11.2000

 -- Тестовый Человек Слева Родился 16.11.1980

  -- Тестовый Иван СлеваСлева Родился 12.5.1950

   -- Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000

  -- Тестовый Олег СправаСлева Родился 12.5.1950

   -- Тестовый Иван ЛПЛ Родился 12.5.1920 Умер 14.1.2000

 -- Тестовый Человек Справа Родился 16.11.1980

  -- Тестовый Иван СлеваСправа Родился 12.5.1950

Построение отсортированного дерева

Элемент: Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000

Элемент записан

Элемент: Тестовый Иван ЛПЛ Родился 12.5.1920 Умер 14.1.2000

Сравнение текущего элемента с Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000

Переход вправо

Элемент записан

Создано дерево:

-- Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000

 -- Тестовый Иван ЛПЛ Родился 12.5.1920 Умер 14.1.2000

План поминок выглядит:

    Тестовый Иван ЛЛЛ | Умер 14.1.2000

    Тестовый Иван ЛПЛ | Умер 14.1.2000

Вызвано добавление элемента вручную

Элемент: Тестовый Человек Доп Родился 11.11.2020 Умер 11.11.2222

Сравнение текущего элемента с Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000

Переход вправо

Сравнение текущего элемента с Тестовый Иван ЛПЛ Родился 12.5.1920 Умер 14.1.2000

Переход вправо

Элемент записан

Отсортированное дерево:

-- Тестовый Иван ЛЛЛ Родился 12.5.1920 Умер 14.1.2000

 -- Тестовый Иван ЛПЛ Родился 12.5.1920 Умер 14.1.2000

  -- Тестовый Человек Доп Родился 11.11.2020 Умер 11.11.2222

Вывод деревьев на экран типом Лево-Корень-Право

Вывод плана поминок

Производится поиск элемента по ключу

Проверка элемента Тестовый Человек Корневой Родился 11.11.2000

У элемента найдены дедушки:

Тестовый Иван СлеваСлева Родился 12.5.1950

Тестовый Иван СлеваСправа Родился 12.5.1950

Элемент соответствует поиску

Вызвано пересоздание деревьев

Уничтожено обыкновенное дерево

Уничтожено отсортированное дерево

Создание из D:\Documents\GitHub\Labs_Progs\Sem2\TC\Tests\in1.txt

Построение обыкновенного дерева

Найден новый элемент

Прочитан новый элемент

Создана ветвь Molekula Chelovek Odin Родился 11.11.2000 Умер 1.2.2060 Родился в Doma

Переход влево

Найден новый элемент

Прочитан новый элемент

Создана ветвь Molekula Chelovek DwaL Родился 16.11.2010 Родился в Uliza

Переход влево

Прочитан стоп-знак

Возврат на предыдущую позицию

Переход вправо

Прочитан стоп-знак

Возврат на предыдущую позицию

Возврат на предыдущую позицию

Переход вправо

Найден новый элемент

Прочитан новый элемент

Создана ветвь Molekula Chelovek DwaR Родился 16.10.1920 Умер 1.2.2000

Переход влево

Найден новый элемент

Прочитан новый элемент

Создана ветвь Molekula Chelovek TriL Родился 16.10.1910 Умер 2.1.1990

Переход влево

Прочитан стоп-знак

Возврат на предыдущую позицию

Переход вправо

Прочитан стоп-знак

Возврат на предыдущую позицию

Возврат на предыдущую позицию

Переход вправо

Найден новый элемент

Прочитан новый элемент

Создана ветвь Molekula Chelovek TriR Родился 16.10.1911 Умер 2.1.1991

Переход влево

Прочитан стоп-знак

Возврат на предыдущую позицию

Переход вправо

Конец файла

Создано дерево:

-- Molekula Chelovek Odin Родился 11.11.2000 Умер 1.2.2060 Родился в Doma

-- Molekula Chelovek DwaL Родился 16.11.2010 Родился в Uliza

-- Molekula Chelovek DwaR Родился 16.10.1920 Умер 1.2.2000

-- Molekula Chelovek TriL Родился 16.10.1910 Умер 2.1.1990

-- Molekula Chelovek TriR Родился 16.10.1911 Умер 2.1.1991

Построение отсортированного дерева

Элемент: Molekula Chelovek Odin Родился 11.11.2000 Умер 1.2.2060 Родился в Doma

Элемент записан

Элемент: Molekula Chelovek DwaR Родился 16.10.1920 Умер 1.2.2000

Сравнение текущего элемента с Molekula Chelovek Odin Родился 11.11.2000 Умер 1.2.2060 Родился в Doma

Переход вправо

Элемент записан

Элемент: Molekula Chelovek TriL Родился 16.10.1910 Умер 2.1.1990

Сравнение текущего элемента с Molekula Chelovek Odin Родился 11.11.2000 Умер 1.2.2060 Родился в Doma

Переход влево

Элемент записан

Элемент: Molekula Chelovek TriR Родился 16.10.1911 Умер 2.1.1991

Сравнение текущего элемента с Molekula Chelovek Odin Родился 11.11.2000 Умер 1.2.2060 Родился в Doma

Переход влево

Сравнение текущего элемента с Molekula Chelovek TriL Родился 16.10.1910 Умер 2.1.1990

Переход вправо

Элемент записан

Создано дерево:

-- Molekula Chelovek Odin Родился 11.11.2000 Умер 1.2.2060 Родился в Doma

-- Molekula Chelovek TriL Родился 16.10.1910 Умер 2.1.1990

-- Molekula Chelovek TriR Родился 16.10.1911 Умер 2.1.1991

-- Molekula Chelovek DwaR Родился 16.10.1920 Умер 1.2.2000

План поминок выглядит:

Molekula Chelovek TriL | Умер 2.1.1990

Molekula Chelovek TriR | Умер 2.1.1991

Molekula Chelovek Odin | Умер 1.2.2060

Molekula Chelovek DwaR | Умер 1.2.2000

Вывод деревьев на экран типом Лево-Право-Корень

Вызвано завершение программы

Уничтожено обыкновенное дерево

Уничтожено отсортированное дерево

Завершение программы