БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Кафедра программного обеспечения информационных технологий

Факультет НиДО

Специальность ПОИТ

Курсовая работа

по дисциплине «Основы алгоритмизации и программирования»

тема: «Программное средство "Каталог ПО на CD-дисках"»

Выполнила: Н. Г. Карпеко

Договор № 696 от 16.05.2017г.

Минск 2019

Учреждение образования

«Белорусский государственный университет информатики и радиоэлектроники»

Факультет информационных технологий и управления

УТВЕРЖДАЮ Заведующий кафедрой

(подпись)

Лапицкая Н.В. 2018 г.

ЗАДАНИЕ

по курсовому проектированию

Студенту *Карпеко Наталье Георгиевне*

1. Тема проекта Программное средство "Каталог ПО на CD-дисках"

2. Срок сдачи студентом законченного проекта *20 июня 2018 г*

3. Исходные данные к проекту *ОС Windows, язык программирования Pascal, среда разработки Delphi, файл с данными.*

4. Содержание расчетно-пояснительной записки (перечень вопросов, которые подлежат разработке)

*Введение.*

*1.Аналитический обзор литературы и существующих аналогов; 2.Разработка алгоритма;*

*3. Разработка программного средства;*

*4. Обоснование технических приемов программирования;*

*5. Тестирование, экспериментальные исследования и анализ полученных результатов;*

*6. Руководство пользователя программы;*

*Заключение, список использованных источников, листинг программы.*

5. Перечень графического материала (с точным обозначением обязательных чертежей и графиков) ––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––––

6. Консультант по проекту (с обозначением разделов проекта) *Глухова Л.А.*

7. Дата выдачи задания *1июня 2018г*

8. Календарный график работы над проектом на весь период проектирования (с обозначением сроков выполнения и трудоемкости отдельных этапов): *разделы 1,2 к 04.06 – 15 %;*

*раздел 3 к 06.06 – 10 %;*

*разделы 4,5 к 11.06 – 10 %;*

*разделы 6,7 к 14.06 – 35 %;*

*раздел 8 к 18.06 – 5 %;*

*раздел 9 к 22.06 – 10 %;*

*оформление пояснительной записки и графического материала к 25.06 – 15 % Защита курсового проекта с 26.06 по 27.06.*

РУКОВОДИТЕЛЬ  *Л.А.Глухова*

(подпись)

Задание приняла к исполнению *1.06.2018г Н.Г. Карпеко*

(дата и подпись студента)

Содержание

Введение ................................................................................................................. 5

1. Аналитический обзор литературы ................................................................... 6

1.1 Динамические структуры данных .................................................................. 6

1.2 Постановка задачи .......................................................................................... 10

2. Разработка алгоритма ....................................................................................... 11

3. Разработка программного средства ............................................................... 13

3.1. Общие понятия разработки программного средства ................................ 13

3.2. Требования к программной реализации ..................................................... 13

3.3. Описание основных процедур и функций модулей программы .............. 13

4. Обоснование технических приемов программирования ............................. 15

4.1. Критерии хорошей архитектуры программы ............................................ 15

4.2. Декомпозиция как основа модульной архитектуры ................................. 16

4.3. Технические приемы разработки данной программы в среде Delphi ..... 17

5. Тестирование, экспериментальные исследования и анализ полученных

результатов ...................................................................................................... 18

5.1. Основные определения, этапы и принципы тестирования ....................... 18

5.2. Тестирование программы ............................................................................ 22

6. Руководство пользователя программы ......................................................... 24

Заключение ........................................................................................................... 30

Список использованных источников ................................................................. 31

Листинг программы с использованием динамических переменных .............. 32

Листинг программы на языке Delphi ................................................................. 47

**ВВЕДЕНИЕ**

Данный курсовой проект является заключительным этапом в изучении предмета «Основы алгоритмизации и программирования».

Целью данной курсовой работы является разработка программного средства для управления базой данных – "Каталог программного обеспечения (ПО) на CD-дисках". Разработанное программное средство обеспечивает систематизацию информации об имеющемся Программном Обеспечении: добавление, удаление записей, их поиск и сортировку.

Для решения данной комплексной задачи была выбрана среда разработки Delphi, где используется язык Object Pascal.

Пояснительная записка содержит следующие разделы:

1. Аналитический обзор литературы, где приведен сравнительный анализ существующих по данной тематике технических научных решений.

2. Разработка алгоритма выполнена согласно ГОСТ 19.701-90.

3. Разработка программного средства – описаны основные понятия, требования к программной реализации и описаны основные процедуры и функции программы.

4. Обоснование технических приемов программирования – в данном разделе записки приведено обоснование архитектурного построения программы.

5. Тестирование, экспериментальные исследования и анализ полученных результатов, в котором описаны способы проверки правильности и работоспособности отдельных функций программы системы в целом.

6. Руководство пользователя программы, где указана последовательность действий, выполняемых пользователем для успешной работы системы.

Заключение содержит основные результаты выполненной работы в рамках данного курсового проекта.

Список использованных источников включает в себя ссылки на литературу, нормативно-техническую и другую документацию, иные источники, использованные при составлении текста пояснительной записки.

Разделы Листинг программы с использованием динамических переменных, а также Листинг программы на языке Delphi – содержат код программы, написанные в среде разработки Delphi с использованием динамических списков, и на самом языке Delphi соответственно.

**1. АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ**

1.1 Динамические структуры данных

Динамические структуры данных - данные, размер которых может меняться во время работы программы.

Обычно переменная хранит некоторые данные. Однако помимо обычных, существуют переменные, которые ссылаются на другие переменные. Такие переменные называются указателями [1].

Указатель — это переменная, значением которой является адрес другой переменной или структуры данных. Графически указатель можно изобразить так, как на рисунке 1.1.

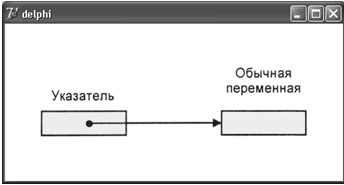


Рисунок 1.1. Переменная-указатель

Указатель, как и любая другая переменная программы, должна быть объявлена в разделе объявления. В общем виде объявление указателя выглядит следующим образом: Имя: ^ Тип;

где: имя — имя переменной-указателя; Тип — тип переменной, на которую указывает переменная-указатель; ^ - объявляемая переменная является указателем.

Тип переменной, на которую ссылается указатель, называют типом указателя. Например, если в программе объявлен указатель р: ^integer, то говорят: ^р — "указатель целого типа" или "р — это указатель на целое число".

В начале работы программы переменная-указатель "ни на что не указывает". В этом случае говорят, что значение указателя равно NIL.

Идентификатор NIL можно использовать в инструкциях присваивания и в условиях. Например, если переменные p1 и р2 объявлены как указатели, то инструкция p1 := NIL;

устанавливает значение переменной, а инструкция if р2 = NIL then ShowMessage('Указатель р2 не инициализирован!');

проверяет, инициализирован ли указатель р2.

Указатель можно использовать для доступа к переменной, адрес которой содержит указатель. Например, если р указывает на переменную i, то в результате выполнения инструкции: р^ : = 5;

значение переменной i будет равно пяти. В приведенном примере значок ^ показывает, что значение пять присваивается переменной, на которую указывает переменная-указатель.

Динамическая переменная – это переменная, память для которой выделяется во время работы программы и осуществляется вызовом процедуры new. У процедуры new один параметр — указатель на переменную того типа, память для которой надо выделить. Например, если р является указателем на тип real, то в результате выполнения процедуры new(p); будет выделена память для переменной типа real (создана переменная типа real), и переменная-указатель р будет содержать адрес памяти, выделенной для этой переменной.

У динамической переменной нет имени, поэтому обратиться к ней можно только при помощи указателя.

Процедура Dispose используется для освобождения занимаемой переменной памяти. Пример создания, использования и уничтожения динамических переменных

Var p1, p2, p3: ^ Integer; // указатели на переменные типа integer

begin

// создание динамических переменных типа integer

// выделение памяти для динамических переменных

New(p1);

New(p2);

New(p3);

р1^ := 5;

р2^ := 3;

р3^ := р1^ + р2^;

ShowMessage('Сумма чисел равна ' + IntToStr(р3^));

// уничтожение динамических переменных, освобождение памяти

Dispose(p1);

Dispose(р2);

Dispose(р3);

end;

Динамические связанные списки

Список можно изобразить графически следующим образом (рис 1.2):

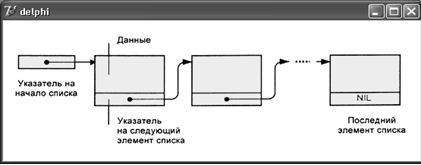


Рисунок 1.2. Графическое изображение списка.

Каждый элемент списка (узел) представляет собой запись, состоящую из двух частей. Первая часть — информационная. Вторая часть отвечает за связь со следующим и, возможно, с предыдущим элементом списка. Список, в котором обеспечивается связь только со следующим элементом, называется односвязным.

Для того чтобы программа могла использовать список, надо определить тип компонентов списка и переменную-указатель на первый элемент списка. Ниже приведен пример объявления компонента списка студентов:

type

TPStudent = ^TStudent; // указатель на переменную типа TStudent

// описание типа элемента списка

TStudent = record

FIO: string[60]; // фамилия

group: integer; // номер группы

next: TPStudent; // указатель на следующий элемент списка

end;

var head: TPStudent; // указатель на первый элемент списка

Преимуществом связанных списков по сравнению со статическими массивами является отсутствие ограничений на количество динамически добавляемых элементов (в реальности ограничено только объемом доступной памяти). Также значительно проще, чем в статических массивах, выполняется процедура удаления.

Недостатком связанных списков является отсутствие возможности

произвольного доступа к узлам, т.е. необходимость просмотра всех узлов, предшествующих заданному. Поэтому, в общем случае, работа со списками медленнее, чем с массивами. Так же, для хранения указателей на соседние элементы расходуется дополнительное количество памяти.

Базы данных

База данных – это информационная модель, позволяющая упорядоченно хранить данные о группе объектов, обладающих одинаковым набором свойств.

Преимуществом использования баз данных является удобство и относительная простота работы с данными, т.к. компоненты среды Delphi предоставляют для этого достаточно полные и гибкие возможности. Множество операций над данными, таких как поиск и фильтрация, с их помощью описываются и выполняются быстрее.

Недостатком баз данных является более низкая скорость работы, т.к. физически база данных представляет собой совокупность файлов, хранимых на внешнем носителе. Поэтому использование базы данных требует большего объема памяти, в том числе и для хранения служебной информации.

В условиях поставленной задачи, нет необходимости в использовании базы данных, т.к. она эффективна лишь при достаточно большом объеме хранимой информации. В противном случае, использование базы данных приведет к существенному снижению производительности программы и неоправданным затратам памяти.

Таким образом, проанализировав вышеописанные структуры данных, можно сделать вывод, что в случае поставленной задачи оптимальным решением является использование динамических связанных списков.

С целью экономии памяти каждая запись списка содержит указатель только на следующий элемент списка, т.к. используемые алгоритмы позволяют решить задачу с помощью односвязных списков.

Среды Delphi

Для написания программы целесообразно использовать среду Delphi.

Процесс разработки в Delphi значительно упрощен. В первую очередь это относится к созданию интерфейса, на который уходит 80% времени разработки программы.

Изначально среда разработки Delphi была предназначена исключительно для разработки приложений Windows, затем реализован вариант платформ Linux, однако после выпуска в 2002 году Kylix 3 его разработка была прекращена, и вскоре было объявлено о поддержке Microsoft. NET, которая, в свою очередь, была прекращена с выходом Delphi 2007.

На сегодняшний день, наряду с поддержкой разработки 32 и 64- разрядных программ для Windows, реализована возможность создавать приложения для Apple Mac OS X (начиная с Embarcadero Delphi XE2), iOS (включая симулятор, начиная с XE4 посредством собственного компилятора), а также, в Delphi XE5, для Google Android(непосредственно исполняемые на ARM-процессоре).

Мощность и гибкость Delphi при работе с базами данных основана на низкоуровневом ядре - процессоре баз данных Borland Database Engine (BDE). BDE позволяет осуществлять доступ к данным как с использованием традиционного record-ориентированного (навигационного) подхода, так и с использованием set-ориентированного подхода, используемого в SQL-серверах баз данных [2].

В Delphi предусмотрены специальные наборы компонент, отвечающих за доступ к данным, и компонент, отображающих данные. Компоненты доступа к данным позволяют осуществлять соединения с БД, производить выборку, копирование данных, и т.п. Компоненты визуализации данных позволяют отображать данные виде таблиц, полей, списков. Отображаемые данные могут быть текстового, графического или произвольного формата.

Таблицы сохраняются в базе данных. Некоторые СУБД сохраняют базу данных в виде нескольких отдельных файлов, представляющих собой таблицы (в основном, все локальные СУБД), в то время как другие состоят из одного файла, который содержит в себе все таблицы и индексы (InterBase).

Использование среды Delphi позволяет написать программу с удобным интерфейсом. Кроме того, программы, написанные на Delphi, хорошо сочетаются с широко распространенными на сегодняшний день операционными системами Windows [3].

Основу Delphi составляет не только сам язык, но и RAD (Rapid Application Development) – среда быстрой разработки приложений. Благодаря визуальному программированию, а также достаточно большой библиотеке визуальных компонентов, Delphi позволяет создавать программы наиболее быстро и эффективно, принимая на себя основную работу, и оставляя программисту творческий процесс.

Возможность быстрого создания профессиональных приложений для Windows делает среду разработки Delphi востребованной во всех отраслях человеческой деятельности.

1.2 Постановка задачи

Разработать программное средство для управления базой данных с информацией об имеющемся ПО.

Входные данные:

Файл с информацией о дисках.

Элемент списка имеет структуру:

Type

PCatalog = ^Catalog;

Catalog = Record

Title: String; (название ПО)

Number: String; (количество дисков)

Subject : String; (категория ПО: прикладное, системное)

Next : PCatalog;

end;

В программе должны присутствовать процедуры:

1. Формирование динамического списка, хранящего информацию о дисках;

2. Вывод списка на экран;

3. Добавление элемента (диска) в список;

4. Удаление элемента (диска) из списка;

5. Поиск элемента списка по соответствующему полю;

6. Запись в файл и загрузка из файла.

7. Сортировка дисков по заданному критерию.

Выходные данные:

Файл с информацией о дисках.

**2. РАЗРАБОТКА АЛГОРИТМА**

На этапе создания программы необходимо определить последовательность действий, выполнение которые позволяет решать поставленную задачу, т. е. разработать алгоритм. Алгоритм — это точное предписание, определяющее процесс перехода от исходных данных к результату.

Алгоритм решения задачи может быть представлен в виде словесного описания или графически — в виде блок-схемы с использование специальных символов [2].

Представление алгоритма в виде блок-схемы позволяет уяснить последовательность действий, которые должны быть выполнены для решения задачи, убедиться в правильности понимания поставленной задачи.

При программировании в Delphi алгоритм решения задачи представляет собой совокупность алгоритмов процедур обработки событий.

При разработке представленной программы значительное внимание было уделено интерфейсу. Чтобы предоставить пользователю возможность быстрого доступа к данным и комфортного их использования, в программе было задействовано опимальное количество элементов, на которые и возложено все управление программой.

В частности такими элементами является таблица StringGrid, в которую выводится список существующих дисков и информация о них, вторая форма для добавления информации с компонентами Edit, куда вносятся данные о дисках, панель с несколькими кнопками добавления, удаления и вставки элементов в таблицу. А так же панели поиска и сортировки элементов в таблице. Все это позволило сделать программу наглядной без ущерба ее функциональности [3].

Согласно поставленной задаче алгоритм предусматривает основные возможности работы с данными дисков ПО:

- добавление элемента (диска) в список (категорию, наименование, количество);

- удаление элемента (диска) из списка;

- вставка новой строки в список;

- очистка всей таблицы;

- поиск элемента списка по соответствующему полю;

- сортировка дисков по заданному критерию.

- сохранение списка дисков в текстовый файл и загрузка из файла.

Блок-схема программы представлена на рисунке 2.1.

После разработки алгоритмов обработки событий можно приступить к написанию программы.

Начало

Выход

Ввод данных

Сохранение списка в файл

Обработка элементов

Добавить

Удалить

Вставить

Очистить

Поиск элементов по критерию

По Наименованию

По Количеству

Нет

Да

Сортировка элементов по критерию

По Наименованию

По Количеству

Нет

Да

Нет

Да

Рисунок 4.1. Блок-схема программы Базы Данных ПО на дисках.

**3. РАЗРАБОТКА ПРОГРАММНОГО СРЕДСТВА**

3.1. Общие понятия разработки программного средства

Программное средство, программное обеспечение (software) – полный набор (программное обеспечение) или часть (программные средства) программ, процедур, правил и связанной с ними документации системы обработки информации. Программное средство – ограниченная часть программного обеспечения системы обработки информации, имеющая определенное функциональное назначение.

Программный модуль (software unit) – отдельно компилируемая часть программного кода (программы) [4].

        Разработка программ – сложный процесс, основной целью которого является создание, сопровождение программного кода, обеспечивающего необходимый уровень надежности и качества. Для достижения основной цели разработки программ используются средства разработки программного обеспечения.

3.2. Требования к программной реализации

1. Все переменные и константы должны иметь осмысленные имена в рамках тематики варианта к курсовой работе.

2. Имена процедур и функций должны быть осмысленными.

3. Код не должен содержать неименованных числовых констант (так называемых «магических» чисел), неименованных строковых констант (например, имен файлов и др.). Подобного рода информацию следует выносить в глобальные переменные с атрибутом const. По правилам хорошего стиля программирования тексты всех информационных сообщений, выводимых пользователю в ответ на его действия, также оформляются как константы.

4. Код необходимо комментировать (как минимум в части объявления структур, массивов/векторов, заголовков функций, нетривиальной логики).

5. Код не должен дублироваться – для этого существуют процедуры и функции.

6. Одна процедура решает только одну задачу (например, не допускается в одной процедуре считывать и выводить данные – это две разные процедуры). При этом внутри процедуры возможен вызов других процедур или функций.

7. Следует избегать длинных процедур и глубокой вложенности [1].

3.3. Описание основных процедур и функций модулей программы

Модуль Unit1

Методы:

**procedure** TFrmMain.FormCreate(Sender: TObject) - создание главной формы программы, на которой будут расположены компоненты программы.

**procedure** TFrmMain.btnAddClick(Sender: TObject) - появляется вторая форма при нажатии на кнопку «Добавить» и поля ввода на этой второй форме очищены.

**procedure** SGInsertRow(SG: TStringGrid; NewRow: Integer) - процедура вставки пустой строки. Используется в дальнейшем для процедуры. TFrmMain.btnInsertClick

**procedure** TFrmMain.btnInsertClick(Sender: TObject) – вставка пустой строки при нажатии в меню кнопки «Вставить». В таблице появляется новая строка ниже выделенной строки (куда поставлен курсор).

**procedure** SGDeleteRow(RowNumber: integer) – процедура удаления строки. Используется в дальнейшем для процедуры TFrmMain.btnDeleteClick.

**procedure** TFrmMain.btnDeleteClick(Sender: TObject) - удаляется выделенная строка при нажатии на кнопку «Удалить».

**procedure** PrintSearch(Searching: string; T: integer) - процедура печати искомой строки. Используется в дальнейшем для двух процедур: TFrmMain.BtnTitleSearchClick и TFrmMain.BtnNumberSearchClick

**procedure** TFrmMain.BtnTitleSearchClick(Sender: TObject) - поиск элементов списка по наименованию дисков.

**procedure** TFrmMain.BtnNumberSearchClick(Sender: TObject) - поиск элементов списка по количеству дисков.

**procedure** TFrmMain.EdtTitleSearchClick(Sender: TObject) - очищение строк поиска для ввода строки при нажатии на поле ввода строки.

**procedure** TFrmMain.EdtNumberSearchClick(Sender: TObject) - очищение строк поиска для ввода строки при нажатии на поле ввода строки.

**procedure** TFrmMain.Sort(SortType: integer); процедура сортировки. Используется в дальнейшем для процедуры TFrmMain.RGClick

**procedure** TFrmMain.RGClick(Sender: TObject) - при нажатии на один из вариантов сортировки в компоненте радиогрупп.

**procedure** TFrmMain.FormClose(Sender: TObject; var Action: TCloseAction) - при закрытии проекта (при нажатии на крестик в правом верхнем углу формы) появляется выпадающая форма с вопросом «Закрыть проект?» и двумя вариантами выбора. При нажатии на «ОК» - программа будет закрыта. При нажатии на «Cancel» - программа продолжит свою работу.

Модуль Unit2

Методы:

**procedure** DiskAdd(H: PCatalog) - процедура добавления диска в список. Используется в дальнейшем для процедуры TFrmNewDisk.BtnAddNewDiskClick.

**procedure** TFrmNewDisk.BtnAddNewDiskClick(Sender: TObject) - добавление новой строки в таблицу главной формы при нажатии на кнопку "Добавить" на второй форме.

**4. ОБОСНОВАНИЕ ТЕХНИЧЕСКИХ ПРИЕМОВ ПРОГРАММИРОВАНИЯ**

# 4.1. Критерии хорошей архитектуры программы

При разработке программы, важно чтобы программа не только хорошо работала, но и была хорошо организована. Сложность, как правило, растет гораздо быстрее размеров программы. И если не позаботиться об этом заранее, то довольно быстро наступает момент, когда программист перестает эту сложность контролировать [5].

Общепринятого термина «архитектура программного обеспечения» не существует. Тем не менее, хорошая архитектура это прежде всего выгодная архитектура, делающая процесс разработки и сопровождения программы более простым и эффективным. Программу с хорошей архитектурой легче расширять и изменять, а также тестировать, отлаживать и понимать. То есть, можно сформулировать список универсальных критериев:  
  
**- Эффективность системы**. В первую очередь программа должна решать поставленные задачи и хорошо выполнять свои функции. Сюда можно отнести такие характеристики, как надежность, безопасность, производительность, способность справляться с увеличением нагрузки (масштабируемость) и прочие.   
  
**- Гибкость системы.** Любое приложение приходится менять со временем — изменяются требования, добавляются новые. Чем быстрее и удобнее можно внести изменения в существующий функционал, чем меньше ошибок это вызовет. Поэтому в процессе разработки следует оценивать программу на предмет дальнейших возможных ее изменений.  
**- Расширяемость системы**. Возможность добавлять в систему новые сущности и функции, не нарушая ее основной структуры. На начальном этапе в систему имеет смысл закладывать лишь основной и самый необходимый функционал. Но при этом архитектура должна позволять легко наращивать дополнительный функционал по мере необходимости так, чтобы внесение наиболее вероятных изменений требовало меньших усилии.   
**- Масштабируемость процесса разработки**. Возможность сократить срок разработки за счёт добавления к проекту новых людей. Архитектура должна позволять распараллелить процесс разработки, так чтобы множество людей могли работать над программой одновременно.  
- **Тестируемость**. Код, который легче тестировать, будет содержать меньше ошибок и надежнее работать. Существует целая методология разработки программ на основе тестов, которая так и называется — Разработка через тестирование (Test-Driven Development, TDD).  
**- Возможность повторного использования**. Систему желательно проектировать так, чтобы ее фрагменты можно было повторно использовать в других системах.  
**- Хорошо структурированный, читаемый и понятный код. Сопровождаемость**. Над программой, как правило, работает множество людей. После написания сопровождать программу тоже, как правило, приходится людям, не участвовавшем в ее разработке. Поэтому хорошая архитектура должна давать возможность легко и быстро разобраться в системе. Проект должен быть хорошо структурирован, не содержать дублирования, иметь хорошо оформленный код и документацию.

# 4.2. Декомпозиция как основа модульной архитектуры.

# Главным критерием при разработке больших систем считается задача снижения сложности. И лучшее решение – это деление на части, то есть **иерархическая декомпозиция [6]**. Сложная система должна строится из небольшого количества более простых подсистем, каждая из которых, в свою очередь, строится из частей меньшего размера, и т.д., до тех пор, пока самые небольшие части не будут достаточно просты для непосредственного понимания и создания (рис 4.1).

# 

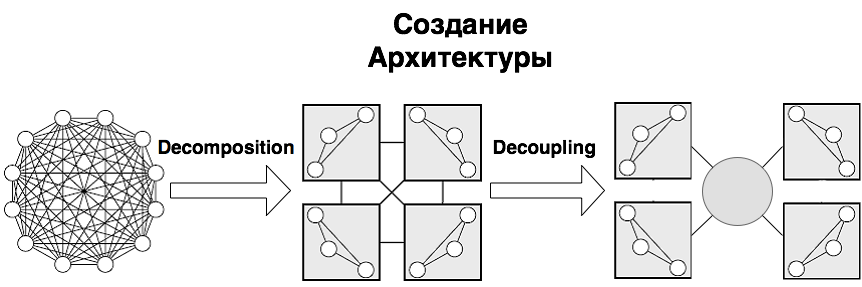


Рисунок 4.1. Процесс создания архитектуры программы (декомпозиции).

Данное решение является не только единственно известным, но и универсальным. Помимо снижения сложности, оно одновременно обеспечивает гибкость системы, дает хорошие возможности для масштабирования, а также позволяет повышать устойчивость за счет дублирования критически важных частей.  
  
Соответственно, при построении архитектуры программы, создании ее структуры, под этим подразумевается декомпозиция программы на подсистемы (функциональные модули, сервисы, слои, подпрограммы) и организация их взаимодействия друг с другом и внешним миром. Причем, чем более независимы подсистемы, тем безопаснее сосредоточиться на разработке каждой из них в отдельности в конкретный момент времени.

Преимущества данного подхода создания архитектуры програмы:

**- Масштабируемость (Scalability)** возможность расширять систему и увеличивать ее производительность, за счет добавления новых модулей.

- **Ремонтопригодность (Maintainability)** изменение одного модуля не требует изменения других модулей.

- **Заменимость модулей (Swappability)** модуль легко заменить на другой.

-**Возможность тестирования (Unit Testing)** модуль можно отсоединить от всех остальных и протестировать / починить.

- **Переиспользование (Reusability)** модуль может быть переиспользован в других программах и другом окружении.

- **Сопровождаемость (Maintenance)** разбитую на модули программу легче понимать и сопровождать

Можно сказать, что архитектура идентифицирует главные компоненты системы и способы их взаимодействия. Также это выбор таких решений, которые интерпретируются как основополагающие и не подлежащие изменению в будущем.  
  
 Архитектура — это **организация системы**, воплощенная в ее **компонентах**, их **отношениях** между собой и с **окружением**.

**Система** — это набор компонентов, объединенных для выполнения определенной функции.

4.3. Технические приемы разработки данной программы в среде Delphi.

При разработке данной программы мною были применены следующие технические приемы [5]:

1) При разработке интерфейса программы использованы компоненты: TLabel, TEdit, TCombobox, TButton, TRadioGroup, TStringGrid, TPanel.

2) В программе используется один динамический список для хранения данных.

3) Для хранения указателя на начало списка типа PCatalog используется глобальная переменная, чтобы доступ к списку был обеспечен в любой точке программы.

4) для экономии памяти определена процедура для физического удаления списков.

**5. ТЕСТИРОВАНИЕ, ЭКСПЕРИМЕНТАЛЬНЫЕ ИССЛЕДОВАНИЯ И АНАЛИЗ ПОЛУЧЕННЫХ РЕЗУЛЬТАТОВ**

При создании программного средства очень важно не только его разработать, но и проверить на работоспособность, то есть протестировать. Это дает повышение качества программного продукта по многим критериям. Рассмотрим что такое процесс тестирования программы, этапы тестирования, критерии, градация и другие ее аспекты [6].

5.1. Основные определения, этапы и принципы тестирования

Тестирование программного обеспечения — проверка соответствия между реальным и ожидаемым поведением программы, осуществляемая на конечном наборе тестов, выбранном определенным образом.   
 Качество программного обеспечения (Software Quality) — это совокупность характеристик программного обеспечения, относящихся к его способности удовлетворять установленные и предполагаемые потребности.   
 Верификация (verification) — это процесс оценки системы или её компонентов с целью определения удовлетворяют ли результаты текущего этапа разработки условиям, сформированным в начале этого этапа[IEEE]: выполняются ли цели, сроки, задачи по разработке проекта.  
 Валидация (validation) — это определение соответствия разрабатываемого ПО ожиданиям и потребностям пользователя, требованиям к системе.

Целью тестирвоания является повышение вероятности того, что разработанная программа будет работать правильно при любых обстоятельствах [7].

Этапы тестирования:

1. Анализ  
2. Разработка стратегии тестирования и планирование процедур контроля качества  
3. Работа с требованиями  
4. Создание тестовой документации  
5. Тестирование прототипа  
6. Основное тестирование  
7. Стабилизация  
8. Эксплуатация

Тест план (Test Plan) — это документ, описывающий весь объем работ по тестированию, начиная с описания объекта, стратегии, расписания, критериев начала и окончания тестирования, до необходимого в процессе работы оборудования, специальных знаний, а также оценки рисков с вариантами их разрешения.   
Отвечает на вопросы:  
- Что надо тестировать?  
- Как будете тестировать?  
- Когда будете тестировать?  
- Критерии начала и окончания тестирования.

Каждый тест кейс должен иметь 3 части:

- Список действий, которые приводят систему к состоянию пригодному для проведения основной проверки. Либо список условий, выполнение которых говорит о том, что система находится в пригодном для проведения основного теста состояния.  
- Список действий, переводящих систему из одного состояния в другое, для получения результата, на основании которого можно сделать вывод о удовлетворении реализации, поставленным требованиям  
- Список действий, переводящих систему в первоначальное состояние.

Дефект (он же баг) — это несоответствие фактического результата выполнения программы ожидаемому результату. Дефекты обнаруживаются на этапе тестирования программного обеспечения (ПО), когда тестировщик проводит сравнение полученных результатов работы программы (компонента или дизайна) с ожидаемым результатом, описанным в спецификации требований [6].

Error — ошибка пользователя, то есть он пытается использовать программу иным способом.

Пример — вводит буквы в поля, где требуется вводить цифры (возраст, количество товара и т.п.).

В качественной программе предусмотрены такие ситуации и выдаются сообщение об ошибке (error message), с красным крестиком которые.  
Градация Серьезности дефекта

- Блокирующая ошибка, приводящая приложение в нерабочее состояние, в результате которого дальнейшая работа с тестируемой системой или ее ключевыми функциями становится невозможна. Решение проблемы необходимо для дальнейшего функционирования системы.  
- Критическая ошибка, неправильно работающая ключевая бизнес логика, дыра в системе безопасности, проблема, приведшая к временному падению сервера или приводящая в нерабочее состояние некоторую часть системы, без возможности решения проблемы, используя другие входные точки. Решение проблемы необходимо для дальнейшей работы с ключевыми функциями тестируемой системой.  
- Значительная ошибка, часть основной бизнес логики работает некорректно. Ошибка не критична или есть возможность для работы с тестируемой функцией, используя другие входные точки.  
- Незначительная ошибка, не нарушающая бизнес логику тестируемой части приложения, очевидная проблема пользовательского интерфейса.  
- Тривиальная ошибка, не касающаяся бизнес логики приложения, плохо воспроизводимая проблема, малозаметная посредствам пользовательского интерфейса, проблема сторонних библиотек или сервисов, проблема, не оказывающая никакого влияния на общее качество продукта.

Градация Приоритета дефекта

- Высокий. Ошибка должна быть исправлена как можно быстрее, т.к. ее наличие является критической для проекта.  
- Средний. Ошибка должна быть исправлена, ее наличие не является критичной, но требует обязательного решения.  
- Низкий. Ошибка должна быть исправлена, ее наличие не является критичной, и не требует срочного решения.

Принципы тестирования

1) Тестирование демонстрирует наличие дефектов и может показать, что дефекты присутствуют, но не может доказать, что их нет. Тестирование снижает вероятность наличия дефектов, находящихся в программном обеспечении, но, даже если дефекты не были обнаружены, это не доказывает его корректности.

2) Исчерпывающее тестирование недостижимо. Полное тестирование с использованием всех комбинаций вводов и предусловий физически невыполнимо, за исключением тривиальных случаев. Вместо исчерпывающего тестирования должны использоваться анализ рисков и расстановка приоритетов, чтобы более точно сфокусировать усилия по тестированию.  
3) Раннее тестирование.Чтобы найти дефекты как можно раньше, активности по тестированию должны быть начаты как можно раньше в жизненном цикле разработки программного обеспечения или системы, и должны быть сфокусированы на определенных целях.

4) Скопление дефектов.Усилия тестирования должны быть сосредоточены пропорционально ожидаемой, а позже реальной плотности дефектов по модулям. Как правило, большая часть дефектов, обнаруженных при тестировании или повлекших за собой основное количество сбоев системы, содержится в небольшом количестве модулей.

5) Парадокс пестицида. Если одни и те же тесты будут прогоняться много раз, в конечном счете этот набор тестовых сценариев больше не будет находить новых дефектов. Чтобы преодолеть этот «парадокс пестицида», тестовые сценарии должны регулярно рецензироваться и корректироваться, новые тесты должны быть разносторонними, чтобы охватить все компоненты программного обеспечения, или системы, и найти как можно больше дефектов.  
6) Тестирование зависит от контекста. Тестирование выполняется по-разному в зависимости от контекста. Например, программное обеспечение, в котором критически важна безопасность, тестируется иначе, чем сайт электронной коммерции.   
7) Заблуждение об отсутствии ошибок. Обнаружение и исправление дефектов не помогут, если созданная система не подходит пользователю и не удовлетворяет его ожиданиям и потребностям.

Виды / типы тестирования

- Функциональное: функциональное тестирование, тестирование безопасности и тестирование взаимодействия.

- Нефункциональные виды тестирования: все виды тестирования производительности (нагрузочное, тестирование стабильности или надежности, объемное), тестирование установки, тестирование удобства пользования, тестирование на отказ и восстановление, конфигурационное.  
 Функциональное тестирование рассматривает заранее указанное поведение и основывается на анализе спецификаций функциональности компонента или системы в целом.

Тестирование безопасности — это стратегия тестирования, используемая для проверки безопасности системы, а также для анализа рисков, связанных с обеспечением целостного подхода к защите приложения, атак хакеров, вирусов, несанкционированного доступа к конфиденциальным данным.  
 Тестирование взаимодействия — это функциональное тестирование, проверяющее способность приложения взаимодействовать с одним и более компонентами или системами и включающее в себя тестирование совместимости и интеграционное тестирование.

Нагрузочное тестирование — это автоматизированное тестирование, имитирующее работу определенного количества бизнес пользователей на каком-либо общем (разделяемом ими) ресурсе.

Стрессовое тестирование позволяет проверить насколько приложение и система в целом работоспособны в условиях стресса и также оценить способность системы к регенерации, т.е. к возвращению к нормальному состоянию после прекращения воздействия стресса. Стрессом в данном контексте может быть повышение интенсивности выполнения операций до очень высоких значений или аварийное изменение конфигурации сервера. Также одной из задач при стрессовом тестировании может быть оценка деградации производительности, таким образом цели стрессового тестирования могут пересекаться с целями тестирования производительности.

Тестирование удобства пользования — это метод тестирования, направленный на установление степени удобства использования, обучаемости, понятности и привлекательности для пользователей разрабатываемого продукта в контексте заданных условий. Сюда также входит:  
 Тестирование пользовательского интерфейса — это вид тестирования исследования, выполняемого с целью определения, удобен ли некоторый искусственный объект для его предполагаемого применения.

Предугадывание ошибки – это когда тест аналитик использует свои знания системы и способность к интерпретации спецификации на предмет того, чтобы «предугадать» при каких входных условиях система может выдать ошибку. Например, спецификация говорит: «пользователь должен ввести код». Тест аналитик, будет думать: «Что, если я не введу код?», «Что, если я введу неправильный код? », и так далее. Это и есть предугадывание ошибки.

Статическое и динамическое тестирование

Статическое тестирование отличается от динамического тем, что производится без запуска программного кода продукта. Тестирование осуществляется путем анализа программного кода (code review) или скомпилированного кода. Анализ может производиться как вручную, так и с помощью специальных инструментальных средств. Целью анализа является раннее выявление ошибок и потенциальных проблем в продукте. Также к статическому тестирвоанию относится тестирования спецификации и прочей документации.

5.2. Тестирование программы

В результате проведенного тестирования программы, я получила следующие результаты, представленные в таблице 5.1

Таблица 5.1 - Результаты тестов подпрограмм

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Название подпрограммы | Входные данные | Ожидаемые результаты | Пример вызова | Полученные результаты |
| AddNewDisk | Принимает введенные пользователем в поля ввода данные о диске. | Чтение данных и занесение в список данных о новом диске. | Содержимое полей ввода Системное ПО; UnixWar; 2 | Данные из полей ввода прочитаны, занесены в список  новый диск, содержащий введенные данные. |
| DeleteDisk | Принимает данные о диске из полей выделенной строки таблицы. | Поиск и удаление из списка диска с заданными данными. | Содержимое полей строки таблицы: Прикладное ПО; Auto CAD; 8 | Был найден и удален из списка диск с заданными данными. |
| TitleSearch | Принимает введенное пользователем в поле ввода наименование диска. | Чтение введенных данных, вывод на экран найденных по данному запросу дисков. | Содержимое поля ввода:  MS Outlook Express | Данные прочитаны, поиск диска с наименованием, содержащим введенное, на экран выведен диск |
| Sort | Принимает значение 0 или 1 (в процедуре RgClick) и в зависимости от этого сортирует по наименованию или по количеству | Сортирует список дисков по наименованию в алфавитном порядке, либо по количеству в порядке возрастани |  | Список успешно отсортировался. |

**6. РУКОВОДСТВО ПОЛЬЗОВАТЕЛЯ СИСТЕМЫ**

При запуске программы появляется главное окно (рис 6.1). На главном окне можно видеть таблицу, а также операции, выполняемые над её элементами. В верхнем меню находятся операции «Загрузить», «Сохранить» и «Очистить».

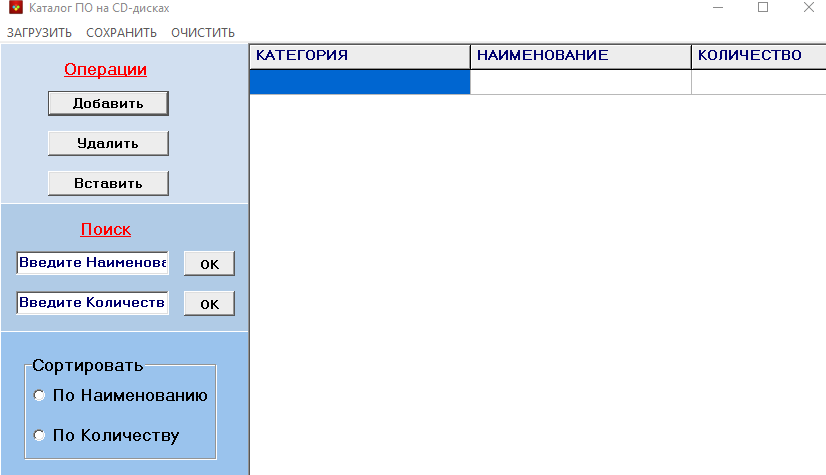


Рисунок 6.1. Главное окно программы.

Загрузка из файла происходит при нажатии кнопки «Загрузить» в вехнем меню. По нажатию появляется диалоговое окно «Загрузить файл» с выбором файлов. Для упрощения поиска нужного файла, можно в окне «Тип файла» выбрать тип либо dat, либо txt. Тогда появляются файлы с данным расширением соответственно (рис 6.2).

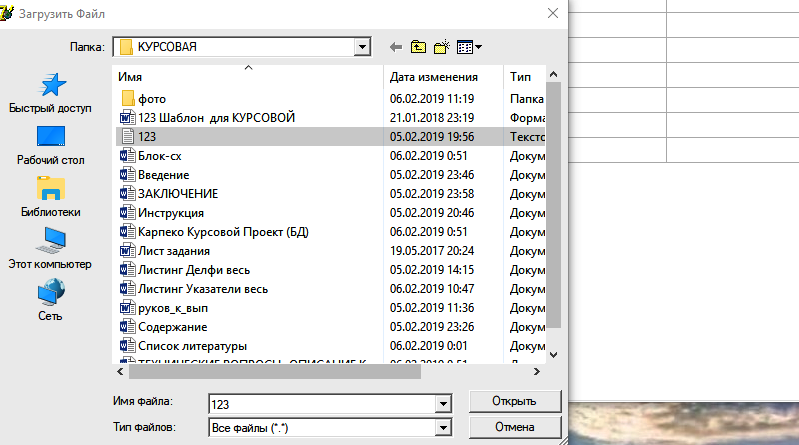


Рисунок 6.2. Загрузка файла.

Очищение всей таблицы от записей происходит при нажатии на кнопку «Очистить» в верхнем меню. Все строки таблицы будут очищены (рис 6.3).

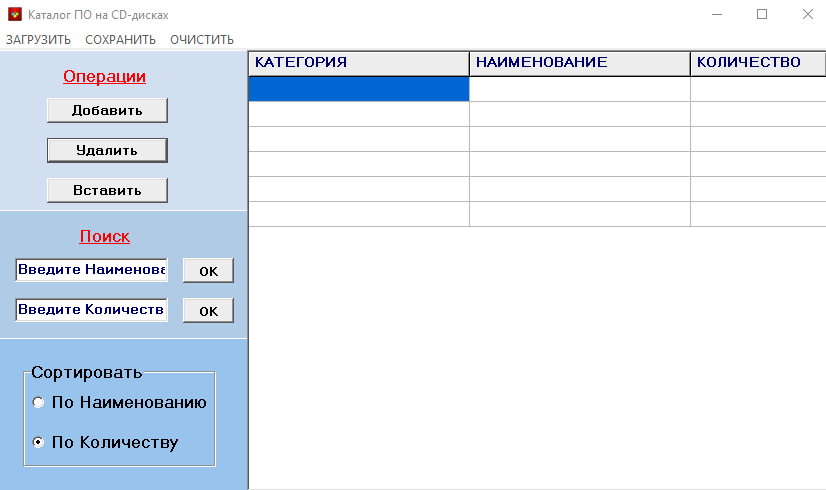


Рисунок 6.3. Очищение таблицы от записей.

Добавление новой строки можно производить непосредственно в самой таблице, либо с помощью второй формы, которая вызывается нажатием на кнопку «Добавить» в боковом меню программы (рис 6.4, 6.5).

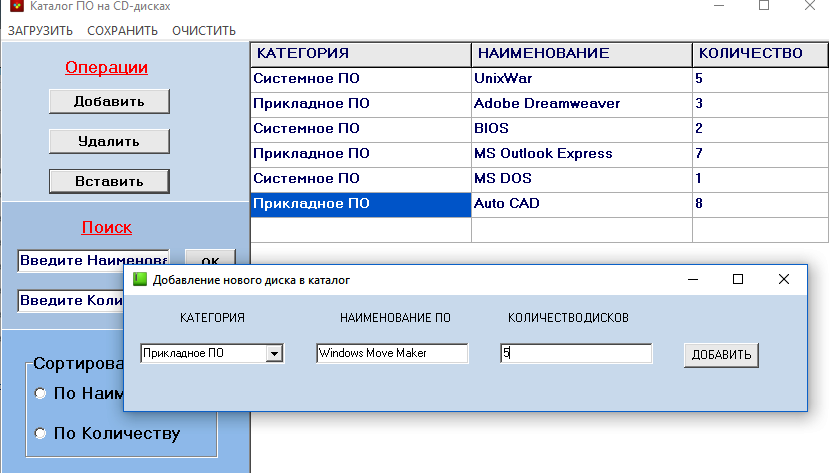


Рисунок 6.4. Вторая форма для добавления строки в таблицу.

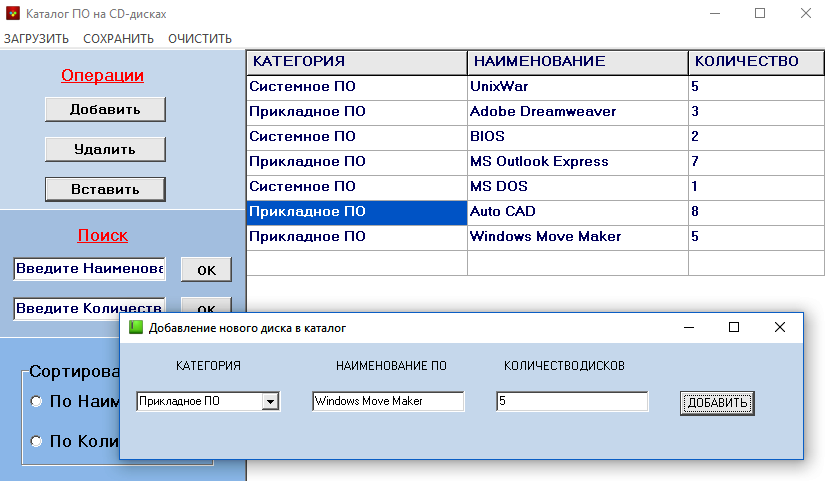


Рисунок 6.5. Результат добавления новой строки.

В таблицу можно вставлять пустые строки ниже выделенной строки (рис 6.6).

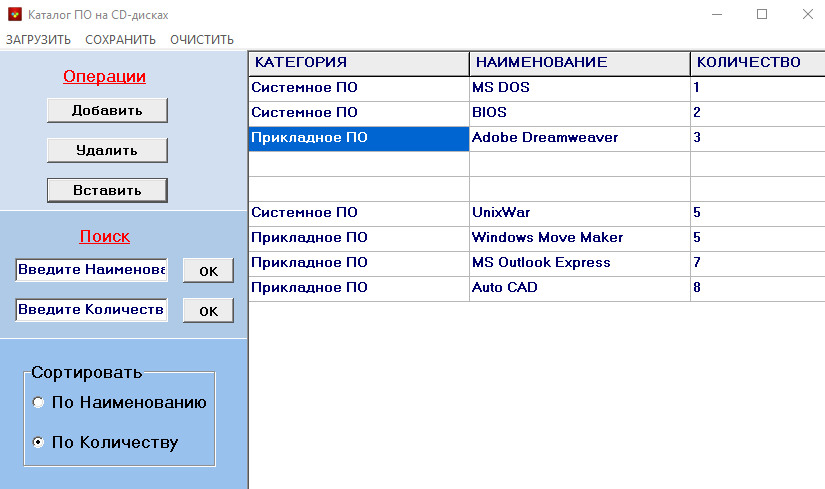


Рисунок 6.6. Вставка пустой строки ниже выделенной

Сортировка происходит либо по наименованию дисков (рис 6.7), либо по их количеству (рис 6.8).

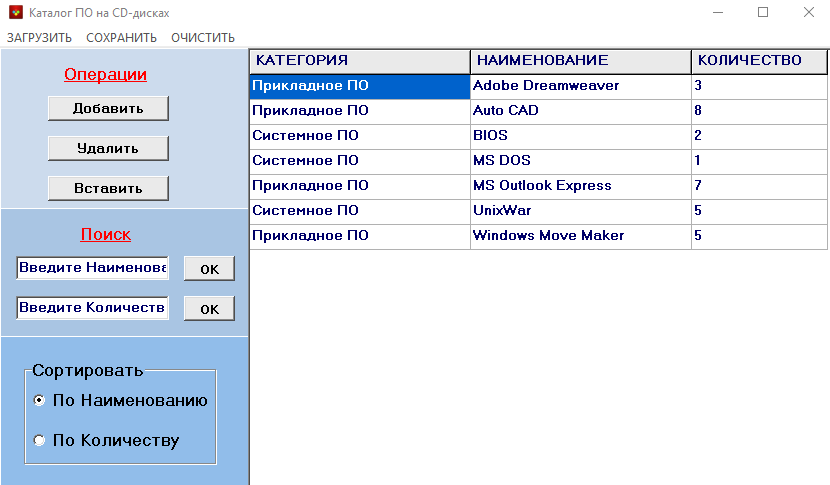


Рисунок 6.7. Сортировка по наименованию диска

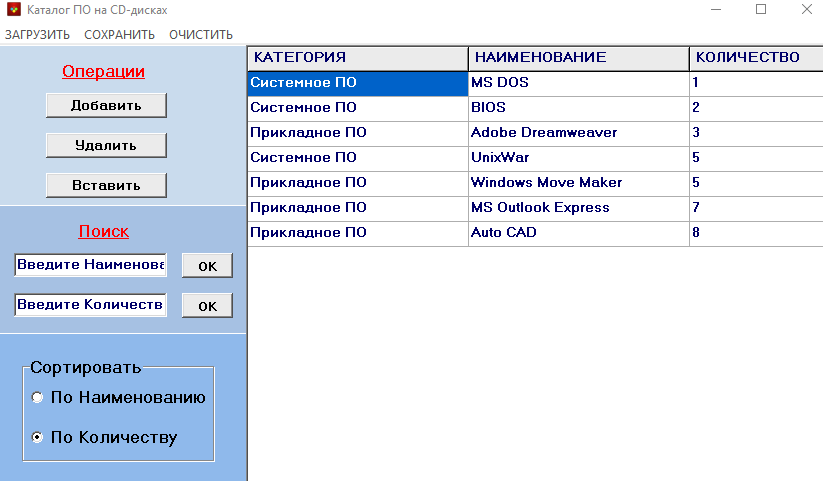


Рисунок 6.8. Сортировка по количеству дисков

При нажатии на кнопку «Удалить» в группе «Операции» удаляется выделенная строка (рис 6.9).

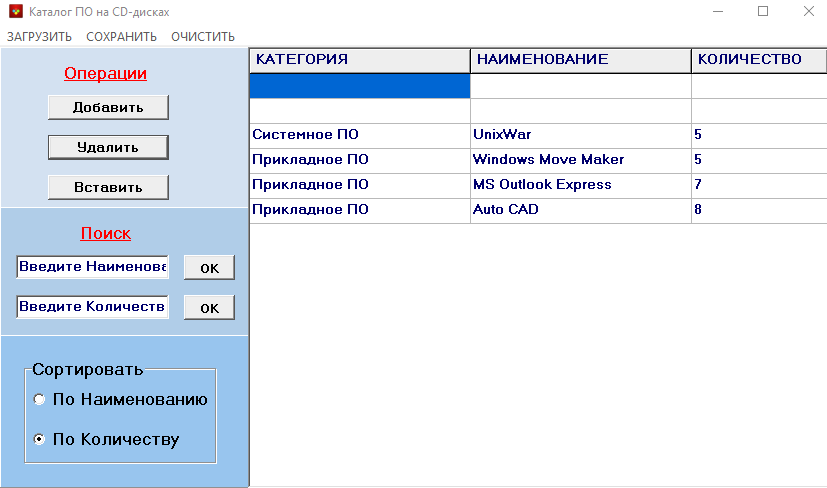


Рисунок 6.9. Удаление строки таблицы.

Для сохранения данных в файл – нажать на кнопку «Сохранить» в верхнем меню. Появляется диалоговое окно, где можно выбрать место сохранения файла, его наименование и обязательно указать тип сохраняемого файла (рис 6.10).

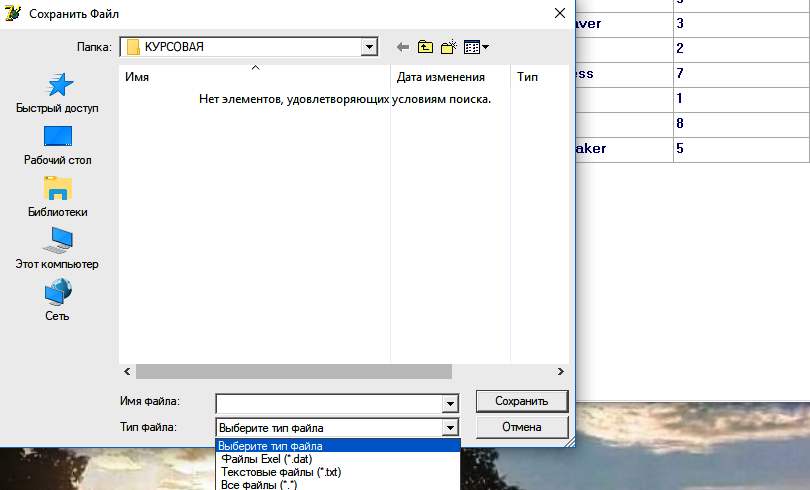


Рисунок 6.10. Сохранение таблицы в файл.

По окончании работы с программой и нажатии на верхний правый крестик окна программы, появяется форма с вопросом о завершении программы. При нажатии на «ОК» - программа будет закрыта. При нажатии на «Cancel» - программа продолжит свою работу (рис 6.11).

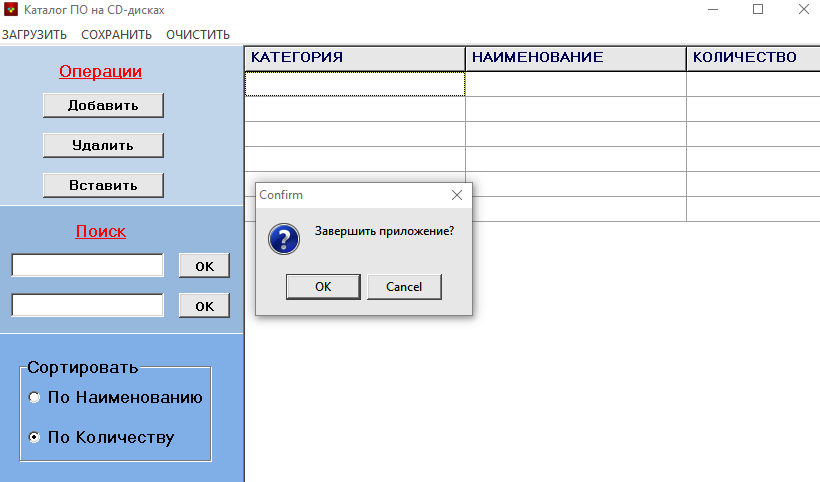


Рисунок 6.11. Диалоговое окно с выбором продолжения или прекращения работы программы.

# ЗАКЛЮЧЕНИЕ

Результаты следует излагать в форме констатации фактов, используя слова: **«изучены», «исследованы», «сформулированы», «показано», «разработана», «предложена», «подготовлены», «изготовлена», «испытана» и т.п.**

Данный курсовой проект был разработан и написан в среде программирования Delphi, программа, позволяющая управлять базой данных об имеющемся Программном Обеспечении: добавление, удаление записей, их поиск и сортировку.

Для решения задачи использованы структуры данных (динамические связанные списки), выбор которых обоснован в пункте 1. Аналитический обзор литературы. Созданный алгоритм в среде программирования Delphi, эффективно использует предоставляемые ею возможности.

Таким образом, в процессе разработки данного программного средства я на практике убедилась в преимуществах работы с динамическими структурами данных.

**СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ**

[1]. Культин Н.Б. «Основы программирования в Delphi 7».: Издательство BHV СПБ. 2008г. - 480 с. –ISBN 978-5-9775-0235-1

[2]. С.И. Бобровский «Delphi 7 Учебный курс».: Издательство «Питер» 2004. - 736 с. – ISBN 5-8046-0086-9

[3]. Г.В. Галисеев «Компоненты в Delphi 7».: Издательство «Вильямс» 2004г. - 624стр. –ISBN 5-8459-0555-9

[4]. А. Шкрыль «Delphi. **Народные советы».:** Издательство BHV-СПБ 2007г. – 400стр. – ISBN 5-9775-0047-5

[5]. Архангельский А.Я. «Приемы программирования в Delphi».: Издательство «Бином-Пресс» 2006г. -944 стр. –ISBN 5-9518-0145-1

[6]. В.Кадлец «Delphi. Книга рецептов. Практические примеры, трюки и секреты».: Издательство «Наука и техника». 2006г. – 384стр. – ISBN 80-251-0017-0

[7]. Фаронов В.В. «Delphi 2005. Разработка приложений для баз данных и Интернета».: Издательство «Питер». 2006г. – 608стр. – ISBN 5-469-01191-7

**ЛИСТИНГ ПРОГРАММЫ С ИСПОЛЬЗОВАНИЕМ ДИНАМИЧЕСКИХ ПЕРЕМЕННЫХ**

**unit** Unit1;

**interface**

**uses**

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Grids, ExtCtrls, Buttons, Menus;

**type**

PCatalog = ^Catalog;

Catalog = **Record**

Subject : String[50]; {Subject - категория ПО (прикладное ПО,

системное);}

Title: String[50]; {Title - название ПО;}

Number: String[50]; {Number - количество дисков;}

Next : PCatalog; {Next - указатель на следующий элемент списка.}

**end**;

TFrmMain = **class**(TForm)

pnlOperations: TPanel;

pnlSearch: TPanel;

pnlSorting: TPanel;

lblOperations: TLabel;

btnAdd: TButton;

btnDelete: TButton;

lblSearch: TLabel;

EdtTitleSearch: TEdit;

EdtNumberSearch: TEdit;

BtnTitleSearch: TButton;

BtnNumberSearch: TButton;

RG: TRadioGroup;

OpenDialog1: TOpenDialog;

SaveDialog1: TSaveDialog;

SG: TStringGrid;

btnInsert: TButton;

**procedure** FormCreate(Sender: TObject);

**procedure** mniSaveClick(Sender: TObject);

**procedure** mniLoadClick(Sender: TObject);

**procedure** N1Click(Sender: TObject);

**procedure** btnAddClick(Sender: TObject);

**procedure** btnInsertClick(Sender: TObject);

**procedure** btnDeleteClick(Sender: TObject);

**procedure** BtnTitleSearchClick(Sender: TObject);

**procedure** BtnNumberSearchClick(Sender: TObject);

**procedure** EdtTitleSearchClick(Sender: TObject);

**procedure** EdtNumberSearchClick(Sender: TObject);

**procedure** RGClick(Sender: TObject);

**procedure** FormClose(Sender: TObject; **var** Action: TCloseAction);

**private**

{ Private declarations }

**public**

{ Public declarations }

**end**;

**var**

FrmMain: TFrmMain;

Head: PCatalog;

IsAdded: Boolean;

**implementation**

**uses** u\_f2;

{$R \*.dfm}

//-------------------------------------------------------------------------

//1 ==== создание главной формы ======================

**procedure** TFrmMain.FormCreate(Sender: TObject);

**var**

F: TextFile;

C: char;

S: string;

H: PCatalog;

i: integer; // i - номер печатающейся строки

**begin**

**{**присваиваем фиксированной строке таблицы имена, устанавливаем

ширину столбцов}

Sg.ColWidths[0] := 220;

Sg.ColWidths[1] := 220;

Sg.ColWidths[2] := 135;

Sg.Cells[0,0] := ' КАТЕГОРИЯ';

Sg.Cells[1,0] := ' НАИМЕНОВАНИЕ';

Sg.Cells[2,0] := ' КОЛИЧЕСТВО';

**if** FileExists('Data.txt') **then**  {если файл с данными есть}

**begin**

AssignFile(F, 'Data.txt'); {открываем файл F для чтения}

Reset(F);

i := 1; {i - номер печатающейся строки}

**while not** Eof(F) **do** {пока не конец файла}

**begin**

// если список пуст -----------

**if** Head = nil **then**

**begin**

**New**(Head); {создаём список, считыв данные о I диске из файла}

Read(F, C);

**case** strtoint(C) **of**

0: Head^.Subject := 'Прикладное ПО';

1: Head^.Subject := 'Системное ПО';

**end**;

Read(F, C, C);

S := '';

**while** C <> ' ' **do**

**begin**

S := S + C;

Read(F, C);

**end**;

Head^.Title := S;

S := '';

Read(F, C);

**while** C <> ' ' **do**

**begin**

S := S + C;

Read(F, C);

**end**;

Head^.Number := S;

Head^.Next := nil;

//==== вывод списка( I элемента) на экран ======

Sg.RowCount := Sg.RowCount + 1; {добавляем вначале пустую

cтроку}

Sg.Cells[0, 1] := Head^.Subject;

Sg.Cells[1, 1] := Head^.Title;

Sg.Cells[2, 1] := Head^.Number;

**End**

**else**

// если список не пуст ------------

**begin**

Readln(F); {считываем данные о диске из файла и добавляем в

cписок}

H := Head;

**while** H^.Next <> nil **do**

H := H^.Next;

**New**(H^.Next);

H := H^.Next;

Read(F, C);

**case** strtoint(C) **of**

0: Head^.Subject := 'Прикладное ПО';

1: Head^.Subject := 'Системное ПО';

**end**;

Read(F, C, C);

S := '';

**while** C <> ' ' **do**

**begin**

S := S + C;

Read(F, C);

**end**;

H^.Title := S;

S := '';

Read(F, C);

**while** C <> ' ' **do**

**begin**

S := S + C;

Read(F, C);

**end**;

H^.Number := S;

H^.Next := nil;

{вывод на экран следующего диска}

Sg.RowCount := Sg.RowCount + 1;

i := i + 1; {очередная строчка}

Sg.Cells[0, 1] := H^.Subject;

Sg.Cells[1, 1] := H^.Title;

Sg.Cells[2, 1] := H^.Number;

**end**;

**end**;

CloseFile(F); {закрываем файл}

**end**;

**end**; //---FormCreate

//2===== при закрытии проекта =====================

**procedure** TFrmMain.FormClose(Sender: TObject; **var** Action: TCloseAction);

**var**

F: TextFile;

**begin**

AssignFile(F, 'Data.txt'); {связываем переменные и файл}

**if** Head <> nil **then**

**begin** {создаём новый файл}

Rewrite(F); {откр для записи}

**while** Head <> nil **do** {проходимся по каждому элементу списка}

**begin** {записываем информацию в файл}

**if** Head^.Subject := 'Прикладное ПО' **then**

write(F, '0 ')

**else if** Head^.Subject := 'Системное ПО' **then**

write(F, '1 ');

write(F, Head^.Title, ' ');

**if** Head^.Next = nil **then** {если текущ диск - последний, то пишем в

файл последнее поле}

write(F, Head^.Number)

**else** {если текущ диск - не последний, то после записи последнего

поля (Number) в файл - переход на следующую строку файла}

writeln(F, Head^.Number);

Head := Head^.Next; {переход к следующему диску}

**end**;

CloseFile(F); {закрываем файл}

**end**;

**If** MessageDlg('Завершить приложение?', mtConfirmation, mbOkCancel, 0) = mrOk **Then**

Action := caFree {Нажата Ok - закрываем форму.}

**Else**

Action := caNone; {Нажата Cancel - игнорируем закрытие.}

**end**; //---FormClose

//3 ====== при нажатии на кнопку "Добавить" появляется вторая форма ==

**procedure** TFrmMain.btnAddClick(Sender: TObject);

**begin**

**with** FrmNewDisk **do**

**begin** {очищаем поля компонентов на второй форме}

cbbSubject.ItemIndex := 0; {меняем с -1 на первый вариант в

Combobox.ItemIndex}

EdtTitle.Text := '';

EdtNumber.Text := '';

IsAdded := true; {глобальная var IsAdded: Boolean;}

FrmNewDisk.Show; // открываем вторую форму

**end**;

**end**; //---btnAddClick

//4 ==== Процедура вставки пустой строки ======================

**procedure** SGInsertRow(SG: TStringGrid; NewRow: Integer);

**var**

i: Integer;

**begin**

**with** SG **do**

**begin**

RowCount := RowCount + 1;

**if** NewRow < RowCount - 1 **then**

**begin**

**for** i := RowCount - 1 **downto** NewRow + 1 **do**

Rows[i].Assign(Rows[i - 1]);

**end**;

Rows[NewRow].Clear;

**end**;

**end**; //---SGInsertRow

//6 ==== Вставка пустой строки ========================

**procedure** TFrmMain.btnInsertClick(Sender: TObject);

**begin**

SGInsertRow(SG, SG.Row + 1);

**end**; //---btnInsertClick

//7 ===== Удаление строки ===========================

**procedure** SGDeleteRow(RowNumber: integer);

**var**

i: integer;

**begin**

**with** FrmMain **do**

**begin**

Sg.Row := RowNumber;

**if** Sg.Row = Sg.RowCount-1 **then {**если удаляем последнюю строку

таблицы}

Sg.RowCount := Sg.RowCount-1

**else**

**begin** {если строка - не последняя}

**for** i := RowNumber **to** Sg.RowCount-1 **do**

Sg.Rows[i] := Sg.Rows[i + 1];

Sg.RowCount := Sg.RowCount-1;

**end**;

**end**;

**end**; //---SGDeleteRow

//8 ===== при нажатии на кнопку "Удалить" ===============

**procedure** TFrmMain.btnDeleteClick(Sender: TObject);

**var**

H, Temp: PDisk;

**begin**

H := Head;

**if** Sg.Row <> 0 **then** {если выделена не шапка таблицы}

**begin** {если данные I элемента в списке совпадают с данными диска

выделенной строки таблицы, то удаляем I эл-т из списка и чистим память}

**if** (H^.Subject = Sg.Cells[0, Sg.Row]) **and** (H^.Title = Sg.Cells[1,Sg.Row]) **and** (H^. Number = Sg.Cells[2, Sg.Row]) **then**

**begin**

Temp := Head;

Head := Head^.Next;

Dispose(Temp);

**End**

**else** {ищем совпадения данных текущего элемента с данными в

выделенной строке таблицы, потом удаляем этот элемент из

списка и чистим память}

**begin**

**while** (H^.Next^.Subject <> Sg.Cells[0, Sg.Row]) **and** (H^.Next^.Title <> Sg.Cells[1,Sg.Row]) **and** (H^.Next^.Number <> Sg.Cells[2, Sg.Row]) **do**

H := H^.Next;

Temp := H^.Next;

H^.Next := H^.Next^.Next;

Dispose(Temp);

**end**;

{удаляем данные элемента из таблицы}

SGDeleteRow(Sg.Row);

**end**;

**end**; //---btnDeleteClick

//9 ==== очищение строк поиска для ввода строки ================

**procedure** TFrmMain.EdtTitleSearchClick(Sender: TObject);

**begin**

EdtTitleSearch.Clear;

**end**; //---EdtTitleSearchClick

**procedure** TFrmMain.EdtNumberSearchClick(Sender: TObject);

**begin**

EdtNumberSearch.Clear;

**end**; //---EdtNumberSearchClick

//10 ==== процедура печати искомой строки ====================

**procedure** PrintSearch(Searching: string; T: integer);

**var**

H, Temp, PrintDisk: PCatalog;

Comparing: string;

**begin**

**with** FrmMain **do**

**begin**

PrintDisk := nil;

H := Head;

**while** H <> nil **do**

**begin**

**if** T = 1 **then**

Comparing := H^. Title

**Else**

Comparing := H^.Number;

**if** Pos(AnsiLowerCase(Searching), AnsiLowerCase(Comparing)) <> 0 **then**

**begin**

**if** PrintDisk = nil **then**

**begin**

**New**(PrintDisk);

PrintDisk^ := H^;

PrintDisk^.Next := nil;

**end**

**else**

**begin**

Temp := PrintDisk;

**while** Temp^.Next <> nil **do**

Temp := Temp^.Next;

**New**(Temp^.Next);

Temp^.Next^ := H^;

Temp^.Next^.Next := nil;

**end**;

**end**;

H := H^.Next;

**end**;

**if** PrintDisk = nil **then**

ShowMessage( ' Диск не найден ')

**else**

**begin**

Sg.RowCount := 1;

**while** PrintDisk <> nil **do**

**begin**

Sg.RowCount := Sg.RowCount + 1;

Sg.Cells[0, Sg.RowCount - 1] := PrintDisk^.Subject;

Sg.Cells[1, Sg.RowCount - 1] := PrintDisk^.Title;

Sg.Cells[2, Sg.RowCount - 1] := PrintDisk^.Number;

Temp := PrintDisk;

PrintDisk := PrintDisk^.Next;

Dispose(Temp);

**end**;

**end**;

**end**;

**end**; //---PrintSearch

//11 ==== поиск по наименованию =========================

**procedure** TFrmMain.BtnTitleSearchClick(Sender: TObject);

**begin**

PrintSearch(EdtTitleSearch.Text, 2);

**end**;

//12 ==== поиск по количестку ==============================

**procedure** TFrmMain.BtnNumberSearchClick(Sender: TObject);

**begin**

PrintSearch(EdtNumberSearch.Text, 1);

**end**;

//13 ==== процедура сортировки ================================

**procedure** TFrmMain.Sort(SortType: integer);

**var**

H, N, NewHead, Temp: PDisk;

DiskType, NewDiskType, RowNum, ColNum: integer;

CurName, NewName: String; {NewName - диск из отсортированного списка,

CurName - текущий диск. Строковые вел-ы для сравнения}

**begin**

H := Head;

NewHead := nil;

**while** H <> nil **do** {выполнять, пока не дойдем до конца списка}

**begin** {DiskType - числовое значение категории текущего диска: 0, 1}

**if** H^.Subject = 'Прикладное ПО' **then** DiskType := 0

**else if** H^.Subject = 'Системное ПО' **then** DiskType := 1;

{если сортированного списка нет, то создаём (голову)}

**if** NewHead = nil **then**

**begin**

**New**(NewHead);

NewHead^ := H^;

NewHead^.Next := nil;

**End** //---

**else**

{если сортированный список есть}

**begin**

N := NewHead; {N - назначаем новой головой}

{если сортированный список состоит из 1 элемента и его

приоритет выше, чем текущего, то текущий ставим после

сортированного}

{СПИСОК = 1 ЭЛЕМЕНТ}

**if** NewHead^.Next = nil **then**

**begin**

**New**(N^.Next);

N^.Next^ := H^;

N^.Next^.Next := nil;

{SortType = 0, если сортируем по наименованию, 1 - по количеству}

**if** SortType = 0 **then**  {сортируем по наименованию}

**begin**

CurName := H^.Title;

NewName := N^.Title;

**end**

**else** {SortType = 1 - по количеству}

**begin**

CurName := H^.Number;

NewName := N^.Number;

**end**;

{добавляем в сортир-ный список элемент из текущего списка}

**if** (AnsiLowerCase(CurName)) > (AnsiLowerCase(NewName)) **then**

**begin**

**New**(NewHead^.Next); {создаем место для нового звена}

NewHead^.Next^ := H^;

NewHead^.Next^.Next := nil;

**end**

**else**  {AnsiLowerCase(CurName) < AnsiLowerCase(NewName)}

**begin**

Temp := NewHead;

**New**(NewHead);

NewHead^ := H^;

NewHead^.Next := Temp;

**end**;

**end** //---- if NewHead^.Next = nil

**ELSE**

**{**если в сортированном списке больше одного элемента}

{СПИСОК = БОЛЬШЕ ОДНОГО ЭЛЕМЕНТА}

**begin**

**if** SortType = 0 **then**  {сортируем по наименованию}

**begin**

CurName := H^.Title;

NewName := N^.Title;

**end**

**else** {SortType = 1 - по количеству}

**begin**

CurName := H^.Number;

NewName := N^.Number;

**end**;

{проверяем I элемент сортированного списка:}

**if** AnsiLowerCase(CurName) < AnsiLowerCase(NewName) **then**

**begin**

Temp := NewHead;

**New**(NewHead);

NewHead^ := H^;

NewHead^.Next := Temp;

**End**

**else**  {проверяем остальные элементы}

**begin**

**repeat** {если дошли до конца отсортированного списка, то добавляем

после последнего отсортированного элемента текущий элемент}

**if** N^.Next = nil **then**

**begin**

**New**(N^.Next);

N^.Next^ := H^;

N^.Next^.Next := nil;

**break**;

**end**;

**if** SortType = 0 **then**  {сортируем по наименованию}

**begin**

CurName := H^.Title;

NewName := N^.Next^.Title;

**end**

**else**  {SortType = 1 - по количеству}

**begin**

CurName := H^.Number;

NewName := N^.Next^.Number;

**end**;

{если приоритет текущего диска > следующего за отсортированным,

то идем на следующий сортированный диск}

**if**(AnsiLowerCase(CurName)) > (AnsiLowerCase(NewName)) **then**

N := N^.Next

**else** {если (приоритет текущего не равен приоритету следующего)

ИЛИ (приоритет текущ < следующего за отсортированным),

то добавляем текущий диск к отсортированным}

**begin**

Temp := N^.Next;

**New**(N^.Next);

N^.Next^ := H^;

N^.Next^.Next := Temp;

**break**;

**end**;

**until** N = nil; {повторяем, пока не пройдемся по всему сортированному

списку}

**end**;

H := H^.Next; {переход на следующий элемент(диск) текущего списка}

**end**; //--- 1 или больше 1го элемента

**end**; // ---если сортированный список есть

{когда прошли по всему текущему списку - отсортированный список готов,

неотсортированный список удаляем }

**if** Head <> nil **then**

**begin**

**while** Head <> nil **do**

**begin**

Temp := Head;

Head := Head^.Next;

Dispose(Temp);

**end**;

Head := NewHead;

{печать отсортированного списка}

**if** IsAdded = true **then** {var IsAdded: Boolean. Выбрана 1 из кнопок

RadioGroup}

Sg.RowCount := Sg.RowCount + 1;

H := Head;

**with** FrmMain **do**

**for** RowNum := 1 **to** Sg.RowCount - 1 **do**

**begin**

**for** ColNum := 0 **to** 2 **do**

**begin**

**case** ColNum **of**

0: Sg.Cells[ColNum, RowNum] := H^.Subject;

1: Sg.Cells[ColNum, RowNum] := H^. Title;

2: Sg.Cells[ColNum, RowNum] := H^. Number;

**end**;

**end**;

H := H^.Next;

**end**;

**end**;

**end**;

**end**; //---КОНЕЦ СОРТИРОВКИ.

//14 ===== при нажатии на вид сортировки ======================

**procedure** TFrmMain.RGClick(Sender: TObject);

**begin**

IsAdded := false; {никакая радио кнопка не выбрана}

**if** Rg.ItemIndex = 0 **then**

TFrmMain.Sort(0); {сортируем по наименованию = name}

**if** Rg.ItemIndex = 1 **then**

TFrmMain.Sort(1); {сортируем по автору}

**end**;

**end**. //----- end unit Unit1

//======= unit u\_f2 =====================================

**unit** u\_f2;

**interface**

**uses**

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Unit1;

**type**

TFrmNewDisk = **class**(TForm)

lLblSubject: TLabel;

LblTitle: TLabel;

LblNumber: TLabel;

EdtTitle: TEdit;

EdtNumber: TEdit;

BtnAddNewDisk: TButton;

cbbSubject: TComboBox;

**procedure** BtnAddNewDiskClick(Sender: TObject);

**private**

{ Private declarations }

**public**

{ Public declarations }

**end**;

**var**

FrmNewDisk: TFrmNewDisk;

**implementation**

{$R \*.dfm}

//-------------------------------------------------------------------------------

//1 ====== процедура добавления диска в список ==================

**procedure** DiskAdd(H: PCatalog);

**var**

i, j: integer;

Temp: PCatalog;

**Begin**

**with** FrmNewDisk **do**

**if** H = nil **then** // если списка нет

**begin**

**New**(Head); // создаём список

**with** Head^ **do**

**begin**

**case** cbbSubject.ItemIndex **of**

0: Head^.Subject := 'Прикладное ПО';

1: Head^.Subject := 'Системное ПО';

**end**;

Title:= EdtTitle.Text;

Number := EdtNumber.Text;

Next := nil;

**with** FrmMain **do**

**begin** {вывод на экран списка из 1 элемента}

Sg.RowCount := Sg.RowCount + 1; {добавляем одну строку}

Sg.Cells[0, Sg.RowCount - 1] := Head^.Subject;

Sg.Cells[1, Sg.RowCount - 1] := Head^.Title;

Sg.Cells[2, Sg.RowCount - 1] := Head^.Number;

ShowMessage(' Диск добавлен ');

**end**;

**end**;

**end** //-------

**else**

{если список есть}

**begin**

**if** cbbSubject.ItemIndex = 0 **then {**если список есть и добавляем Прикладн ПО}

**begin**

**New**(H);

H^.Subject:= ' Прикладное ПО ';

H^.Title:= EdtTitle.Text;

H^.Number := EdtNumber.Text;

H^.Next := Head;

Head := H;

Sg.RowCount := Sg.RowCount + 1;

**for** i := Sg.RowCount - 2 **downto** 1 **do**

Sg.Rows[i + 1] := Sg.Rows[i];

Sg.Cells[0, 1] := Head^.Subject;

Sg.Cells[1, 1] := Head^.Title;

Sg.Cells[2, 1] := Head^.Number;

ShowMessage(' Диск добавлен ');

**Exit**;

**end**;

**end**;

**if** cbbSubject.ItemIndex = 1 **then {**если список есть и добавляем Системн ПО}

**begin**

**New**(H);

H^.Subject:= ' Системное ПО ';

H^.Title:= EdtTitle.Text;

H^.Number := EdtNumber.Text;

H^.Next := Head;

Head := H;

Sg.RowCount := Sg.RowCount + 1;

**for** i := Sg.RowCount - 2 **downto** 1 **do**

Sg.Rows[i + 1] := Sg.Rows[i];

Sg.Cells[0, 1] := Head^.Subject;

Sg.Cells[1, 1] := Head^.Title;

Sg.Cells[2, 1] := Head^.Number;

ShowMessage(' Диск добавлен ');

**Exit**;

**end**;

**end**; //---DiskAdd

//2 ==== добавление новой строки в таблицу главной формы при нажатии на кнопку "Добавить" на второй форме ========

**procedure** TFrmNewDisk.BtnAddNewDiskClick(Sender: TObject);

**begin**

DiskAdd(Head);

//close;

**end**;

**end**.

**ЛИСТИНГ ПРОГРАММЫ НА ЯЗЫКЕ DELPHI**

**unit** Unit1;

**interface**

**uses**

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Grids, ExtCtrls, Buttons, Menus;

**type**

PCatalog = ^Catalog;

Catalog = **Record**

Subject : String[50]; {Subject - категория ПО (прикладное ПО,

системное);}

Title: String[50]; {Title - название ПО;}

Number: String[50]; {Number - количество дисков;}

Next : PCatalog; {Next - указатель на следующий элемент списка}

**end**;

TFrmMain = **class**(TForm)

pnlOperations: TPanel;

pnlSearch: TPanel;

pnlSorting: TPanel;

lblOperations: TLabel;

btnAdd: TButton;

btnDelete: TButton;

lblSearch: TLabel;

EdtTitleSearch: TEdit;

EdtNumberSearch: TEdit;

BtnTitleSearch: TButton;

BtnNumberSearch: TButton;

RG: TRadioGroup;

MainMenu1: TMainMenu;

mniLoad: TMenuItem;

mniSave: TMenuItem;

OpenDialog1: TOpenDialog;

SaveDialog1: TSaveDialog;

SG: TStringGrid;

N1: TMenuItem;

btnInsert: TButton;

**procedure** FormCreate(Sender: TObject);

**procedure** mniSaveClick(Sender: TObject);

**procedure** mniLoadClick(Sender: TObject);

**procedure** N1Click(Sender: TObject);

**procedure** btnAddClick(Sender: TObject);

**procedure** btnInsertClick(Sender: TObject);

**procedure** btnDeleteClick(Sender: TObject);

**procedure** BtnTitleSearchClick(Sender: TObject);

**procedure** BtnNumberSearchClick(Sender: TObject);

**procedure** EdtTitleSearchClick(Sender: TObject);

**procedure** EdtNumberSearchClick(Sender: TObject);

**procedure** RGClick(Sender: TObject);

**procedure** FormClose(Sender: TObject; **var** Action: TCloseAction);

**private**

{ Private declarations }

**public**

{ Public declarations }

**end**;

**var**

FrmMain: TFrmMain;

Head: PCatalog;

**implementation**

**uses** u\_f2;

{$R \*.dfm}

//----------------------------------------------------------------------------

//1 ==== создание главной формы ===========================

**procedure** TFrmMain.FormCreate(Sender: TObject);

**begin**

Sg.ColWidths[0] := 220;

Sg.ColWidths[1] := 220;

Sg.ColWidths[2] := 135;

Sg.Cells[0,0] := ' КАТЕГОРИЯ';

Sg.Cells[1,0] := ' НАИМЕНОВАНИЕ';

Sg.Cells[2,0] := ' КОЛИЧЕСТВО';

ChDir(ExtractFilePath(Application.ExeName));

FrmMain.OpenDialog1.DefaultExt := 'Все файлы (\*.\*) |\*.\*| Файлы Exel (\*.dat) |\*.dat | Текстовые файлы (\*.txt) |\*.txt|';

OpenDialog1.Title:= 'Загрузить Файл';

FrmMain.OpenDialog1.Filter := 'Все файлы (\*.\*) |\*.\*| Файлы Exel (\*.dat) |\*.dat | Текстовые файлы (\*.txt) |\*.txt|';

FrmMain.SaveDialog1.DefaultExt := 'Выберите тип файла | | Файлы Exel (\*.dat) |\*.dat | Текстовые файлы (\*.txt) |\*.txt| Все файлы (\*.\*) |\*.\*|';

SaveDialog1.Title:= ' Сохранить Файл';

FrmMain.SaveDialog1.Filter := 'Выберите тип файла | | Файлы Exel (\*.dat) |\*.dat | Текстовые файлы (\*.txt) |\*.txt| Все файлы (\*.\*) |\*.\*|';

**end**;

//2===== вопрос при закрытии проекта (Y/N) ======================

**procedure** TFrmMain.FormClose(Sender: TObject; **var** Action: TCloseAction);

**begin**

**If** MessageDlg('Завершить приложение?', mtConfirmation, mbOkCancel, 0) = mrOk **Then**

Action := caFree {Нажата Ok - закрываем форму}

**Else**

Action := caNone; {Нажата Cancel - игнорируем закрытие}

**end**;

//3 ======== Сохранение списка в файл =========================

**procedure** TFrmMain.mniSaveClick(Sender: TObject);

**var**

f: TStrings;

i, j: word;

**begin**

f := TStringList.Create;

**try**

**with** sg **do**

**for** i := 0 **to** (ColCount-1) **do**  {Столбцы}

**for** j := 1 **to** (RowCount-1) **do**

f.Add(Cells[i, j]);

**if** FrmMain.SaveDialog1.Execute **then**

f.SaveToFile(FrmMain.SaveDialog1.FileName);

**finally**

f.Free;

**end**;

**end**;

//4 ===== открыть (Загрузить) из файла на форму в таблицу ============

**procedure** TFrmMain.mniLoadClick(Sender: TObject);

**var**

f: TStrings;

i, j, n: word;

**begin**

f := TStringList.Create;

**try**

n := 0;

**if** FrmMain.OpenDialog1.Execute **then**

**begin**

f.LoadFromFile(FrmMain.OpenDialog1.FileName);

**with** sg **do**

**for** i := 0 **to** (ColCount-1) **do**

**for** j := 1 **to** (RowCount-1) **do**

**begin**

Cells[i, j] := f.Strings[n];

inc(n);

**end**;

**end**;

**finally**

f.Free;

**end**;

**end**;

//5 ==== Очищение всего списка =================================

**procedure** TFrmMain.N1Click(Sender: TObject);

**var**

i, j: word;

**begin**

**with** sg **do**

**for** i := 0 **to** (ColCount-1) **do**

**for** j := 1 **to** (RowCount-1) **do**

Cells[i, j] := '';

**end**;

//6 ==== при нажатии на кнопку "Добавить" появляется вторая форма ====

**procedure** TFrmMain.btnAddClick(Sender: TObject);

**begin**

FrmNewDisk.Show;

**end**;

//7 ==== Процедура вставки пустой строки ========================

**procedure** SGInsertRow(SG: TStringGrid; NewRow: Integer);

**var**

i: Integer;

**begin**

**with** SG **do**

**begin**

RowCount := RowCount + 1;

**if** NewRow < RowCount - 1 **then**

**begin**

**for** i := RowCount - 1 **downto** NewRow + 1 **do**

Rows[i].Assign(Rows[i - 1]);

**end**;

Rows[NewRow].Clear;

**end**;

**end**;

//8 ==== Вставка пустой строки =============================

**procedure** TFrmMain.btnInsertClick(Sender: TObject);

**begin**

SGInsertRow(SG, SG.Row + 1);

**end**;

//9 ===== Удаление строки ====================================

**procedure** TFrmMain.btnDeleteClick(Sender: TObject);

**var**

i,j: Integer;

**begin**

j:= SG.Row; {строка с выделением}

**if** (j = 1) **and** (SG.RowCount <= 2) **then**

**begin**

**with** SG **do**

**for** i:= 0 **to** ColCount - 1 **do** Cells[i, j]:= '';

**exit**; {не удаляем последнюю строку чтобы не потерять фиксированной}

**end**;

SG.Rows[j].Clear;

**for** i:=j **to** SG.RowCount-2 **do**

SG.Rows[i].Assign(SG.Rows[i+1]);

SG.RowCount:= SG.RowCount-1;

**end**;

//10 ==== очищение строк поиска для ввода строки =================

**procedure** TFrmMain.EdtTitleSearchClick(Sender: TObject);

**begin**

EdtTitleSearch.Clear;

**end**;

**procedure** TFrmMain.EdtNumberSearchClick(Sender: TObject);

**begin**

EdtNumberSearch.Clear;

**end**;

//11 ==== процедура печати искомой строки ========================

**procedure** PrintSearch(Searching: string; T: integer);

**var**

H, Temp, PrintDisk: PCatalog;

Comparing: string;

**begin**

**with** FrmMain **do**

**begin**

PrintDisk := nil;

H := Head;

**while** H <> nil **do**

**begin**

**if** T = 1 **then**

Comparing := H^. Title

**Else**

Comparing := H^.Number;

**if** Pos(AnsiLowerCase(Searching), AnsiLowerCase(Comparing)) <> 0 **then**

**begin**

**if** PrintDisk = nil **then**

**begin**

**New**(PrintDisk);

PrintDisk^ := H^;

PrintDisk^.Next := nil;

**End**

**else**

**begin**

Temp := PrintDisk;

**while** Temp^.Next <> nil **do**

Temp := Temp^.Next;

**New**(Temp^.Next);

Temp^.Next^ := H^;

Temp^.Next^.Next := nil;

**end**;

**end**;

H := H^.Next;

**end**;

**if** PrintDisk = nil **then**

ShowMessage( ' Диск не найден ')

**else**

**begin**

Sg.RowCount := 1;

**while** PrintDisk <> nil **do**

**begin**

Sg.RowCount := Sg.RowCount + 1;

Sg.Cells[0, Sg.RowCount - 1] := PrintDisk^.Subject;

Sg.Cells[1, Sg.RowCount - 1] := PrintDisk^.Title;

Sg.Cells[2, Sg.RowCount - 1] := PrintDisk^.Number;

Temp := PrintDisk;

PrintDisk := PrintDisk^.Next;

Dispose(Temp);

**end**;

**end**;

**end**;

**end**;

//12 ==== поиск по наименованию ==========================

**procedure** TFrmMain.BtnTitleSearchClick(Sender: TObject);

**begin**

PrintSearch(EdtTitleSearch.Text, 2);

**end**;

//13 ==== поиск по количестку ================================

**procedure** TFrmMain.BtnNumberSearchClick(Sender: TObject);

**begin**

PrintSearch(EdtNumberSearch.Text, 1);

**end**;

//14 ==== процедура сортировки =============================

**procedure** SGSort(SG: TStringGrid; NoCol: Integer);

**var**

Line, PosAct: Integer;

Row: TStringList;

Reng : TStringList;

**begin**

Row := TStringList.Create;

Reng := TStringList.Create;

**for** Line := 1 **to** SG.RowCount-1 **do**

**begin**

PosAct := Line;

Row.Assign(SG.Rows[PosAct]);

**while** True **do**

**begin**

**if** (PosAct = 0) **or** (Row.Strings[NoCol] >= SG.Cells[NoCol, PosAct-1])

**then break**;

SG.Rows[PosAct] := SG.Rows[PosAct-1];

Dec(PosAct); {по убывающей}

**end**;

**if** (Row.Strings[NoCol] < SG.Cells[NoCol, PosAct]) **then**

SG.Rows[PosAct].Assign(Row);

**end**;

Row.Free;

Reng.Free;

**end**;

//15 ====== при нажатии на одну из кнопок сортировки ============

**procedure** TFrmMain.RGClick(Sender: TObject);

**begin**

**with** FrmMain.SG **do**

**begin**

**if** Rg.ItemIndex = 0 **then**

SGSort(SG, 1); {сортируем по наименованию = name}

**if** Rg.ItemIndex = 1 **then**

SGSort(SG, 2);

**end**;

**end**;

**end**. //---- end unit Unit1

//================ unit u\_f2 =============================

**unit** u\_f2;

**interface**

**uses**

Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,

Dialogs, StdCtrls, Unit1;

**type**

TFrmNewDisk = **class**(TForm)

lLblSubject: TLabel;

LblTitle: TLabel;

LblNumber: TLabel;

EdtTitle: TEdit;

EdtNumber: TEdit;

BtnAddNewDisk: TButton;

cbbSubject: TComboBox;

**procedure** BtnAddNewDiskClick(Sender: TObject);

**procedure** EdtTitleClick(Sender: TObject);

**procedure** EdtNumberClick(Sender: TObject);

**private**

{ Private declarations }

**public**

{ Public declarations }

**end**;

**var**

FrmNewDisk: TFrmNewDisk;

**implementation**

{$R \*.dfm}

//-----------------------------------------------------------------------------

//1==== добавление новой строки при нажатии на кнопку "Добавить" ====

**procedure** TFrmNewDisk.BtnAddNewDiskClick(Sender: TObject);

**var** i: Integer;

**begin**

**with** FrmMain.SG **do**

**begin**

**case** cbbSubject.ItemIndex **of**

0: cells[0,rowcount-1]:= cbbSubject.Items[0];

1: cells[0,rowcount-1]:= cbbSubject.Items[1];

**end**;

cells[1,rowcount-1]:= EdtTitle.text;

cells[2,rowcount-1]:= EdtNumber.text;

rowcount := rowcount+1;

**end**;

//close;

**end**;

//2 ===== Очищение поля Edit при клике на него ================

**procedure** TFrmNewDisk.EdtTitleClick(Sender: TObject);

**begin**

EdtTitle.clear;

**end**;

**procedure** TFrmNewDisk.EdtNumberClick(Sender: TObject);

**begin**

EdtNumber.Clear;

**end**;

**end**.