**ЛАБОРАТОРНАЯ РАБОТА № 5**

# ПОСЛЕДОВАТЕЛЬНОСТИ ДАННЫХ

## ТЕОРИЯ И ПОЯСНЕНИЯ

Для размещения в памяти последовательности данных одного типа можно использовать массив или создать список, в котором каждый элемент содержит одно из значений данных.

### *Массивы*

Для любого массива определены функции:

**Length** число элементов;

**High** наибольшее значение индекса;

**Low** наименьшее значение индекса.

Функции для числовых массивов, определенные в модуле Math (файл ..\Lazarus\fpc\2.0.4\source\rtl\objpas\math.pp):

**MaxIntValue** максимальное значение в массиве целых чисел;

**MinIntValue** минимальное значение в массиве целых чисел;

**MaxValue** максимальное значение в числовом массиве;

**MinValue** минимальное значение в числовом массиве;

**Sum** сумма элементов массива.

### *Статический массив*

Статический массив, т.е. массив с исходно заданным и неизменяемым количеством элементов, удобен в тех случаях, когда длина последовательности данных известна до начала выполнения программы.

Пример описания статического массива.

Длина массива – константа в коде программы



Выделение памяти для 100 элементов

### *Динамический массив*

Динамический массив следует применять, если количество элементов в последовательности данных удобнее задать во время работы, а не при описании массива. Описание динамического массива:

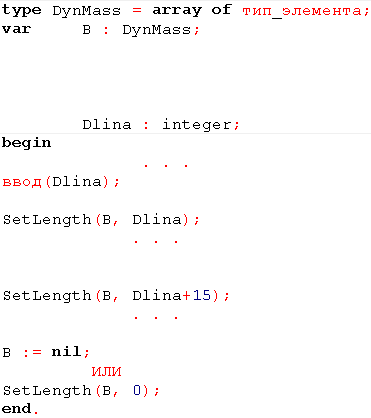


Указатель на начало массива B = **nil**

Выделение нужного количества памяти для динамического массива производится процедурой



Пример описания и размещения в памяти динамического массива:



Выделение памяти в куче для указанного количества элементов,

значения всех элементов равны 0

Выделение новой памяти для массива,

вычисленные значения копируются в новую память

До вызова SetLength B = **nil**

Переменная – будущая длина массива

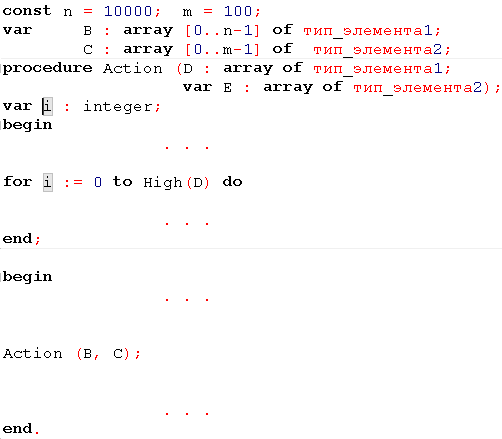
Указатель на начало массива

Удаление массива из памяти

### *Открытый массив*

Открытый массив – формальный параметр процедуры или функции, описанный как массив базового типа без указания размерности, т.е. как динамический массив. При вызове процедуры фактические массивы, соответствующие открытым массивам, объявленным как параметры-значения, копируются в стек, и с этой копией работает процедура. Если открытый массив объявлен как параметр-переменная, то при вызове в процедуру передается ссылка на размещенный в памяти фактический массив.

Элементы открытого массива нумеруются с 0. Соответствующий фактический массив может иметь другую нумерацию, сопоставление элементов выполняется автоматически от начала массивов. Однако для удобства отладки рекомендуется фактические массивы нумеровать от 0. В процедурах с открытыми массивами весьма полезна функция High, она избавляет от необходимости передавать как параметр длину фактического массива. Пример использования открытого массива:



Адрес начала массива C копируется в стек

Цикл по всем элементам массива D

10000 элементов массива B копируются в стек

### *Списки*

***Список*** эффективно и удобно использовать, если решение задачи связано с многократными удалениями и вставками отдельных элементов в последовательность данных.

***Линейный список*** представляет собой последовательно соединенные друг с другом элементы. Каждый элемент списка состоит из двух частей: содержимое и указатель, значение которого – адрес следующего элемента списка. Содержимое, в свою очередь, тоже может состоять из нескольких частей, каждая – своего типа. Следовательно, элемент списка является структурой данных, которую можно описать с помощью типа запись (**record**), содержащего, по крайней мере, два поля: данное и указатель. Записи с указателями на такие же записи называются самоадресуемыми. Таким образом, линейный список строится на основе самоадресуемой записи.

Пример создания списка из двух элементов, содержащего целочисленные данные:

ДАННОЕ1

Адрес

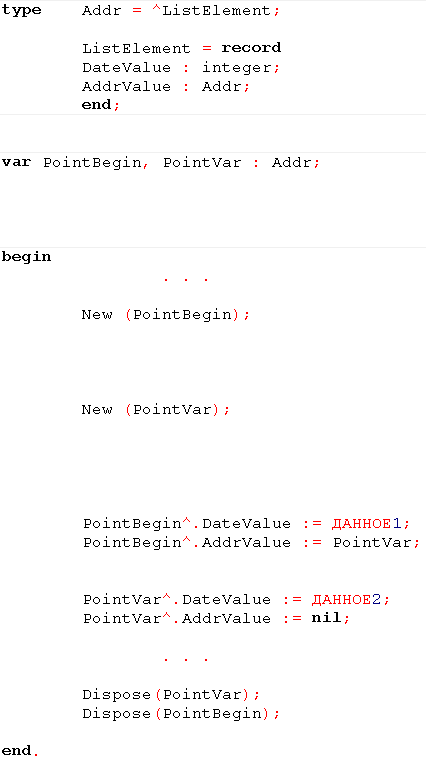
второго

элемента

ДАННОЕ2

Пустой адрес **nil**

Адрес первого элемента



Выделение памяти из ***кучи (heap)*** первому элементу списка. Начальный адрес выделенной памяти становится значением указателя PointBegin

Заполнение второго элемента списка: **nil** – пустая ссылка

Заполнение первого элемента списка

Освобождение памяти

Операция разыменования **^** – обращение к записи, адрес которой содержится в указателе

Выделение памяти для второго элемента

Переменные – указатели на самоадресуемую запись

Тип записи с указателем на этот же тип

Тип указателя на запись

### *Организация диалога для работы с файлами из приложения*

При выполнении заданий вы будете создавать 3 файла, их имена можно задать непосредственно в программном коде. Переход из приложения к просмотру полученных файлов не предусматривается. Однако отсутствие интерфейса для работы с файлами ограничивает возможности пользователя.

Диалог для открытия (сохранения) нужного файла во время работы приложения обеспечивает компонент класса **TOpenDialog (TSaveDialog)**. Компонент не отображается на форме во время работы приложения. Основные *свойства* этих классов:

**DefaultExt** – расширение, которое будет дано создаваемому файлу по умолчанию;

**FileName** – имя открытого (сохраненного) файла;

**Filter** – **Filter name** – текст в поле Тип файла в окне диалога;

**– Filter** – шаблон для выбора файлов определенного типа;

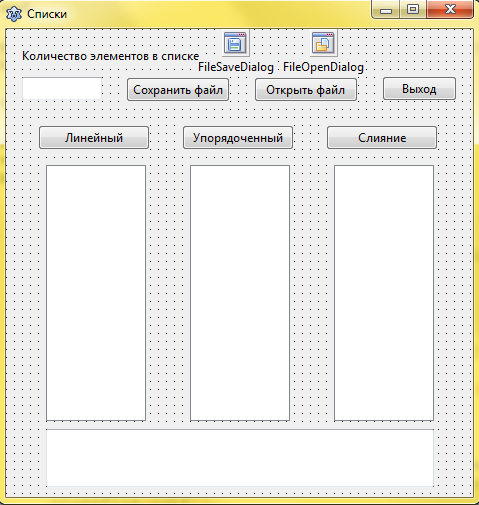
**Title** – заголовок в окне диалога.

*Метод* класса **function TOpenDialog.Execute:Boolean;**

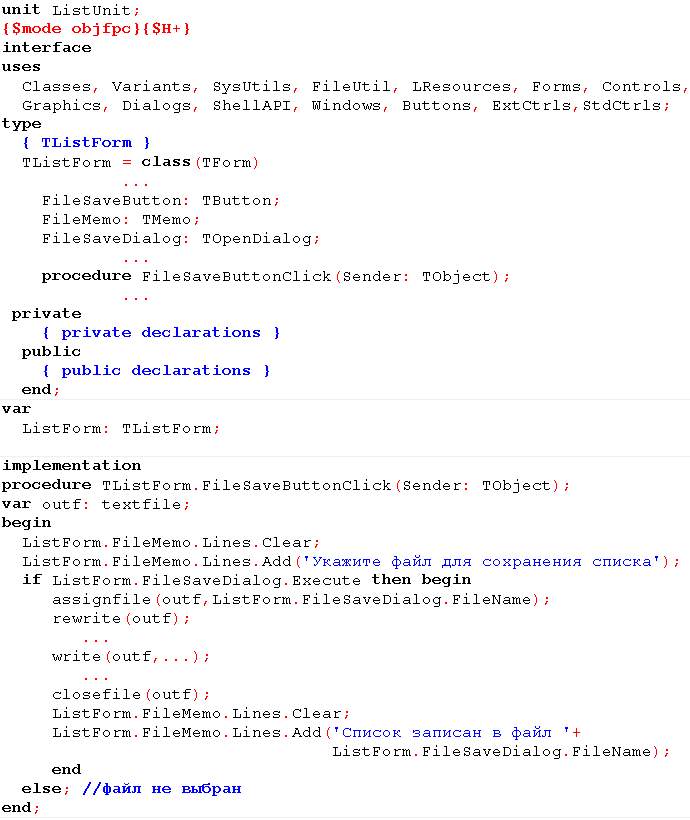
формирует стандартное окно диалога и возвращает значение **TRUE**, если пользователь завершает диалог кнопкой **Открыть.** Аналогичный метод **function TSaveDialog.Execute:Boolean;** возвращает значение **TRUE**, если пользователь завершает диалог кнопкой **Сохранить**.

Вызов метода **Execute** выполняется из процедуры обработки события, например, нажатия на определенную кнопку.

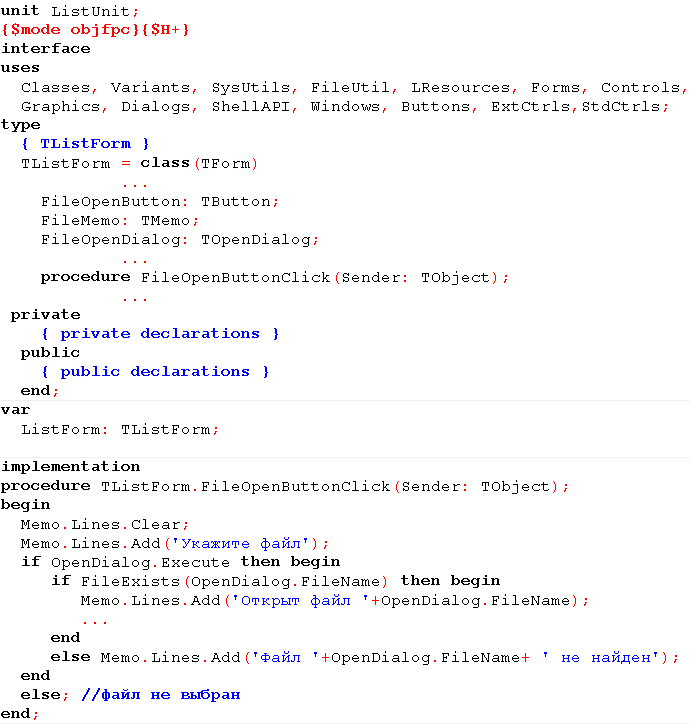
Возможный вид формы приводится ниже.



Фрагмент программного кода для сохранения файла:



Фрагмент программного кода для проверки наличия файла и открытия этого файла:



Чтобы указанный для открытия файл открылся для чтения и редактирования в соответствующем ему приложении (текстовый файл ­– в Блокноте), нужно использовать функцию **ShellExecute** из модуля **ShellAPI** со следующими значениями параметров:

**ShellExecute**(дескриптор (**Handle**) окна приложения,

**'open'**,

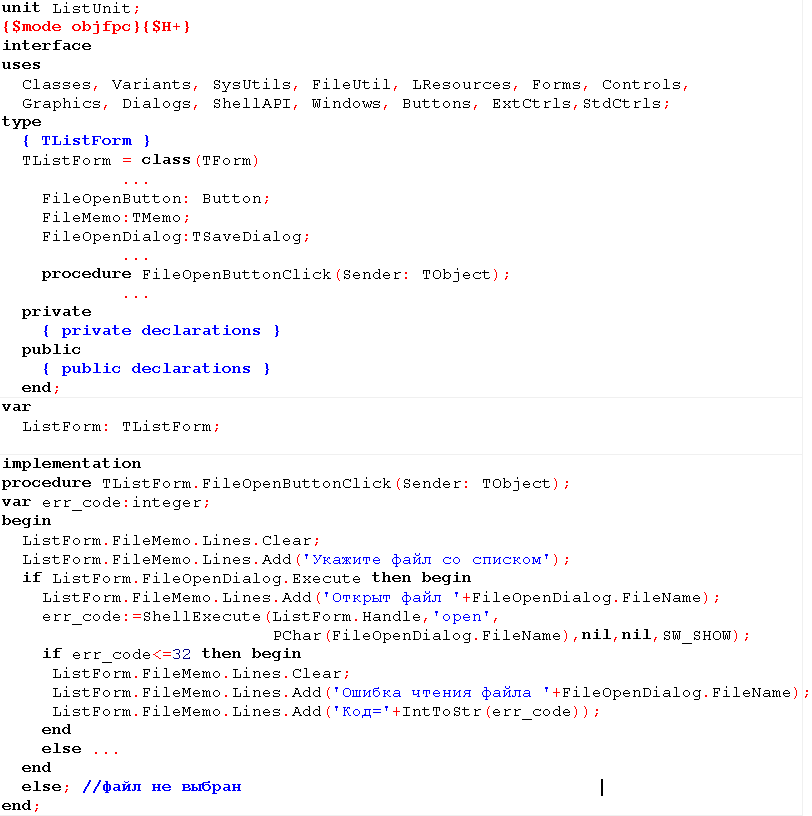
**PChar**(имя файла),

**nil,**

**nil**,

**SW\_SHOW**).

Фрагмент программного кода для открытия файла и чтения его в соответствующем приложении:



## ЗАДАНИЯ

### ОФОРМЛЕНИЕ ПРОЕКТА ДЛЯ ЛАБОРАТОРНОЙ РАБОТЫ № 2 (АЛГОРИТМЫ СОРТИРОВКИ) С ИСПОЛЬЗОВАНИЕМ СТАТИЧЕСКИХ, ДИНАМИЧЕСКИХ И ОТКРЫТЫХ МАССИВОВ.

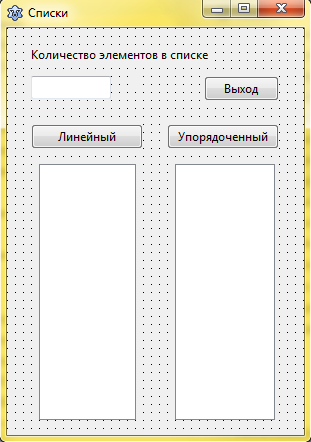
Освобождайте память после вывода результатов.

### ГЕНЕРАЦИЯ ЛИНЕЙНОГО СПИСКА.

Создайте приложение для генерации линейного списка, представляющего в памяти целочисленную последовательность данных, генерируемых случайным образом из интервала значений [‑999, 999]. Алгоритм генерации списка оформите в виде отдельной процедуры с параметрами.

***Обязательной входной величиной*** в процедуре генерации списка является число *n* элементов списка.

***Результатом*** работы процедуры является список, размещенный в динамической памяти. Готовый список выведите в текстовый файл. Для списка, удовлетворяющего условию *n<=40,* также выполните вывод данных на форму в окно соответствующего компонента (объект классов TMemo или TList). Возможный вид формы приводится ниже (задания II.2 и II.3 выполняются в одном модуле, используется одна форма).



### ГЕНЕРАЦИЯ УПОРЯДОЧЕННОГО ЛИНЕЙНОГО СПИСКА.

В модуле для задания II.2 дополнительно создайте упорядоченный линейный список. В процедуре формирования списка добавление каждого очередного элемента выполняйте как включение в упорядоченный список.

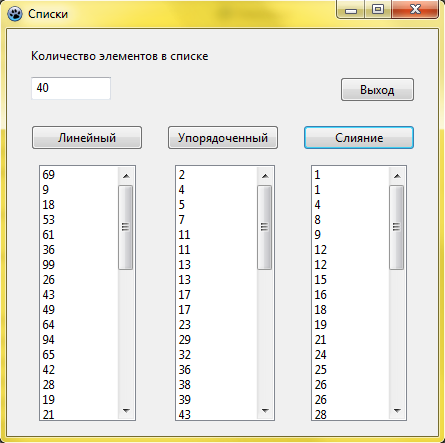
***Обязательной входной величиной*** в процедуре генерации упорядоченного линейного списка является число *n* элементов списка.

***Результатом*** работы процедуры является список, размещенный в динамической памяти. Готовый список выведите в текстовый файл. Для списка, удовлетворяющего условию *n<=40,* также выполните вывод данных на форму в окно соответствующего компонента (объект классов TMemo или TList). Возможный вид формы приведен в задании II.2 (задания II.2 и II.3 выполняются в одном модуле, используется одна форма).

**Следующие задания не являются обязательными для выполнения.**

### УПОРЯДОЧЕНИЕ МЕТОДОМ СЛИЯНИЯ ЛИНЕЙНОГО СПИСКА, СОЗДАННОГО ПРИ ВЫПОЛНЕНИИ ЗАДАНИЯ II.2.

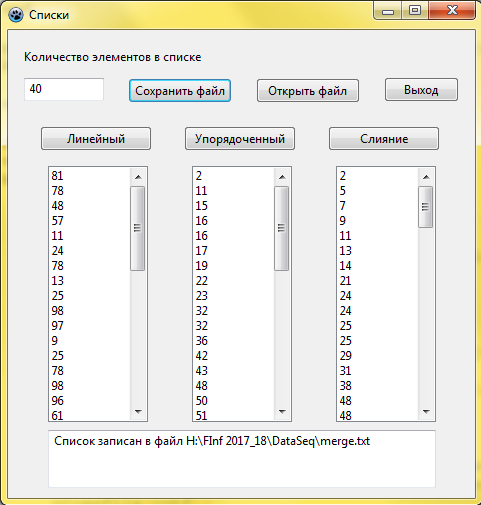
Окно работающего приложения:



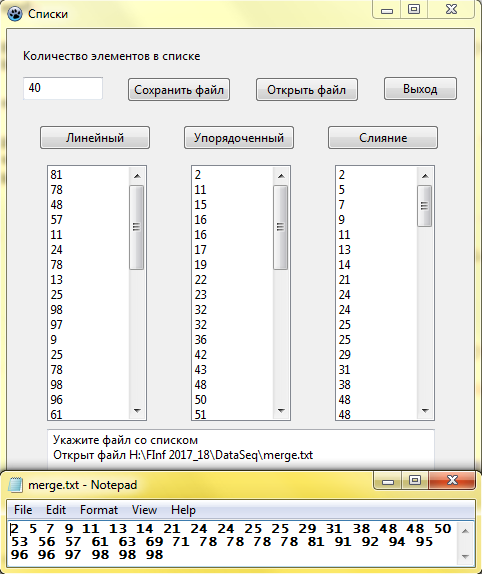
### ДОПОЛНЕНИЕ ПРИЛОЖЕНИЯ ЭЛЕМЕНТАМИ ДИАЛОГА:

– выбор или ввод имени файла, в который записывается сформированный список;

– просмотр этого файла, инициированный из созданного приложения.



Окно работающего приложения ­– сохранение файла



Окно работающего приложения ­– открытие и чтение файла в соответствующем приложении