Содержание

Введение 2
1. Методы разработки программных комплексов. 3
1.1 Объектно-ориентированное программирование 3
1.2 Существующие объектно-ориентированные языки 6
2. Алгоритмический анализ задачи 13
2.1 Постановка задачи. 13
2.2 Бинарное дерево 13
2.3 Операции в бинарном дереве 16
3.Описание разработанного приложения 19
3.1 Диаграмма и описание классов 19
3.2 Возможности пользователя 22
3.3 Пример работы приложения 23
Заключение (отражает результаты проделанной работы) 28
Список использованных источников 29
Приложения – Код программы 30

Введение

Информационные технологии все больше и больше затрагивает сферы деятельности человека. И сейчас под натиском информационных и телекоммуникационным технологиям необходимо введение информационных систем в те области где они не применяются или слабо развиты и которые помогут уменьшить затраты, время на обработку данных, и увеличить производительность труда.

Цель данной работы заключается в разработке справочной информационной системы «Спортсмен» с использованием объектно-ориентированного подхода.

В качестве входных данных будет использоваться XML с данными спортсмена, записанный в виде бинарного дерева. Для качественной работы необходимо разобраться что собой представляют бинарные деревья и способы работы с ними.

Требования системы являются следующими, приложение должно поддерживать:

- Чтение данных из файла
- Сохранение данных в файл
- Добавление данных через интерфейс
- Изменение данных через интерфейс
- Удаление данных через интерфейс
- Вывод информации о местах, которые не удалось прочитать

Разрабатываемое программное обеспечение должно соответствовать всем требованиям, а цель курсового проекта должна быть полностью достигнута.

1. Методы разработки программных комплексов.

1.1 Объектно-ориентированное программирование

ООП — это такой подход к программированию, где на первом месте стоят объекты. На самом деле там всё немного сложнее, но мы до этого ещё доберёмся. Для начала поговорим про ООП вообще и разберём, с чего оно начинается.

Чаще всего под обычным понимают процедурное программирование, в основе которого — процедуры и функции. Функция — это мини-программа, которая получает на вход какие-то данные, что-то делает внутри себя и может отдавать какие-то данные в результате вычислений. Представьте, что это такой конвейер, который упакован в коробочку.

Например, в интернет-магазине может быть функция «Проверить email». Она получает на вход какой-то текст, сопоставляет со своими правилами и выдаёт ответ: это правильный электронный адрес или нет. Если правильный, то true, если нет — то false.

Функции полезны, когда нужно упаковать много команд в одну. Например, проверка электронного адреса может состоять из одной проверки на регулярные выражения, а может содержать множество команд: запросы в словари, проверку по базам спамеров и даже сопоставление с уже известными электронными адресами. В функцию можно упаковать любой комбайн из действий и потом просто вызывать их все одним движением.

Процедурное программирование идеально работает в простых программах, где все задачи можно решить, грубо говоря, десятком функций. Функции аккуратно вложены друг в друга, взаимодействуют друг с другом, можно передать данные из одной функции в другую.

Например, вы пишете функцию «Зарегистрировать пользователя интернет-магазина». Внутри неё вам нужно проверить его электронный адрес. Вы вызываете функцию «Проверить email» внутри функции «Зарегистрировать пользователя», и в зависимости от ответа функции вы либо регистрируете пользователя, либо выводите ошибку. И у вас эта функция встречается ещё в десяти местах. Функции как бы переплетены.

Затем подошел менеджер по продукту и сказал:"Я хочу, чтобы пользователь точно знал, в чем заключается ошибка при вводе адреса электронной почты."Теперь нужно научить функцию выводить не только true и false,но и код ошибки: например, если адрес имеет опечатку, то код 01, Если адрес спам-код 02 и так далее. Это легко осуществить.

Вы заходите в эту функцию и меняете ее поведение: теперь она возвращает код ошибки вместо true-false, а если ошибки нет, то пишет"OK".

Затем ваш код ломается: все десять мест, которые ожидают true или false от валидатора, теперь получают"OK"и в результате ломаются.

Теперь вам нужно:

- Оба переписывают все функции и учат их понимать новую реакцию адресной проверки;

- Или переделать саму проверку адресов так,чтобы она все еще была совместима со старым местом, но каким-то образом все еще выдавала код ошибки там, где вам это нужно;

- Или написать новый инспектор, вывести код ошибки и использовать старый инспектор на старом месте.

Конечно, эту проблему можно решить за час-два. Но теперь представьте, что у вас есть сотни таких функций. Вам нужно вносить в них десятки изменений в день. Каждое изменение обычно приводит к тому, что функция работает более сложным образом и дает более сложные результаты. И каждое изменение в одном месте нарушает три других места. В результате у вас будут десятки клонированных функций, которые вы сначала поймете, а потом нет.

Это называется спагетти-код,и для его обработки было изобретено объектно-ориентированное программирование.

Основная цель ООП - сделать сложный код проще. Для этого программа разбивается на отдельные блоки, которые мы называем объектами.

Объект - это не космическая сущность. Это просто набор данных и функций-то же самое, что и традиционное функциональное

программирование. Как вы можете себе представить, вы просто кладете кусочек программы и кладете его в коробку, закрываете крышкой. Вот крышка-это предмет.

Программисты соглашаются, что данные внутри объекта будут называться свойствами, а функции-методами. Но это всего лишь литералы,на самом деле это те же самые переменные и функции.

Вы можете думать об объекте как об отдельном приборе на вашей кухне. Чайник кипятит воду,плита греет,блендер взбивает,мясорубка делает фарш. Внутри каждого устройства есть много вещей: двигатели, контроллеры, кнопки, пружины, предохранители-но вы не будете думать о них. Вы нажимаете кнопку на панели каждого устройства, и оно делает то, что ожидается. И благодаря совместной работе этих устройств вы получаете обед.

Объекты характеризуются четырьмя словами: инкапсуляция, абстракция, наследование и полиморфизм. Если вам интересно, что это такое, мы приглашаем вас к кошке:

Инкапсуляция,абстракция,наследование,полиморфизм

Такой подход позволяет программировать каждый модуль независимо от других модулей. Главное-заранее продумать, как модули будут взаимодействовать друг с другом и по каким правилам. При таком подходе можно повысить производительность одного модуля, не затрагивая другие модули-пока правила его использования остаются неизменными, содержимое каждого блока не имеет значения для всей программы.

Объектно-ориентированное программирование имеет много преимуществ,именно поэтому большинство современных программистов используют именно этот подход.

− Визуально код становится проще, и его легче читать. Когда всё разбито на объекты и у них есть понятный набор правил, можно сразу понять, за что отвечает каждый объект и из чего он состоит.

− Меньше одинакового кода. Если в обычном программировании одна функция считает повторяющиеся символы в одномерном массиве, а другая — в двумерном, то у них большая часть кода будет одинаковой. В ООП это решается наследованием.

− Сложные программы пишутся проще. Каждую большую программу можно разложить на несколько блоков, сделать им минимальное наполнение, а потом раз за разом подробно наполнить каждый блок.

− Увеличивается скорость написания. На старте можно быстро создать нужные компоненты внутри программы, чтобы получить минимально работающий прототип.

А теперь про минусы:

− Сложно понять и начать работать. Подход ООП намного сложнее обычного процедурного программирования — нужно знать много теории, прежде чем будет написана хоть одна строчка кода.

− Требует больше памяти. Объекты в ООП состоят из данных, интерфейсов, методов и много другого, а это занимает намного больше памяти, чем простая переменная.

− Иногда производительность кода будет ниже. Из-за особенностей подхода часть вещей может быть реализована сложнее, чем могла бы быть. Поэтому бывает такое, что ООП-программа работает медленнее, чем процедурная (хотя с современными мощностями процессоров это мало кого волнует).

1.2 Существующие объектно-ориентированные языки

Smalltalk

Язык Smalltalk был разработан исследовательской группой Xerox Palo Alto Research Center learning research group в рамках программного обеспечения Dynabook, фантастического проекта Алана Кея. Она основана на идее Симулы.. Smalltalk - это одновременно язык программирования и среда разработки программного обеспечения. Это чисто объектно-ориентированный язык, который абсолютно все рассматривает как объект;даже целые числа являются классами. После Simula Smalltalk является наиболее важным объектно-ориентированным языком, поскольку он не только повлиял на последующие поколения языков программирования,но и заложил основу для современных графических пользовательских интерфейсов, основанных непосредственно на интерфейсах Macintosh, Windows и Motif.

В основу языка положены две простые идеи:

- все является объектами;

- объекты взаимодействуют, обмениваясь сообщениями.

В таблице 1 приведены характеристики языка Smalltalk с точки зрения семи основных элементов объектного подхода.

Таблица1- Характеристика

Абстракции	Переменные экземпляра Методы экземпляра Переменные класса Методы класса	Да Да Да Да
Инкапсуляция	Переменных Методов	Закрытые Открытые
Модульность	Разновидности модулей	Нет
Иерархии	Наследование Шаблоны Метаклассы	Одиночное Нет Да
Типизация	Сильная типизация Полиморфизм	Нет Да (одиночный)
Параллельность	Многозадачность	Непрямая (посредством классов)
Сохраняемость	Долгоживущие объекты	Нет

Большим недостатком Smalltalk являются большие требования к памяти и низкая производительность полученных программ. Это связано с не очень удачной реализацией объектно-ориентированных особенностей.

C++ (си-плас-плас)

Язык программирования C++был разработан Бьорном страструпом, сотрудником AT & T Bell Labs. Прямым предшественником C++является C с классами, созданный в 1980 году тем же автором. В свою очередь, язык Си с классами находится под сильным влиянием языка Си и Simula. В некотором смысле мы можем назвать C++улучшенным C, который обеспечивает управление типами, перегрузку функций и многие другие средства. Но главное, что C++добавляет объектно-ориентированный язык в C.Характеристики C++ приведены в таблице 2.

Таблица2- Характеристика

Абстракции	Переменные экземпляра Методы экземпляра Переменные класса Методы класса	Да Да Да Да
Инкапсуляция	Переменных Методов	Открытые, защищенные, закрытые Открытые, защищенные, закрытые
Модульность	Разновидности модулей	Файл
Иерархии	Наследование Шаблоны Метаклассы	Множественное Да Нет
Типизация	Сильная типизация Полиморфизм	Да Да (одиночный)
Параллельность	Многозадачность	Непрямая (посредством классов)
Сохраняемость	Долгоживущие объекты	Нет

C# (си-шарп)

C# (произносится «си шарп») — объектно-ориентированный язык программирования. Разработан в 1998—2001 годах группой инженеров под руководством Андерса Хейлсберга в компании Microsoft как язык разработки приложений для платформы Microsoft .NET Framework и впоследствии был стандартизирован как ECMA-334 и ISO/IEC 23270.

Переняв многое от своих предшественников — языков C++, Pascal, Модула, Smalltalk и в особенности Java — C#, опираясь на практику их использования, исключает некоторые модели, зарекомендовавшие себя как проблематичные при разработке программных систем, например, C# в отличие от C++ не поддерживает множественное наследование классов (между тем допускается множественное наследование интерфейсов).

Common Lisp Object System (CLOS)

В начале 1980-х годов под влиянием идеи объектно-ориентированного программирования появился ряд новых диалектов Лиспа, многие из которых были сосредоточены на представлении знаний. Успех стандартизации Common Lisp привел к попытке стандартизации объектно-ориентированных диалектов в 1986 году.

Поскольку новый диалект должен быть дополнением к Common Lisp, он называется Common Lisp object system, называемой CLOS. Языки newflavors и CommonLoops оказали значительное влияние на проект CLOS. После двух лет усилий полная спецификация Clos была выпущена в 1988 году.

CLOS расширяется с помощью протокола meta-object, который реализует этот механизм без непосредственной поддержки механизма long-life object.

Ada

В 1983 году под эгидой Министерства Обороны США был создан язык Ada. Язык замечателен тем, что очень много ошибок может быть выявлено на этапе компиляции. Кроме того, поддерживаются многие аспекты программирования, которые часто отдаются на откуп операционной системе (параллелизм, обработка исключений). В 1995 году был принят стандарт языка Ada 95, который развивает предыдущую версию, добавляя в нее

объекно-ориентированность и исправляя некоторые неточности. Оба этих языка не получили широкого распространения вне военных и прочих крупномасштабных проектов (авиация, железнодорожные перевозки). Основной причиной является сложность освоения языка и достаточно громоздкий синтаксис.

Непосредственными предшественниками Ada являются Pascal и его производные, включая Euclid, Lis, Mesa, Modula и Sue. Были использованы некоторые концепции ALGOL-68, Simula, CLU и Alphard.

В таблице 3 приведены основные характеристики языка Ada с точки зрения объектного подхода.

Таблица 3- Характеристика

Абстракции	Переменные экземпляра Методы экземпляра Переменные класса Методы класса	Да Да Нет Нет
Инкапсуляция	Переменных Методов	Открытые, закрытые Открытые, закрытые
Модульность	Разновидности модулей	Пакет
Иерархии	Наследование Шаблоны Метаклассы	Нет (входит в Ada9x) Да Нет
Типизация	Сильная типизация Полиморфизм	Да Нет (входит в Ada9x)
Параллельность	Многозадачность	Да
Сохраняемость	Долгоживущие	Нет

	объекты	

Eiffel

Автор Eiffel Бертран Мейер (Bertrand Meyer) создавал не только язык объектно-ориентированного программирование, но и инструмент проектирования программ. Несмотря на сильное влияние Simula, Eiffel - вполне самостоятельный объектно-ориентированный язык со своей собственной средой разработки. Свойства языка показаны в таблице 4.

Таблица 4- Характеристика

Абстракции	Переменные экземпляра Методы экземпляра Переменные класса Методы класса	Да Да Нет Нет
Инкапсуляция	Переменных Методов	Закрытые Открытые, закрытые
Модульность	Разновидности модулей	Блок (unit)
Иерархии	Наследование Шаблоны Метаклассы	Множественное Да Нет
Типизация	Сильная типизация Полиморфизм	Да Да
Параллельность	Многозадачность	Нет
Сохраняемость	Долгоживущие объекты	Нет

Java

С 1995 года Java, новый объектно-ориентированный язык программирования, получил широкое распространение, главным образом в компьютерных сетях, особенно в интернете. Синтаксис этого языка похож на

C++,но эти два языка имеют мало общего. Java-это язык, который интерпретируется на заключительных этапах: он имеет внутреннее представление (предварительно скомпилированное в байт-код) и пост-интерпретатор для этого представления на целевой машине, который был реализован на большинстве платформ. Постинтерпретатор (виртуальная машина Java) упрощает отладку программ, написанных на Java, обеспечивая их переносимость на новые платформы и адаптацию к новым средам. Это позволяет исключить влияние программ, написанных на Java, на другие программы и файлы, доступные на новой платформе, обеспечивая тем самым безопасность операционной среды этих программ (без нарушения их производительности, что не будет прагматичным).

Object Pascal

Object Pascal был создан сотрудниками компании Apple computers (некоторые из которых являются участниками проекта Smalltalk) совместно с Никлаусом уортом, создателем языка Pascal. Object Pascal известен с 1986 года и является первым объектно-ориентированным языком программирования, включенным в macintosh programmer's workshop (MPW), среду разработки для компьютеров Apple Macintosh. В более ранних версиях этого языка не было методов классов, переменных классов, множественного наследования или метаклассов. Эти механизмы специально исключены, чтобы облегчить изучение языка начинающим "объектным" программистам.

Система визуального ООП Delphi

Появление Delphi среди многих пользователей компьютеров нельзя игнорировать. Эксперты, изучающие особенности этого нового продукта Borland, обычно полны энтузиазма. Главное преимущество Delphi заключается в том, что он реализует идею визуального программирования. Визуальная среда программирования превращает процесс создания программы в приятную и легкую для понимания структуру приложения, состоящую из большого количества графических и структурных примитивов.

2. Алгоритмический анализ задачи

2.1 Постановка задачи.

Задачей данного курсового проекта является разработка справочной информационной системы «Спортсмен».

Необходимо разработать представление данных пользователю. Исходные данные будут получены из файла формата XML. Сохраняться данные будут в том же формате. Для работы с данными необходимо разработать класс Спортсмен со следующими полями

− ФИО
− Возраст
− Количество побед
− Вид спорта.
− Сохранять

Структура файла будет в виде бинарного дерева. Для его обхода будет использоваться префиксный обход (в глубину сверху вниз).

2.2 Бинарное дерево

Дерево является одним из частных случаев графа. Древовидная модель оказывается довольно эффективной для представления динамических данных с целью быстрого поиска информации.

Дерево – это структура данных, представляющая собой совокупность элементов и отношений, образующих иерархическую структуру этих элементов (см. рисунок 1).

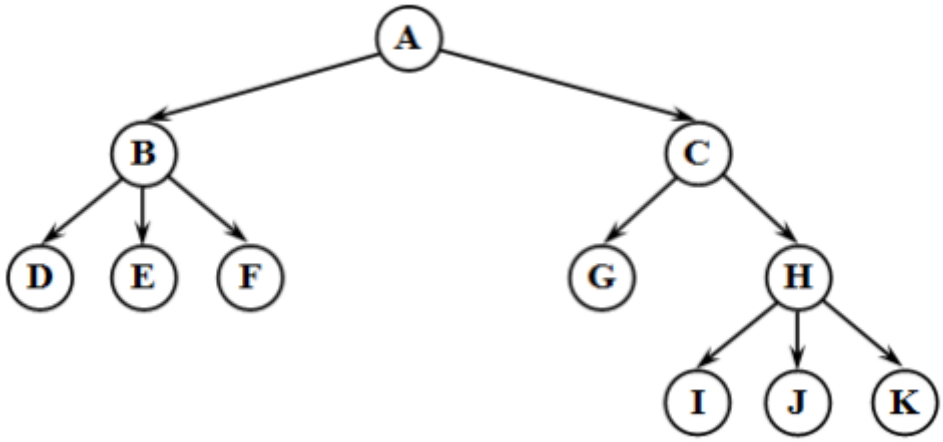

Рисунок 2.1 – Пример дерева

Каждый элемент дерева называется вершиной (узлом) дерева. Вершины дерева соединены направленными дугами, которые называют ветвями дерева. Начальный узел дерева называют корнем дерева. Листьями дерева называют вершины, в которые входит одна ветвь и не выходит ни одной ветви

Каждое дерево обладает следующими свойствами:

− существует узел, в который не входит ни одной ветви (корень);
− в каждую вершину, кроме корня, входит одна ветвь.

Деревья особенно часто используют на практике при изображении различных иерархий.

Все вершины, в которые входят ветви, исходящие из одной общей вершины, называются потомками, а сама вершина – предком. Корень дерева не имеет предка, а листья дерева не имеют потомков. Высота (глубина) дерева определяется количеством уровней, на которых располагаются его вершины. Высота пустого дерева равна нулю, высота дерева из одного корня – единице.

На нулевом уровне дерева может быть только одна вершина – корень дерева, на первом – потомки корня дерева, на втором – потомки потомков корня дерева и т.д.

Поддерево – часть древообразной структуры данных, которая может быть представлена в виде отдельного дерева.

Упорядоченное дерево – это дерево, у которого ветви, исходящие из каждой вершины, упорядочены по определенному критерию.

По величине степени дерева различают два типа деревьев:

− двоичные – степень дерева не более двух;
− сильноветвящиеся – степень дерева более двух.

Бинарное (двоичное) дерево – это динамическая структура данных, представляющее собой дерево, в котором каждая вершина имеет не более двух потомков. Таким образом, бинарное дерево состоит из элементов, каждый из которых содержит информационное поле и не более двух ссылок на различные бинарные поддеревья. На каждый элемент дерева имеется ровно одна ссылка.

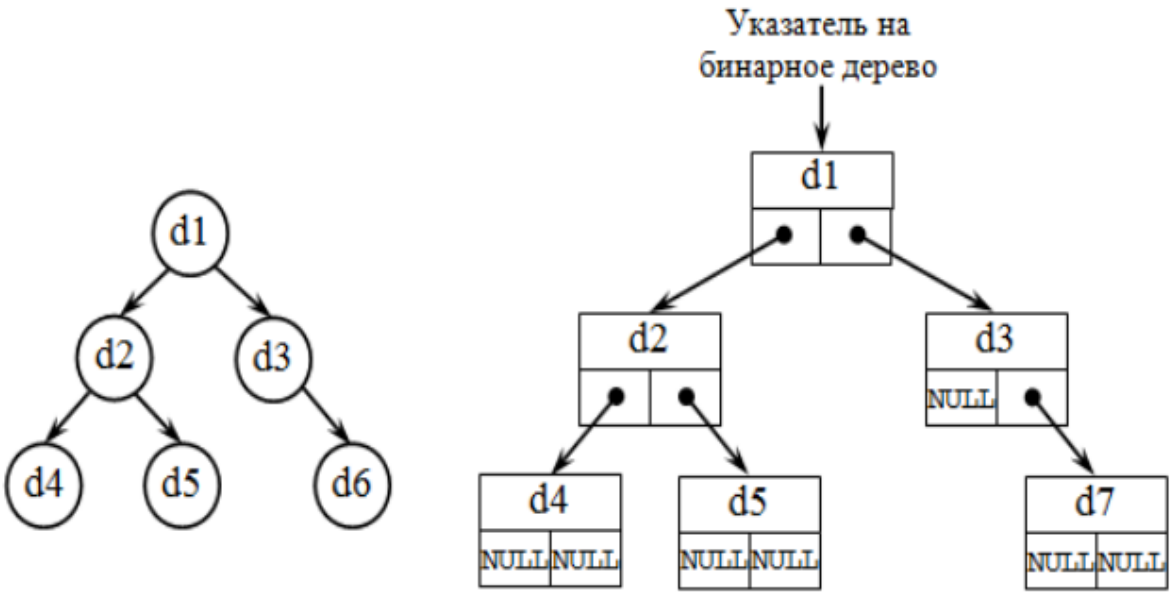

Рисунок 2.2 – Бинарное дерево и его представление

Каждая вершина бинарного дерева является структурой, состоящей, как правило, из четырех видов полей. Содержимым этих полей будут соответственно:

− информационное поле (ключ вершины);

− служебное поле (содержит вспомогательную информацию причем, таких полей может быть несколько или ни одного);

− указатель на левое поддерево;

− указатель на правое поддерево.

По степени вершин бинарные деревья делятся (см. рисунок 5) на:

− строгие – вершины дерева имеют степень ноль (у листьев) или два (у узлов);

− нестрогие – вершины дерева имеют степень ноль (листья), один или два (узлы).

− полные – бинарное дерево содержит только полностью заполненные уровни.

− неполные – бинарное дерево содержит частично заполненные уровни.

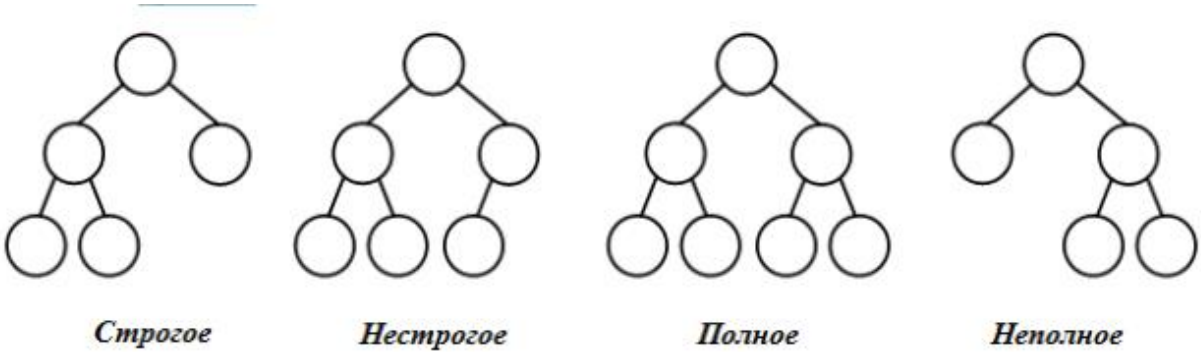

Рисунок 2.3 – Виды бинарных деревьев по степени вершин

2.3 Операции в бинарном дереве

Базовый интерфейс двоичного дерева поиска состоит из трёх операций:

- FIND(K) — поиск узла, в котором хранится пара (key, value) с key = K.
- INSERT(K, V) — добавление в дерево пары (key, value) = (K, V).
- REMOVE(K) — удаление узла, в котором хранится пара (key, value) с key = K.

По сути, двоичное дерево поиска — это структура данных, способная хранить таблицу пар (key, value) и поддерживающая три операции: FIND, INSERT, REMOVE.

Кроме того, интерфейс двоичного дерева включает ещё три дополнительных операции обхода узлов дерева: INFIX_TRAVERSE, PREFIX_TRAVERSE и POSTFIX_TRAVERSE. Первая из них позволяет обойти узлы дерева в порядке неубывания ключей.

Поиск элемента (FIND)

Алгоритм:

- Если дерево пусто, сообщить, что узел не найден, и остановиться.
- Иначе сравнить K со значением ключа корневого узла X.
- Если K=X, выдать ссылку на этот узел и остановиться.
- Если K>X, рекурсивно искать ключ K в правом поддереве T.
- Если K<X, рекурсивно искать ключ K в левом поддереве T.

Добавление элемента (INSERT)

Алгоритм:

- Если дерево пусто, заменить его на дерево с одним корневым узлом ((K, V), null, null) и остановиться.
- Иначе сравнить K с ключом корневого узла X.
    - Если K>X, рекурсивно добавить (K, V) в правое поддерево T.
    - Если K<X, рекурсивно добавить (K, V) в левое поддерево T.
    - Если K=X, заменить V текущего узла новым значением.

Удаление узла (REMOVE)

Алгоритм:

- Если дерево T пусто, остановиться;
- Иначе сравнить K с ключом X корневого узла n.
    - Если K>X, рекурсивно удалить K из правого поддерева T;
    - Если K<X, рекурсивно удалить K из левого поддерева T;
    - Если K=X, то необходимо рассмотреть три случая.
        - Если обоих детей нет, то удаляем текущий узел и обнуляем ссылку на него у родительского узла;
        - Если одного из детей нет, то значения полей ребёнка m ставим вместо соответствующих значений корневого узла, затирая его старые значения, и освобождаем память, занимаемую узлом m;
        - Если оба ребёнка присутствуют, то
            - Если самый левый элемент правого поддерева m не имеет поддеревьев
                - Копируем значения K, V из m в удаляемый элемент
                - Удаляем m
            - Если m имеет правое поддерево

- Копируем значения K, V из m в удаляемый элемент
- Заменяем у родительского узла ссылку на m ссылкой на правое поддерево m
- Удаляем m

Обход дерева (TRAVERSE)

Есть три операции обхода узлов дерева, отличающиеся порядком обхода узлов.

Первая операция — INFIX_TRAVERSE — позволяет обойти все узлы дерева в порядке возрастания ключей и применить к каждому узлу заданную пользователем функцию обратного вызова f, операндом которой является адрес узла. Эта функция обычно работает только с парой (K, V), хранящейся в узле. Операция INFIX_TRAVERSE может быть реализована рекурсивным образом: сначала она запускает себя для левого поддерева, потом запускает данную функцию для корня, потом запускает себя для правого поддерева.

- INFIX_TRAVERSE (tr) — обойти всё дерево, следуя порядку (левое поддерево, **вершина**, правое поддерево). Элементы по возрастанию
- PREFIX_TRAVERSE (tr) — обойти всё дерево, следуя порядку (**вершина**, левое поддерево, правое поддерево). Элементы, как в дереве
- POSTFIX_TRAVERSE (tr) — обойти всё дерево, следуя порядку (левое поддерево, правое поддерево, **вершина**). Элементы в обратном порядке, как в дереве.

В других источниках эти функции именуются inorder, preorder, postorder соответственно.

3.Описание разработанного приложения

3.1 Диаграмма и описание классов

В приложении используются следующие классы: Programm, HelperXML, Form1, Tree, Node, Sportman.

Диаграмма классов представлена ниже

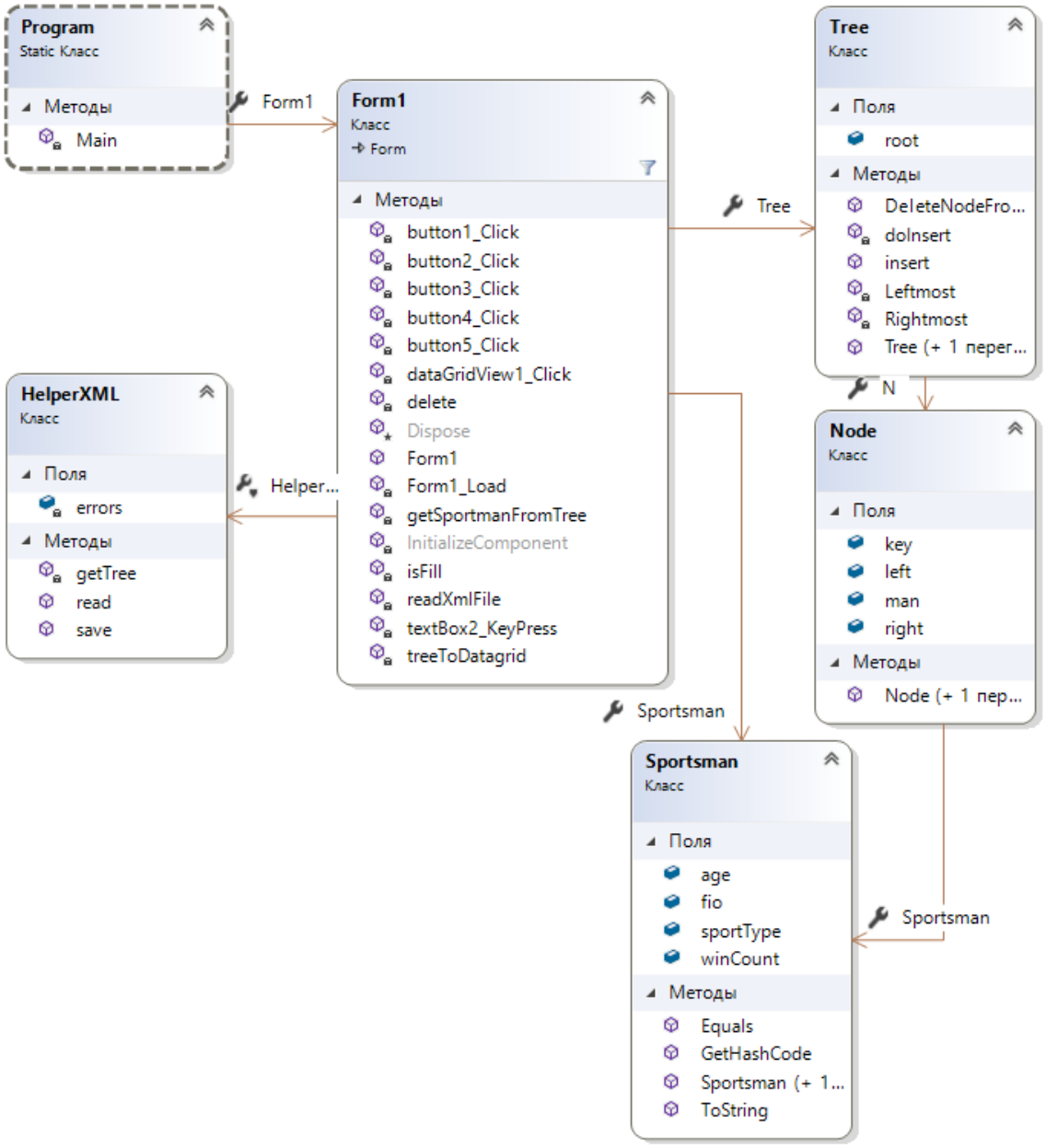

Рисунок 3.1 – Диаграмма классов

Класс Programm является главным классом программы, он запускает форму для отображения.

Класс Form1 служит для отображения интерфейса и взаимодействия с пользователем. При выборе файла для чтения, он загружает все данные файла в память и отображает их на экране.

Для загрузки и сохранения данных, класс Form1 использует другой класс HelperXML и его методы read(), save(). Если нужно вывести данные на форму, он использует метод treeToDatagrid() и клас-сущность Sportsman.

Класс Sportsman – это сущность спортсмена. В нем описаны его данные.

Поскольку нужно использовать бинарное дерево, была написана своя реализация Tree с внутренними Node.

Структура классов

Таблица 5 - HelperXML

Атрибут	**Тип**	**Описание**
Errors	поле	Строка для отображения ошибок при чтении фала
getTree	Матод	Рекурсивный метод обхода дерева
Read	Метод	Метод чтения данных из файла
Save	Метод	Метод сохранения данных в файл

Таблица 6 - Tree

Атрибут	**Тип**	**Описание**
Root	Поле	Корень
deleteNodeFromBinary	Метод	Метод удаления элемента
doInsert	Метод	Метод вставки элемента

insert	Метод	Дополнительный метод вставки элемента
leftmost	Метод	Метод помощник при удалении элемента
rightmost	Метод	Метод помощник при удалении элемента

Таблица 7 - Node

Атрибут	**Тип**	**Описание**
Key	Поле	Ключ
Left	Поле	Левый наследник
Right	Поле	Правый наследник
man	Поле	Спортсмен

Таблица 8 - Sportsman

Атрибут	**Тип**	**Описание**
Fio	Поле	ФИО
Age	Поле	Возраст
winCount	Поле	Количество побед
sportType	Поле	Спорт

Таблица 9 - Form1

Атрибут	**Тип**	**Описание**
Button 1_click	Метод	Метод кнопки добавить
Button 2_click	Метод	Метод кнопки изменить

Button 3_click	Метод	Метод кнопки удалить
Button 4_click	Метод	Метод кнопки сохранить
Button 5_click	Метод	Метод кнопки открыть
dataGridViev1_Click	Метод	Метод нажатия на таблицу. При нажатии на спортсмена, отображает его данные в полях формы
Delete	Метод	Метод удаления спортсмена
getSportmanFromTree	Метод	Метод получение выбранного спортсмена
isFill	Метод	Метод проверки заполнения полей
readXmlFile	Метод	Метод чтения из файла
textBox3_KeyPress	Метод	Метод проверки корректности ввода для возраста и количества побед. Не дает ввести буквы
treeToDatagrid	Метод	Преобразование дерева из памяти в строки таблицы

3.2 Возможности пользователя

После открытия приложения, пользователю будет показана главная страница. На ней он может либо открыть существующий файл, либо сразу начать добавлять спортсменов в список. При нажатии кнопки сохранения,

приложение создаст файл в формате XML со всеми введенными спортсменами. Можно открыть существующий файл, изменить его и сохранить. Во избежание затираний, файл при сохранении создается новый файл с текущей датой и временем.

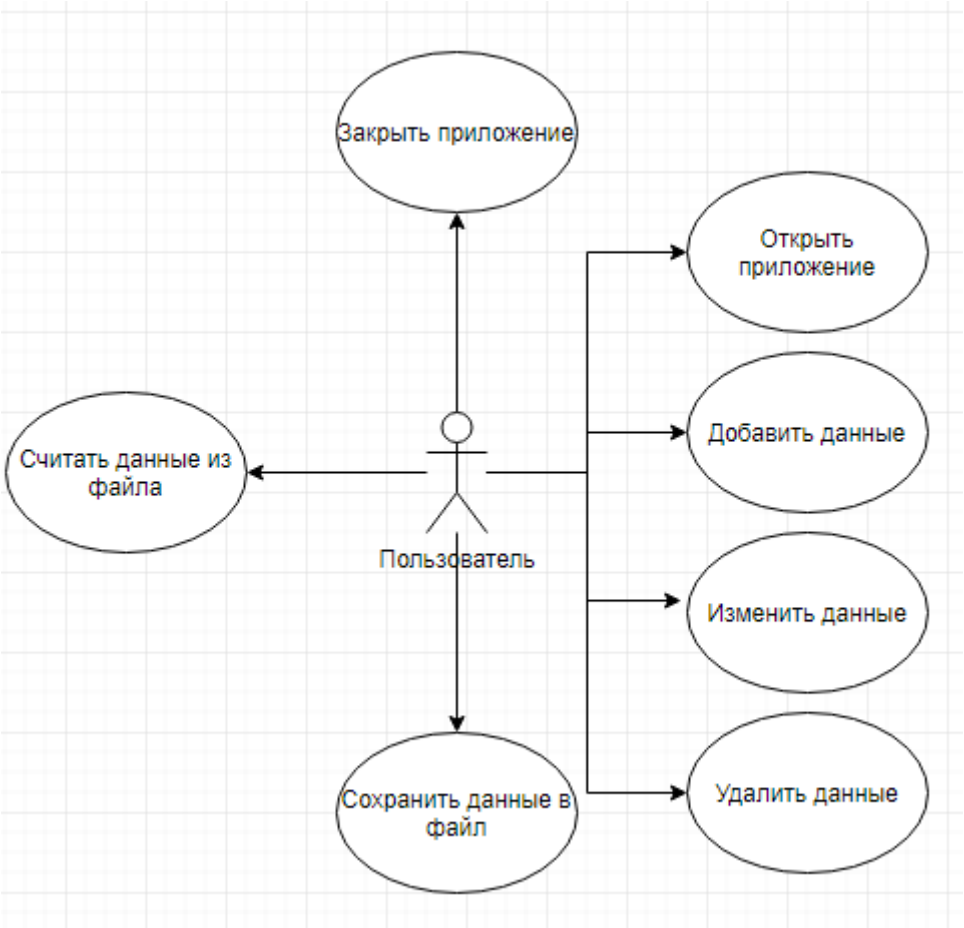

Рисунок 3.2 - UML диаграмма

3.3 Пример работы приложения

После запуска приложения открывается главное окно

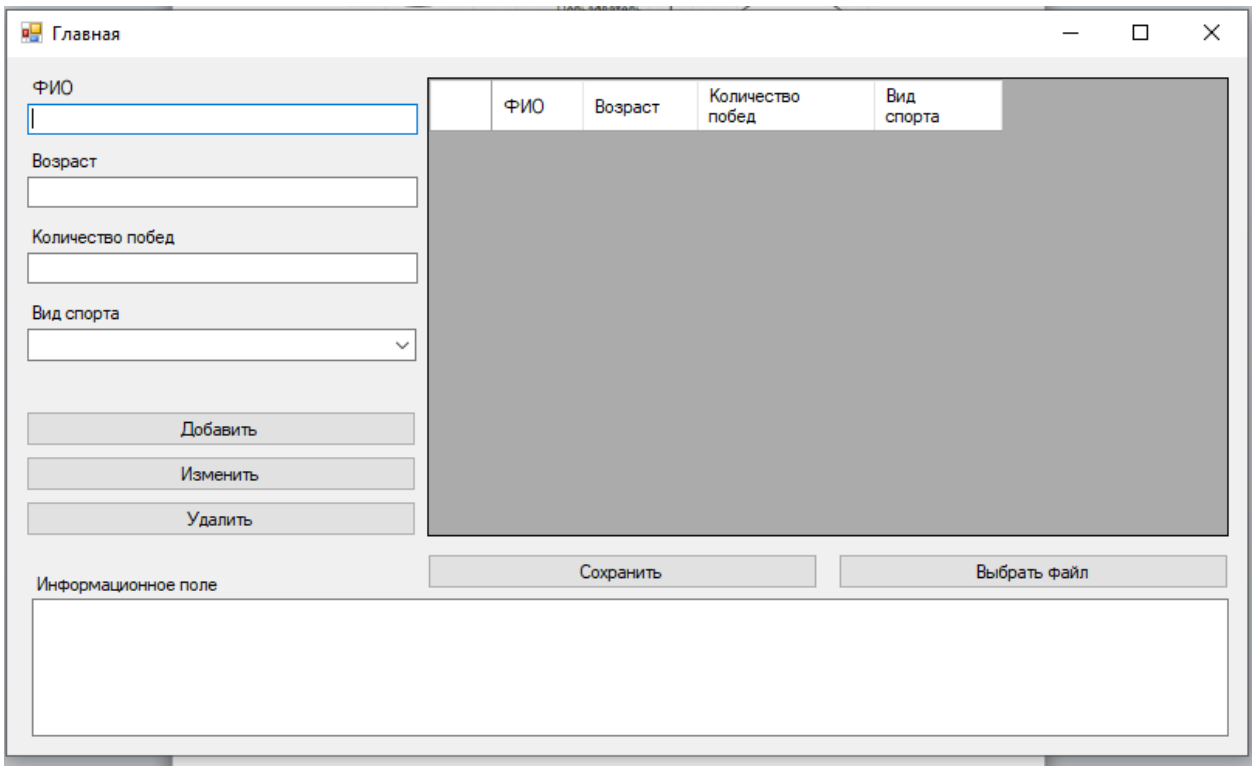

Рисунок 3.3 - Главное окно

На данном моменте можно как считать данные из файла, так и сразу начать вводить данные. Попробуем считать. После нажатия на кнопку выбрать файл открылось диалоговое окно, в котором мы выбираем файл.

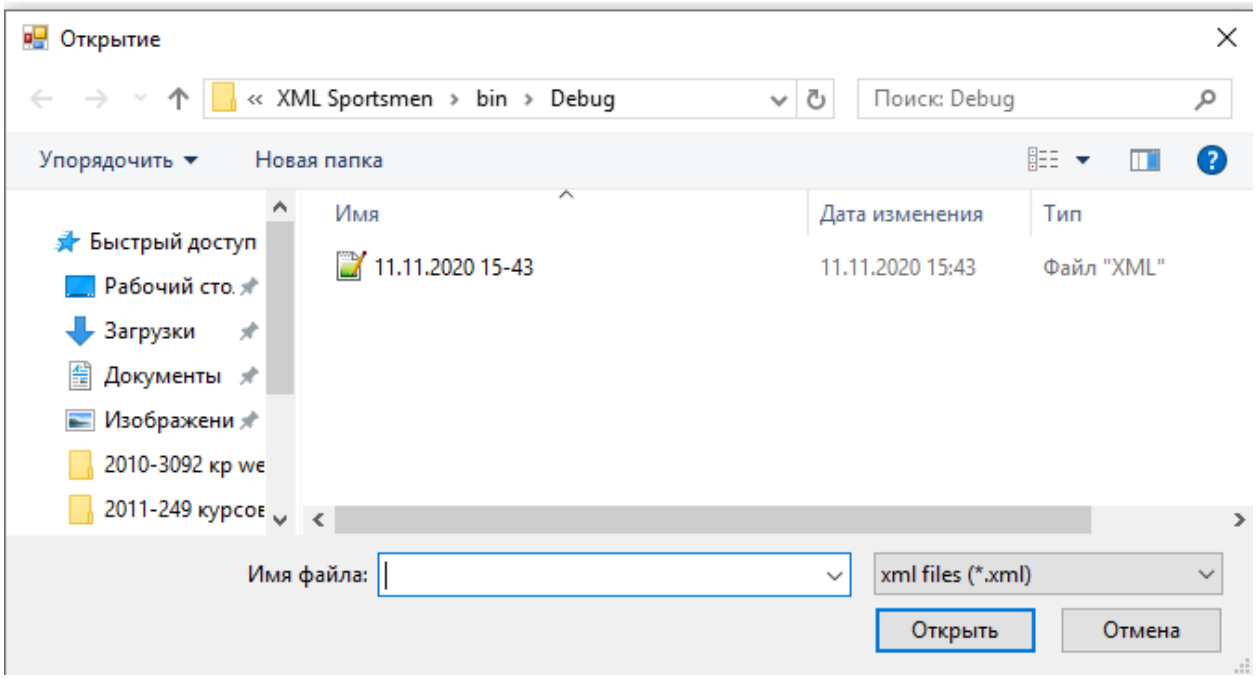

Рисунок 3.4 - Выбор файла

Выбрав файл, все данные из него будут отображены на форме

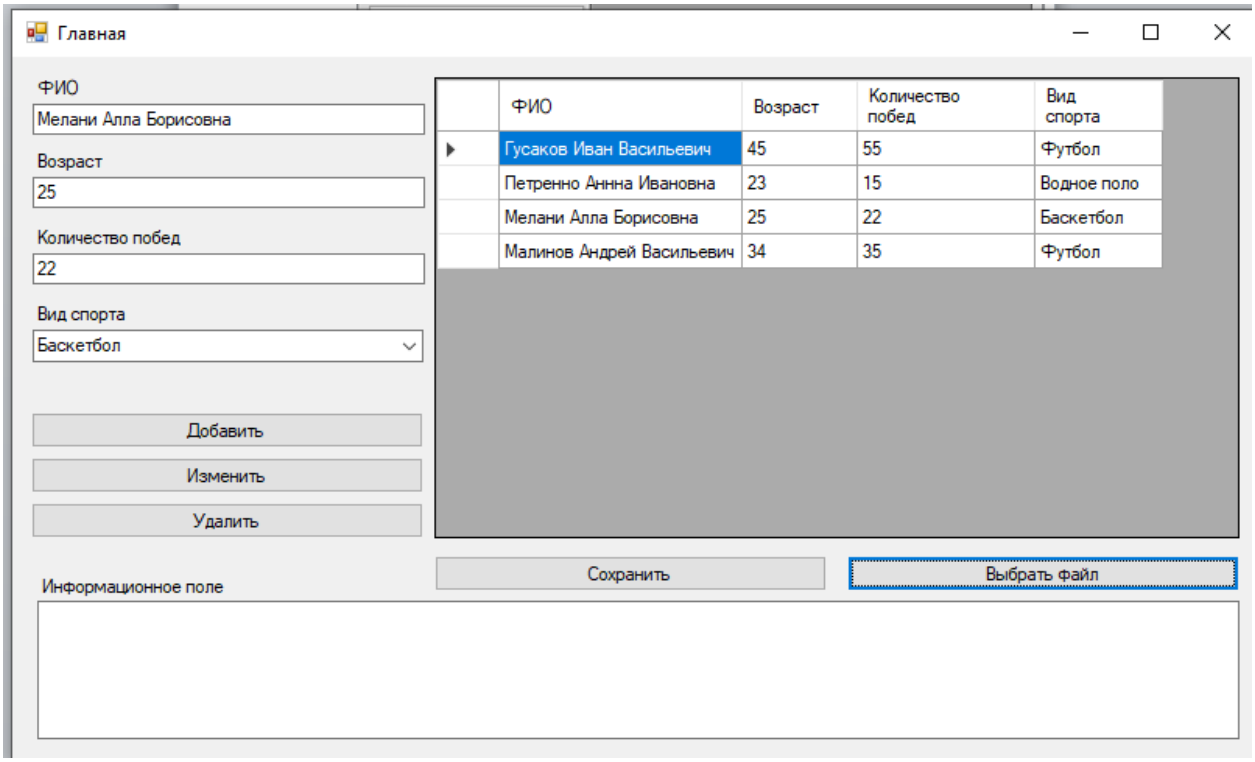

Рисунок 3.5 - Загруженные данные

При нажатии на любого спортсмена, его данные отобразятся в левой части экрана. Если внести туда поправки и нажать изменить, данные обновятся. Аллу изменим на Алену.

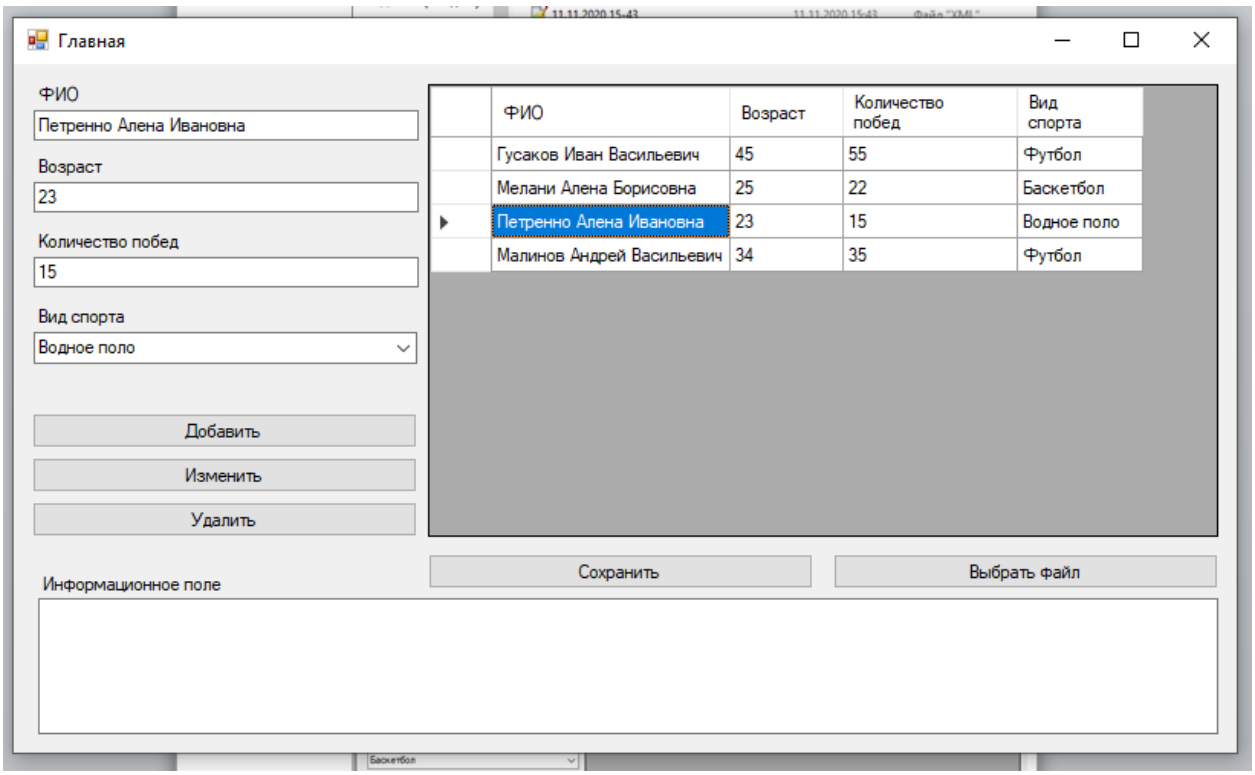

Рисунок 3.6 – Изменение данных

Удаление и изменение работает без проблем.

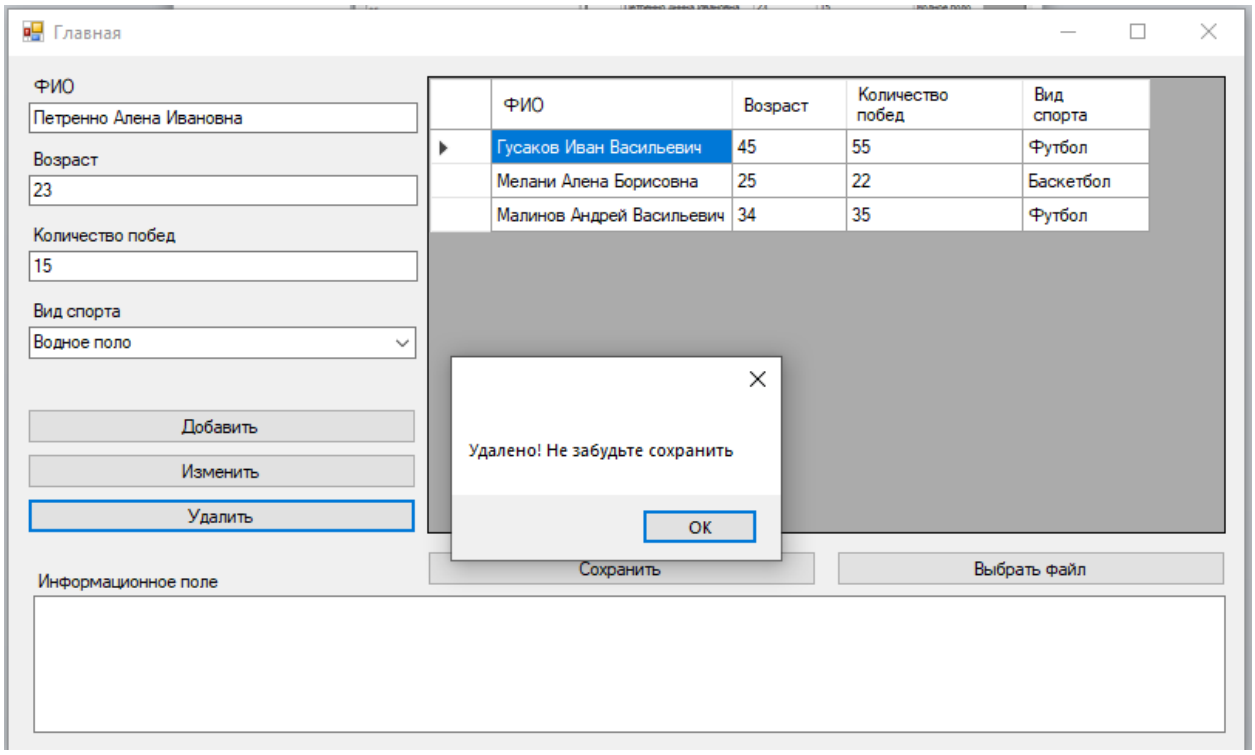

Рисунок 3.7 - Удаление

Важно помнить, что без начатия кнопки сохранить, данные не сохранятся.

Попробуем прочитать файл с ошибочными данными

```
<key>249705924</key>
<man>
  <fio>Гусаков Иван Васильевич</f
  <age>4jh5</age>
  <winCount>55</winCount>
  <sportType>Футбол</sportType>
</man>
```

Рисунок 3.8 – Ошибочные данные

Как результат, внизу экрана отобразится что мы не прочитали

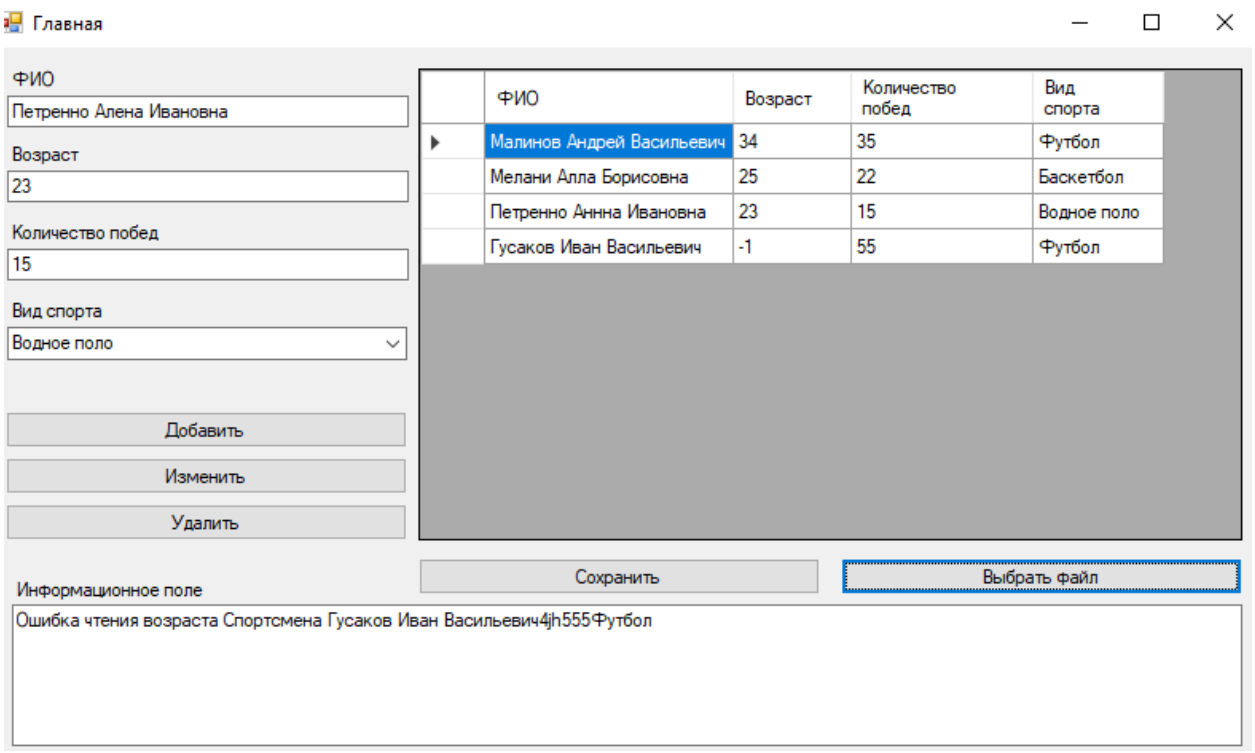

Рисунок 3.9 – Отображение мест ошибок при чтении

Как видно, приложение работает без проблем

Заключение (отражает результаты проделанной работы)

Результатом выполнения данного курсового проекта стало приложение, позволяющее работать со справочником спортсменов.

В процессе выполнения данной курсовой работы были изучены основные возможности языка программирования C#, особенности работы с файлами формата XML, а также получены знания о деревьях, бинарных деревьях и методах работы с ними.

В рамках изучения языка были исследованы технологии работы с объектно-ориентированными языками, приемы обработки данных различных типов, написание своей сущности, работа с WindowsForm.

Итогом выполнения курсовой работы стала информационная справочная система «Спортсмен», а также были получены новые знания и навыки работы с языком программирования C#.

Список использованных источников

1. Троэлсен, Э. C# и платформа .NET. Питер, Санкт-Петербург, 2005. – 230 с.
2. Васильев, А. Программирование на C# для начинающих. Основные сведения. – Эксмо, 2018. – 592 с.
3. ООП в картинках [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/463125/. – Дата доступа: 11.11.2020.
4. Основы ООП [Электронный ресурс]. – Режим доступа: https://www.it-academy.by/course/osnovy-programmirovaniya/osnovy-oop/. – Дата доступа: 11.11.2020.
5. Принципы ООП [Электронный ресурс]. – Режим доступа: https://javarush.ru/groups/posts/1966-principih-obhhektno-orientirovannogo-programmirovanija. – Дата доступа: 11.11.2020.
6. Бинарные деревья [Электронный ресурс]. – Режим доступа: https://habr.com/ru/post/267855/. – Дата доступа: 11.11.2020.
7. Дерево [Электронный ресурс]. – Режим доступа: https://prog-cpp.ru/data-tree/. – Дата доступа: 11.11.2020.
8. Построение диаграмм классов [Электронный ресурс]. – Режим доступа: https://flexberry.github.io/ru/gpg_class-diagram.html. – Дата доступа: 11.11.2020.

Приложение А. Код программы

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
namespace XML_Sportsmen
{
[Serializable]
   public class Sportsman
    {
        public string fio;
        public int age;
        public int winCount;
        public string sportType;

        public Sportsman(string fio, int age, int winCount, string sportType)
        {
            this.fio = fio;
            this.age = age;
            this.winCount = winCount;
            this.sportType = sportType;
        }

        public Sportsman()
        {
        }

        public override bool Equals(object obj)
        {
            return obj is Sportsman sportsman &&
                   fio == sportsman.fio &&
                   age == sportsman.age &&
                   winCount == sportsman.winCount &&
                   sportType == sportsman.sportType;
        }

        public override int GetHashCode()
        {
            var hashCode = 1669081574;
            hashCode = hashCode * -1521134295 + EqualityComparer<string>.Default.GetHashCode(fio);
            hashCode = hashCode * -1521134295 + age.GetHashCode();
            hashCode = hashCode * -1521134295 + winCount.GetHashCode();
            hashCode           =           hashCode           *           -1521134295           +
EqualityComparer<string>.Default.GetHashCode(sportType);
            if (hashCode % 2 == 0) hashCode = -hashCode;
            return hashCode;
        }

        public override string ToString()
        {
            return fio+" "+age + " "+winCount + " "+sportType;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using System.Windows.Forms;
```

```
namespace XML_Sportsmen
{
    static class Program
    {
        public static Form1 Form1
        {
            get => default;
            set
            {
            }
        }

        /// <summary>
        /// Главная точка входа для приложения.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
namespace XML_Sportsmen
{
    partial class Form1
    {
        /// <summary>
        /// Обязательная переменная конструктора.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Освободить все используемые ресурсы.
        /// </summary>
        /// <param name="disposing">истинно, если управляемый ресурс должен быть удален; иначе
ложно.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Код, автоматически созданный конструктором форм Windows

        /// <summary>
        /// Требуемый метод для поддержки конструктора — не изменяйте
        /// содержимое этого метода с помощью редактора кода.
        /// </summary>
        private void InitializeComponent()
        {
            this.label1 = new System.Windows.Forms.Label();
            this.textBox1 = new System.Windows.Forms.TextBox();
            this.textBox2 = new System.Windows.Forms.TextBox();
            this.label2 = new System.Windows.Forms.Label();
            this.textBox3 = new System.Windows.Forms.TextBox();
            this.label3 = new System.Windows.Forms.Label();
            this.label4 = new System.Windows.Forms.Label();
```

```
                    this.comboBox1 = new System.Windows.Forms.ComboBox();
                    this.dataGridView1 = new System.Windows.Forms.DataGridView();
                    this.Column1 = new System.Windows.Forms.DataGridViewTextBoxColumn();
                    this.Column2 = new System.Windows.Forms.DataGridViewTextBoxColumn();
                    this.Column3 = new System.Windows.Forms.DataGridViewTextBoxColumn();
                    this.Column4 = new System.Windows.Forms.DataGridViewTextBoxColumn();
                    this.button1 = new System.Windows.Forms.Button();
                    this.button2 = new System.Windows.Forms.Button();
                    this.button3 = new System.Windows.Forms.Button();
                    this.label5 = new System.Windows.Forms.Label();
                    this.textBox4 = new System.Windows.Forms.TextBox();
                    this.button4 = new System.Windows.Forms.Button();
                    this.button5 = new System.Windows.Forms.Button();
                    ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).BeginInit();
                    this.SuspendLayout();
                    //
                    // label1
                    //
                    this.label1.AutoSize = true;
                    this.label1.Location = new System.Drawing.Point(13, 13);
                    this.label1.Name = "label1";
                    this.label1.Size = new System.Drawing.Size(34, 13);
                    this.label1.TabIndex = 0;
                    this.label1.Text = "ФИО";
                    //
                    // textBox1
                    //
                    this.textBox1.Location = new System.Drawing.Point(13, 30);
                    this.textBox1.Name = "textBox1";
                    this.textBox1.Size = new System.Drawing.Size(252, 20);
                    this.textBox1.TabIndex = 1;
                    //
                    // textBox2
                    //
                    this.textBox2.Location = new System.Drawing.Point(13, 77);
                    this.textBox2.Name = "textBox2";
                    this.textBox2.Size = new System.Drawing.Size(252, 20);
                    this.textBox2.TabIndex = 3;
                    this.textBox2.KeyPress                              +=                              new
System.Windows.Forms.KeyPressEventHandler(this.textBox2_KeyPress);
                    //
                    // label2
                    //
                    this.label2.AutoSize = true;
                    this.label2.Location = new System.Drawing.Point(13, 60);
                    this.label2.Name = "label2";
                    this.label2.Size = new System.Drawing.Size(49, 13);
                    this.label2.TabIndex = 2;
                    this.label2.Text = "Возраст";
                    //
                    // textBox3
                    //
                    this.textBox3.Location = new System.Drawing.Point(13, 126);
                    this.textBox3.Name = "textBox3";
                    this.textBox3.Size = new System.Drawing.Size(252, 20);
                    this.textBox3.TabIndex = 5;
                    this.textBox3.KeyPress                              +=                              new
System.Windows.Forms.KeyPressEventHandler(this.textBox2_KeyPress);
                    //
                    // label3
                    //
                    this.label3.AutoSize = true;
```

```
                    this.label3.Location = new System.Drawing.Point(13, 109);
                    this.label3.Name = "label3";
                    this.label3.Size = new System.Drawing.Size(99, 13);
                    this.label3.TabIndex = 4;
                    this.label3.Text = "Количество побед";
                    //
                    // label4
                    //
                    this.label4.AutoSize = true;
                    this.label4.Location = new System.Drawing.Point(13, 158);
                    this.label4.Name = "label4";
                    this.label4.Size = new System.Drawing.Size(64, 13);
                    this.label4.TabIndex = 6;
                    this.label4.Text = "Вид спорта";
                    //
                    // comboBox1
                    //
                    this.comboBox1.FormattingEnabled = true;
                    this.comboBox1.Items.AddRange(new object[] {
                    "Футбол",
                    "Волейбол",
                    "Баскетбол",
                    "Водное поло",
                    "Регби"});
                    this.comboBox1.Location = new System.Drawing.Point(13, 175);
                    this.comboBox1.Name = "comboBox1";
                    this.comboBox1.Size = new System.Drawing.Size(251, 21);
                    this.comboBox1.TabIndex = 7;
                    //
                    // dataGridView1
                    //
                    this.dataGridView1.AllowUserToAddRows = false;
                    this.dataGridView1.AllowUserToDeleteRows = false;
                    this.dataGridView1.AutoSizeColumnsMode                                                    =
System.Windows.Forms.DataGridViewAutoSizeColumnsMode.AllCells;
                    this.dataGridView1.ColumnHeadersHeightSizeMode                                            =
System.Windows.Forms.DataGridViewColumnHeadersHeightSizeMode.AutoSize;
                    this.dataGridView1.Columns.AddRange(new System.Windows.Forms.DataGridViewColumn[] {
                    this.Column1,
                    this.Column2,
                    this.Column3,
                    this.Column4});
                    this.dataGridView1.Location = new System.Drawing.Point(271, 13);
                    this.dataGridView1.Name = "dataGridView1";
                    this.dataGridView1.ReadOnly = true;
                    this.dataGridView1.Size = new System.Drawing.Size(517, 296);
                    this.dataGridView1.TabIndex = 8;
                    this.dataGridView1.Click += new System.EventHandler(this.dataGridView1_Click);
                    //
                    // Column1
                    //
                    this.Column1.HeaderText = "ФИО";
                    this.Column1.Name = "Column1";
                    this.Column1.ReadOnly = true;
                    this.Column1.Width = 59;
                    //
                    // Column2
                    //
                    this.Column2.HeaderText = "Возраст";
                    this.Column2.Name = "Column2";
                    this.Column2.ReadOnly = true;
                    this.Column2.Width = 74;
```

```
//
// Column3
//
this.Column3.HeaderText = "Количество побед";
this.Column3.Name = "Column3";
this.Column3.ReadOnly = true;
this.Column3.Width = 114;
//
// Column4
//
this.Column4.HeaderText = "Вид спорта";
this.Column4.Name = "Column4";
this.Column4.ReadOnly = true;
this.Column4.Width = 82;
//
// button1
//
this.button1.Location = new System.Drawing.Point(12, 228);
this.button1.Name = "button1";
this.button1.Size = new System.Drawing.Size(252, 23);
this.button1.TabIndex = 9;
this.button1.Text = "Добавить";
this.button1.UseVisualStyleBackColor = true;
this.button1.Click += new System.EventHandler(this.button1_Click);
//
// button2
//
this.button2.Location = new System.Drawing.Point(12, 257);
this.button2.Name = "button2";
this.button2.Size = new System.Drawing.Size(252, 23);
this.button2.TabIndex = 10;
this.button2.Text = "Изменить";
this.button2.UseVisualStyleBackColor = true;
this.button2.Click += new System.EventHandler(this.button2_Click);
//
// button3
//
this.button3.Location = new System.Drawing.Point(12, 286);
this.button3.Name = "button3";
this.button3.Size = new System.Drawing.Size(252, 23);
this.button3.TabIndex = 11;
this.button3.Text = "Удалить";
this.button3.UseVisualStyleBackColor = true;
this.button3.Click += new System.EventHandler(this.button3_Click);
//
// label5
//
this.label5.AutoSize = true;
this.label5.Location = new System.Drawing.Point(16, 333);
this.label5.Name = "label5";
this.label5.Size = new System.Drawing.Size(124, 13);
this.label5.TabIndex = 12;
this.label5.Text = "Информационное поле";
//
// textBox4
//
this.textBox4.Location = new System.Drawing.Point(16, 349);
this.textBox4.Multiline = true;
this.textBox4.Name = "textBox4";
this.textBox4.Size = new System.Drawing.Size(772, 89);
this.textBox4.TabIndex = 13;
//
```

```
    // button4
    //
    this.button4.Location = new System.Drawing.Point(271, 320);
    this.button4.Name = "button4";
    this.button4.Size = new System.Drawing.Size(252, 23);
    this.button4.TabIndex = 14;
    this.button4.Text = "Сохранить";
    this.button4.UseVisualStyleBackColor = true;
    this.button4.Click += new System.EventHandler(this.button4_Click);
    //
    // button5
    //
    this.button5.Location = new System.Drawing.Point(536, 320);
    this.button5.Name = "button5";
    this.button5.Size = new System.Drawing.Size(252, 23);
    this.button5.TabIndex = 15;
    this.button5.Text = "Выбрать файл";
    this.button5.UseVisualStyleBackColor = true;
    this.button5.Click += new System.EventHandler(this.button5_Click);
    //
    // Form1
    //
    this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
    this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
    this.ClientSize = new System.Drawing.Size(800, 450);
    this.Controls.Add(this.button5);
    this.Controls.Add(this.button4);
    this.Controls.Add(this.textBox4);
    this.Controls.Add(this.label5);
    this.Controls.Add(this.button3);
    this.Controls.Add(this.button2);
    this.Controls.Add(this.button1);
    this.Controls.Add(this.dataGridView1);
    this.Controls.Add(this.comboBox1);
    this.Controls.Add(this.label4);
    this.Controls.Add(this.textBox3);
    this.Controls.Add(this.label3);
    this.Controls.Add(this.textBox2);
    this.Controls.Add(this.label2);
    this.Controls.Add(this.textBox1);
    this.Controls.Add(this.label1);
    this.Name = "Form1";
    this.Text = "Главная";
    this.Load += new System.EventHandler(this.Form1_Load);
    ((System.ComponentModel.ISupportInitialize)(this.dataGridView1)).EndInit();
    this.ResumeLayout(false);
    this.PerformLayout();

}

#endregion

private System.Windows.Forms.Label label1;
private System.Windows.Forms.TextBox textBox1;
private System.Windows.Forms.TextBox textBox2;
private System.Windows.Forms.Label label2;
private System.Windows.Forms.TextBox textBox3;
private System.Windows.Forms.Label label3;
private System.Windows.Forms.Label label4;
private System.Windows.Forms.ComboBox comboBox1;
private System.Windows.Forms.DataGridView dataGridView1;
private System.Windows.Forms.Button button1;
```

```
        private System.Windows.Forms.Button button2;
        private System.Windows.Forms.Button button3;
        private System.Windows.Forms.Label label5;
        private System.Windows.Forms.TextBox textBox4;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column1;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column2;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column3;
        private System.Windows.Forms.DataGridViewTextBoxColumn Column4;
        private System.Windows.Forms.Button button4;
        private System.Windows.Forms.Button button5;
    }
}

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace XML_Sportsmen
{
    [Serializable]
public class Tree
    {


        public Node root;

        public Tree(Node root)
        {
            this.root = root;
        }

        public Tree(){}

        public Node Node
        {
            get => default;
            set
            {
            }
        }

        public void insert(Sportsman man_)
        {
            root = doInsert(root, man_.GetHashCode(),man_);
        }

        private static Node doInsert(Node node, int x, Sportsman man_)
        {
            if (node == null)
            {
                return new Node(x,man_);
            }
            if (x < node.key)
            {
                node.left = doInsert(node.left, x, man_);
            }
            else if (x > node.key)
            {
                node.right = doInsert(node.right, x, man_);
            }
```

```
    return node;
}


public void DeleteNodeFromBinary(ref Node node,ref Node parent,int key)
{
    if (node == null)
        return;

    if (key < node.key)
        DeleteNodeFromBinary(ref node.left, ref node, key);

    else if (key > node.key)
        DeleteNodeFromBinary(ref node.right, ref node, key);
    //deleting
    else
    {
        //если оба потомка нулевые
        if (node.left == null && node.right == null)
        {
            //и удаляем не корень
            if (parent != node)
            {
                if (parent.left == node)
                    parent.left = null;
                else
                    parent.right = null;
            }
            node = null;
        }
        else
        {
            Node newnode = null;
            //если есть только один потомок
            if (
                (node.left != null && node.right == null) ||
                (node.left == null && node.right != null)
                )
            {
                //если только левый
                if (node.left != null && node.right == null)
                    //newnode = Rightmost(ref node.left);
                    node = node.left;
                //если только правый
                else
                    //newnode = Leftmost(ref node.right);
                    node = node.right;

                //если один потомок и удаляем не корень
                //if (parent != node)
                //{
                //    if (parent.left == node)
                //        parent.left = newnode;
                //    else
                //        parent.right = newnode;

                //    newnode.right = node.right;
                //    newnode.left = node.left;
                //    node = null;
                //}
                ////если один потомок и удаляем корень
                //else
```

```
                        //{
                        //    if (node.left != null)
                        //        node = node.left;
                        //    else
                        //        node = node.right;
                        //}
                    }
                    //если есть оба потомка
                    else
                    {

                        newnode = node.left;
                        Node pmin = node.right;
                        while (pmin.left != null)
                        {
                            pmin = pmin.left;
                        }
                        pmin.left = node.left;
                        node = node.right;
                    }
                }
            }
        }
        Node Leftmost(ref Node node)
        {
            if (node == null)
                return null;
            if (node.left!= null)
            {
                return Leftmost(ref node.left);
            }
            return node;
        }
        Node Rightmost(ref Node node)
        {
            if (node == null)
                return null;
            if (node.right!= null)
            {
                return Rightmost(ref node.right);
            }
            return node;
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace XML_Sportsmen
{
    [Serializable]
    public class Node
    {
        public int key;
        public Sportsman man;
        public Node left;
        public Node right;

        public Node(int key, Sportsman man)
```

```
        {
            this.key = key;
            this.man = man;
        }
        public Node() { }

        public Sportsman Sportsman
        {
            get => default;
            set
            {
            }
        }
    }

}
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;
using System.Xml;
using System.Xml.Serialization;

namespace XML_Sportsmen
{
    class HelperXML
    {
        string errors = "";
        public bool save(string path, Tree tree)
        {
            try
            {
                XmlSerializer formatter = new XmlSerializer(typeof(Tree));

                using (FileStream fs = new FileStream(path, FileMode.OpenOrCreate))
                {
                    formatter.Serialize(fs, tree);

                    Console.WriteLine("Объект сериализован");
                }
            }
            catch (Exception e)
            {
                Console.WriteLine(e.Message);
                return false;
            }
            return true;
        }

        public object[] read(string pathToXml)
        {
            Stack<Sportsman> stack = new Stack<Sportsman>();

            Tree tree = new Tree();
            errors = "";
            XmlDocument xmlDoc = new XmlDocument();
            xmlDoc.Load(pathToXml);

            if (xmlDoc.ChildNodes.Count > 1)
```

```
                    getTree(stack, xmlDoc.ChildNodes[1].FirstChild);

                stack.Reverse();

                while (stack.Count > 0)
                    tree.insert(stack.Pop());
                Console.WriteLine(errors);
                return new object[] { tree, errors };
            }
            private void getTree(Stack<Sportsman> stack, XmlNode parent)
            {
                if (parent == null) return;



                    string fio, sportType;
                    int age, winCount;

                    XmlNode node = parent["man"];

                    try {
                        fio = node["fio"].InnerText;
                        if (fio.Length == 0)
                            throw new FormatException();
                    }
                    catch (Exception e) {
                        fio = "ERROR";
                        errors += "Ошибка чтения имени в " + node.InnerText + "\r\n";
                    }


                    try {
                        age = int.Parse(node["age"].InnerText);

                    }
                    catch (Exception e) {
                        age = -1;
                        errors += "Ошибка чтения возраста Спортсмена " + node.InnerText + "\r\n";
                    }


                    try {
                        winCount = int.Parse(node["winCount"].InnerText);
                    }
                    catch (Exception e) {
                        winCount = -1;
                        errors += "Ошибка чтения количества побед Спортсмена " + node.InnerText +
"\r\n";
                    }


                    try {
                        sportType = node["sportType"].InnerText;
                        if (sportType.Length == 0)
                            throw new FormatException();
                    }
                    catch (Exception e) {
                        sportType = "ERROR";
                        errors += "Ошибка чтения спорта Спортсмена " + node.InnerText + "\r\n";
                    }

                    Sportsman man = new Sportsman(fio, age, winCount, sportType);
```

```
                stack.Push(man);

                getTree(stack, parent["left"]);
                getTree(stack, parent["right"]);


        }
    }
}
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Xml;
using System.Windows.Forms;

namespace XML_Sportsmen
{
    public partial class Form1 : Form
    {
        Tree tree = new Tree();

        string pathToXml;
        HelperXML helper = new HelperXML();
        public Form1()
        {
            InitializeComponent();
        }

        public Tree Tree
        {
            get => default;
            set
            {
            }
        }

        internal HelperXML HelperXML
        {
            get => default;
            set
            {
            }
        }

        public Sportsman Sportsman
        {
            get => default;
            set
            {
            }
        }

        private void Form1_Load(object sender, EventArgs e)
        {
            pathToXml = DateTime.Now.ToShortDateString().ToString() + ".xml";
        }
```

```
                //сохранить
                private void button4_Click(object sender, EventArgs e)
                {
                    pathToXml          =          DateTime.Now.ToShortDateString().ToString()+"          "+
DateTime.Now.ToShortTimeString().ToString().Replace(":","-")+ ".xml";
                    if (helper.save(pathToXml, tree))
                    {
                        MessageBox.Show("Сохранено успешно!");
                        readXmlFile();
                    }
                    else
                        MessageBox.Show("Ошибка сохранения!");
                }

                private void readXmlFile()
                {
                    //throw new NotImplementedException();
                }

                private bool isFill() {
                    if (textBox1.Text.Length < 1 ||
                        textBox1.Text.Length < 1 ||
                        textBox1.Text.Length < 1 ||
                        comboBox1.Text.Length < 1) {
                        MessageBox.Show("Не все поля заполнены");
                        return false;
                    }
                    return true;
                }

                private void textBox2_KeyPress(object sender, KeyPressEventArgs e)
                {
                    char number = e.KeyChar;
                    if (!Char.IsDigit(number) && number != 8) // цифры и клавиша BackSpace
                    {
                        e.Handled = true;
                    }
                }
                //add
                private void button1_Click(object sender, EventArgs e)
                {
                    try
                    {
                        if (!isFill()) return;
                        Sportsman    man    =    new    Sportsman(textBox1.Text,    int.Parse(textBox2.Text),
int.Parse(textBox3.Text), comboBox1.Text);
                        tree.insert( man);
                        treeToDatagrid();
                        MessageBox.Show("Предварительно добавлен. Не забудьте сохранить");
                    }
                    catch (Exception ex) {
                        MessageBox.Show("Ошибка добавления");
                    }
                }
                //выбрать файл
                private void button5_Click(object sender, EventArgs e)
                {
                    OpenFileDialog OpenFile = new OpenFileDialog();
                    OpenFile.Filter = "xml files (*.xml)|*.xml";
                    if (OpenFile.ShowDialog() == System.Windows.Forms.DialogResult.OK)
                    {
                        textBox4.Text = "";
```

```
                pathToXml = OpenFile.FileName;
                object[]obj= helper.read(pathToXml);
                tree = (Tree)obj[0];
                textBox4.Text = (string)obj[1];
                treeToDatagrid();
            }
        }

        private void treeToDatagrid() {
            dataGridView1.Rows.Clear();

            getSportmanFromTree(tree.root);
        }
        private void getSportmanFromTree(Node parent) {
            if (parent == null) return;

            dataGridView1.Rows.Add(new object []{ parent.man.fio, parent.man.age,
parent.man.winCount, parent.man.sportType });
            getSportmanFromTree(parent.left);
            getSportmanFromTree(parent.right);

        }
        //delete
        private void button3_Click(object sender, EventArgs e)
        {
            if (dataGridView1.Rows.Count > 0 && dataGridView1.CurrentRow != null) {
                try
                {
                    delete();
                    treeToDatagrid();
                    MessageBox.Show("Удалено! Не забудьте сохранить");
                }
                catch (Exception ex) {
                    MessageBox.Show("Ошибка удаления!");
                }
            }
        }
        private Sportsman delete() {
            string fio, sportType;
            int age, winCount;
            fio = dataGridView1[0, dataGridView1.CurrentRow.Index].Value.ToString();
            age = int.Parse(dataGridView1[1, dataGridView1.CurrentRow.Index].Value.ToString());
            winCount = int.Parse(dataGridView1[2,
dataGridView1.CurrentRow.Index].Value.ToString());
            sportType = dataGridView1[3, dataGridView1.CurrentRow.Index].Value.ToString();
            Sportsman sportsman = new Sportsman(fio, age, winCount, sportType);

            tree.DeleteNodeFromBinary(ref tree.root, ref tree.root, sportsman.GetHashCode());

            return sportsman;
        }
        //изменить
        private void button2_Click(object sender, EventArgs e)
        {

            if (dataGridView1.Rows.Count < 1 || dataGridView1.CurrentRow == null) return;
            try
            {
                if (!isFill()) return;
                delete();
                Sportsman man = new Sportsman(textBox1.Text, int.Parse(textBox2.Text),
int.Parse(textBox3.Text), comboBox1.Text);
```

```
                    tree.insert(man);

                    treeToDatagrid();

                }
                catch (Exception ex)
                {
                    MessageBox.Show("Ошибка изменения");
                }
            }

            private void dataGridView1_Click(object sender, EventArgs e)
            {
                if (dataGridView1.Rows.Count > 0 && dataGridView1.CurrentRow != null)
                {
                    textBox1.Text = dataGridView1[0, dataGridView1.CurrentRow.Index].Value.ToString();
                    textBox2.Text = dataGridView1[1, dataGridView1.CurrentRow.Index].Value.ToString();
                    textBox3.Text = dataGridView1[2, dataGridView1.CurrentRow.Index].Value.ToString();
                    comboBox1.Text                          =                          dataGridView1[3,
dataGridView1.CurrentRow.Index].Value.ToString();
                }
            }
        }
    }
```

Приложение Б. Руководство пользователя

Для запуска приложения, 2 раза кликните на ярлык XML Sportsmen.exe. После этого перед вами появится форма отображения.

Если у вас уже есть XML-файл с данными, вы можете загрузить его в форму. Для этого нажмите на кнопку «Выбрать файл». После этого откроется меню выбора файла, выберите нужный файл и нажмите открыть. Сразу после этого на форме будут отображены данные с файла. Если в данных файла есть ошибка, при открытии она будет отображена в нижней части формы.

Для добавления нового спортсмена заполните все поля с левой части приложения и нажмите кнопку «Добавить». Вы не сможете добавить нового спортсмена, если не будут заполнены все поля. Вам покажется уведомление об этом.

Если необходимо изменить данные о каком-то спортсмене, выберите его из списка, введите новые данные в левую часть экрана и нажмите кнопку «Изменить». Вы не сможете изменить данные, если не заполнены все поля или не выбран спортсмен для изменения.

ВАЖНО. Все внесенные или измененные данные не будут сохранены, пока вы не нажмете соответствующую кнопку – «Сохранить».