**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРСТВЕНЫЙ УНИВЕРСИТЕТ»**

ОСНОВЫ ПРОГРАММИРОВАНИЯ

САМАРА 2015

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО
ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«САМАРСКИЙ ГОСУДАРТСВЕННЫЙ УНИВЕРСИТЕТ»

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Методические указания

САМАРА

2015

УДК 519.683.2, 519.72
ББК 32.973

Составители ***И.В. Семенова***

Рецензент к.ф.-м.н., В.М. Сиников

Основы программирования: метод. указания / сост. *И.В. Семенова* – Самара, 2015. – 108с.: ил.

Практикум состоит из разделов, посвященных основам алгоритмизации и программирования. В каждом разделе есть необходимый теоретический материал, примеры решения задач, а также задачи для самостоятельного решения.

Предназначено для студентов специальностей «Математическое обеспечение и администрирование информационных систем»,.

Подготовлено на кафедре информатики и вычислительной математики.

УДК 519.683.2, 519.72
ББК 32.973

СОДЕРЖАНИЕ

Лабораторная работа1. Основы языка Pascal

Теоретический материал

Алфавит. Алфавит языка Pascal включает в себя буквы, цифры, шестнадцатеричные цифры, специальные символы и пробелы.

Буквы — прописные (A—Z) и строчные (a—z) буквы латинского алфавита, а также знак подчеркивания. В языке нет различия между заглавными и строчными буквами алфавита, если только они не входят в символьные и строковые выражения.

Цифры — арабские цифры 0, 1,..., 9.

Каждая *шестнадцатеричная цифра* имеет значение от 0 до 15. Первые 10 значений обозначаются арабскими цифрами 0-9, остальные шесть – латинскими буквами A-F или a-f.

Специальные символы — + - * / . , : ; = > < • () {} [ ] # $ @ ^. Комбинации специальных символов образуют следующие *составные специальные символы*:

:= — присваивание;

<> — не равно;

.. — диапазон значений;

<= — меньше или равно;

>= — больше или равно;

* и * — альтернатива фигурным скобкам { и };

. и . — альтернатива квадратным скобкам [и].

В программе эти пары символов нельзя разделять пробелами.

Особое место в алфавите языка занимают пробелы, к которым относятся любые символы в диапазоне кодов от 0 до 32. Эти символы рассматриваются как ограничители идентификаторов, констант, чисел, зарезервированных слов. Несколько следующих друг за другом пробелов считаются одним пробелом.

Словарь языка. Неделимые последовательности знаков алфавита образуют *слова,* отделяемые друг от друга разделителями и несущие

определенный смысл в программе.

Разделителями могут служить пробел, символ концастроки, комментарий, другие специальные символы и их комбинации.

Слова подразделяются на:

- зарезервированные слова;
- стандартные идентификаторы;
- идентификаторы пользователя.

Зарезервированные (ключевые) слова являются составной частью языка, имеют фиксированное написание и однозначно определенный смысл, изменить который программист не может. Например, зарезервированными являются слова: label, unit, begin, interface, and, array, case do. В редакторе кода ключевые слова выделяются полужирным шрифтом.

Стандартные идентификаторы служат для обозначения следующих заранее определенных разработчиками конструкций языка:

- типов данных;
- констант;
- процедур и функций.

В отличие от зарезервированных слов любой из стандартных идентификаторов можно переопределить. Так как это может привести к ошибкам, то стандартные идентификаторы лучше использовать без каких-либо изменений. Примерами стандартных идентификаторов являются слова sin, pi, real.

Идентификаторы пользователя применяются для обозначения имен констант, переменных, процедур, функций и типов данных. Эти имена задаются программистом и должны отвечать следующим правилам:

- Идентификатор составляется из букв и цифр;
- Идентификатор всегда начинается только с буквы.

В идентификаторе можно использовать как строчные, так и прописные буквы, компилятор интерпретирует их одинаково. Так как нельзя

использовать специальные символы, то для наглядности отдельные составляющие идентификатора полезно выделять прописными буквами, например, btnOpen, или разделять их с помощью знака подчеркивания, который также относится к буквам, например, PictureiD.

• Между двумя идентификаторами в программе должен быть по крайней мере один разделитель.

Комментарии. *Комментарий* представляет собой пояснительный текст, который можно записывать в любом месте программы, где разрешен пробел. Текст комментария ограничен символами (* и *) или их эквивалентами { и } и может содержать любые символы языка, в том числе русские буквы. Комментарий, ограниченный данными символами, может занимать несколько строк.

Однострочный комментарий в начале строки содержит двойной слэш //.

Пример.

Варианты комментариев:

(*Однострочный комментарий *)

// Второй однострочный комментарий

(* Начало многострочного комментария

Окончание многострочного комментария *)

Комментарий игнорируется компилятором и не оказывает никакого влияния на выполнение программы. С помощью комментариев на период отладки можно исключить какие-либо операторы программы.

Виды данных. Обрабатываемые в программе данные подразделяются на переменные, константы и литералы.

Константы представляют собой данные, значения которых установлены в разделе объявления констант и не изменяются в процессе выполнения программы.

В языке Pascal существует два вида констант: *обычные* и

именованные.

Обычная константа — это целое или дробное число, строка символов или отдельный символ, логическое значение.

В тексте программы числовые константы записываются обычным образом, т. е. так же, как числа, например, при решении математических задач. При записи дробных чисел для разделения целой и дробных частей используется точка. Если константа отрицательная, то непосредственно перед первой цифрой ставится знак "минус".

Примеры числовых констант: 1230.0 и -524.03

Дробные константы могут изображаться в виде числа с плавающей точкой. Представление в виде числа с плавающей точкой основано на том, что любое число может быть записано в алгебраической форме как произведение числа, меньшего 10, которое называется мантиссой, и степени десятки, именуемой порядком.

Примеры чисел, записанных в обычной форме, в алгебраической форме и форме с плавающей точкой.

Число	**Алгебраическая форма**	**Форма с плавающей точкой**
1 000 000	$1x10^{6}$	1 .0000000000E+06
-123.452	$-1,23452x10^{2}$	-1 .2345200000E+02
0,0056712	$5,6712x10^{-3}$	5,6712000000E-03

Строковые и символьные константы

Строковые и символьные константы заключаются в кавычки.

Примеры строковых констант:

'Язык программирования Delphi[1] 'Delphi 7'

'2.4'

'Д'

Следует обратить внимание на константу ' 2.4'. Это именно символьная константа, т. е. строка символов, которая изображает число "две целые четыре десятых", а не число 2,4.

Логические константы

Логическое высказывание (выражение) может быть либо истинно, либо ложно. Истине соответствует константа True, значению "ложь" - константа False.

Именованная константа — это имя (идентификатор), которое в программе используется вместо самой константы.

Именованная константа, как и переменная, перед использованием должна быть объявлена. В общем виде инструкция объявления именованной константы выглядит следующим образом:

константа = значение;

где:

- *константа* — имя константы;
- *значение* — значение константы.

Именованные константы объявляются в программе в разделе объявления констант, который начинается словом **const**. Ниже приведен пример объявления именованных констант (целой, строковой и дробной).

```
const
Bound = 10;
Title = 'Скорость бега';
pi = 3.1415926;
```

После объявления именованной константы в программе вместо самой константы можно использовать ее имя.

В отличие от переменной, при объявлении константы тип явно не указывают. Тип константы определяется ее видом, например:

- 125 — константа целого типа;
- 0.0 — константа вещественного типа;
- ' выполнить ' — строковая константа;
- ' \' — символьная константа.

Переменная — это область памяти, в которой находятся данные, которыми оперирует программа. Когда программа манипулирует с данными, она, фактически, оперирует содержимым ячеек памяти, т. е. переменными. Чтобы программа могла обратиться к переменной (области памяти), например, для того, чтобы получить исходные данные для расчета по формуле или

сохранить результат, переменная должна иметь имя. Имя переменной придумывает программист.

Переменные объявляются в разделе объявления переменных, однако в отличие от констант, свои значения они получают в процессе выполнения программы, причем эти значения можно изменять.

К константам и переменным можно обращаться по именам.

Литерал не имеет имени и представляется в тексте программы непосредственно значением, поэтому литералы также называют просто *значениями.*

Каждый элемент данных принадлежит к определенному типу, при этом тип переменной указывается при ее описании, а тип констант и литералов распознается компилятором автоматически по указанному значению.

Типы данных. *Тип* определяет множество значений, которые могут принимать элементы программы, и совокупность операций, допустимых над этими значениями.

Например, значения -34 и 67 относятся к целочисленному типу и их можно умножать, складывать, делить и выполнять другие арифметические операции, а значения abed и sdfhi23 относятся к строковому типу и их можно сцеплять (складывать), но нельзя делить или вычитать.

Типы данных можно разделить на следующие группы:

• простые;

• структурные;

• указатели;

• процедурные;

• вариантные.

В свою очередь, простые и структурные типы включают в свой состав другие типы, например, целочисленные или массивы. Приводимое деление на типы в некоторой мере условно — иногда указатели причисляют к простым типам, а строки, которые относятся к структурным типам, выделяют в отдельный тип.

Важное значение имеет понятие *совместимости типов,* которое

означает, что типы равны друг другу или один из них может быть автоматически преобразован к другому. Совместимыми, например, являются вещественный и целочисленный тип, так как целое число автоматически преобразовывается в вещественное, но не наоборот.

Простые типы данных.

Простые типы не содержат в себе других типов, и данные этих типов могут одновременно содержать одно значение. К простым относятся следующие типы:

- целочисленные;
- литерные (символьные);
- логические (булевы);
- вещественные.

Все типы, кроме вещественного, являются *порядковыми,* то есть значения каждого из этих типов образуют упорядоченную конечную последовательность. Номера соседних значений в ней отличаются на единицу.

Целочисленные типы.

Целочисленные типы включают целые числа. Наиболее часто используется тип integer, допускающий значения в диапазоне от -2 147 483 648 до 2 147 483 647 и используемый для представления чисел в 32 битовом формате.

Также язык Pascal поддерживает еще ряд целых типов данных: shortint, smailint, longint, int64, byte, word и longword.

Для записи целых чисел можно использовать цифры и знаки + и -, если знак числа отсутствует, то число считается положительным. При этом число может быть представлено как в десятичной, так и в шестнадцатеричной системе счисления. Если число записано в шестнадцатеричной системе, то перед ним ставится знак $ (без пробела), а допустимый диапазон значений — от $00000000 до $FFFFFFFF.

Литерные типы.

Значениями *литерного типа* являются элементы из набора литер, то есть отдельные символы. В Pascal определен литерный тип char, который занимает

один байт, а для кодирования символов используется код американского национального института стандартов ANSI (American National Standards Institute). Также язык Pascal поддерживает еще два символьных типа: ansichar и widechar.

Логический тип.

Данные *логического* (булевого) типа данных могут иметь два значения: True (*Истина*) и False (*Ложь*). Для описания данных логического типа используется зарезервированное слово **Boolean**.

Var Имя_переменной: **Boolean**;

Логический тип данных относится к порядковым типам, которые представляют собой упорядоченное множество значений. Логический тип можно считать частным случаем перечисляемого типа с двумя значениями. Данные типа Boolean занимают в памяти компьютера 1 бит. Значению True соответствует число 1, а False – 0.

В Pascal к логическому относится тип Boolean. Этот тип представлен двумя возможными значениями: True (истина) и False (ложь). Для представления логического значения требуется один байт памяти.

Вещественные типы.

Вещественные (действительные) типы включают в себя вещественные числа. Наиболее часто используется тип Real, допускающий максимальное значение $1{,}7 \times 10^{308}$, обеспечивающий точность 15—16 цифр мантиссы и использующий для представления одного числа 8 байт.

Также язык Pascal поддерживает еще ряд вещественных типов: real48, single, double, extended, comp, currency. Типы различаются между собой диапазоном допустимых значений, количеством значащих цифр и количеством байтов, необходимых для хранения данных в памяти компьютера.

Запись вещественных чисел возможна в форме с фиксированной и в форме с плавающей точкой.

Вещественные числа с фиксированной точкой записываются по обычны правилам арифметики. Целая часть отделяется от дробной десятичной

точкой. Перед числом может указываться знак + или -. Если знак отсутствует, то число считается положительным.

Для записи вещественных чисел с плавающей точкой указывается порядок числа со знаком, отделенный от мантиссы знаком E (или e). Примерами вещественных чисел являются: 12.5, -137.0, +10E+3.

Типы Comp и currency представляют вещественные числа с фиксированной точкой и введены для точных расчетов денежных сумм.

Структурные типы данных.

Структурные типы имеют в своей основе один или более других типов, в том числе и структурных. К структурным типам относятся:

• строки;

• массивы;

• множества;

• записи;

• файлы;

• классы.

Операторы. *Операторы* представляют собой законченные предложения языка, которые выполняют некоторые действия над данными. Операторы можно разделить на две группы:

• простые;

• структурированные.

Структурированные операторы могут включать в себя другие операторы. Например, к простым операторам относится оператор присваивания, к структурированным операторам — операторы разветвлений и циклов.

Правила записи операторов.

Операторы разделяются точкой с запятой. Точка с запятой является разделителем операторов, и ее отсутствие между операторами является ошибкой.

Наличие между операторами нескольких точек с запятой не является

ошибкой, так как они обозначают пустые операторы. Отметим, что лишняя точка с запятой в разделе описаний и объявлений является синтаксической ошибкой.

Точка с запятой может не ставиться после слова **begin** и перед словом **end**, так как они являются операторными скобками, а не операторами. В условных операторах и операторах выбора точка с запятой не ставится после слова **then** и перед словом **else**. Отметим, что в операторе цикла с параметром наличие точки с запятой сразу после слова do синтаксической ошибкой не является, но в этом случае тело цикла будет содержать только пустой оператор.

Оператор присваивания.

Оператор присваивания является основным оператором языка. Он предписывает вычислить выражение, заданное в его правой части, и присвоить результат переменной, имя которой расположено в левой части оператора.

Переменная и выражение должны иметь совместимый тип, например, вещественный и целочисленный, но не наоборот. Допустимо присваивание любого типа данных, кроме файлового.

Формат оператора присваивания:

<Имя переменной> := <Выражение>;

Вместо имени переменной можно указывать элемент массива или поле записи. Отметим, что знак присваивания := отличается от знака равенства = и имеет другой смысл. Знак присваивания означает, что сначала вычисляется значение выражения и потом оно присваивается указанной переменной. Поэтому при условии, что x является числовой переменной и имеет определенное значение, будет допустимой следующая конструкция: x := x + 1;

Пример. Операторы присваивания.

```
Var x, y: real;
n: integer;
stroka: string;
 begin
     n := 17 * n - 1;
     stroka := 'Дата ' +
     DateToStr(Date); x := -12.3 *
     sin(pi / 4);
```

```
  y := 23.789E+3;
end;
```

Выражение. Выражение состоит из операндов и операторов. Операторы находятся между операндами и обозначают действия, которые выполняются над операндами. В качестве операндов выражения можно использовать: переменную, константу, функцию или другое выражение.

Тип выражения определяется типом операндов, входящих в выражение, и зависит от операций, выполняемых над ними. Например, если оба операнда, над которыми выполняется операция сложения, целые, то очевидно, что результат тоже является целым. А если хотя бы один из операндов дробный, то тип результата дробный, даже в том случае, если дробная часть значения выражения равна нулю.

Важно уметь определять тип выражения. При определении типа выражения следует иметь в виду, что тип константы определяется ее видом, а тип переменной задается в инструкции объявления. Например, константы 0.1 и -512 — целого типа (integer), а константы 1.0, 0.0 и 3.2E-05 — вещественного типа (real).

Выводы

1. Для чисел возможны типы: вещественные, целые без знака и со знаком, 16-ричные.

2. Целые числа запоминаются в формате с фиксированной точкой. Наиболее часто используемым является тип **integer**.

3. Вещественные числа запоминаются в формате с плавающей точкой. Для них лучше всего использовать тип **real**.

4. Для работы с короткими строками используются типы **ShortString**, **String [N до 255]**. Это область в статической памяти, назначаемая при компиляции программы.

5. Для работы с длинными строками используются типы **String**, **AnsiString**, **WideString**. Это указатели размером 4 байта, в которые помещается адрес начала строки в динамической памяти, назначаемый при исполнении программы.

Контрольные вопросы

1. Алфавит языка Pascal.
2. Идентификаторы.
3. Что такое комментарии? Для чего они нужны? Как они оформляются в языке Pascal?
4. Привести пример не менее 15 зарезервированных слов.
5. Константы. Их объявление. Константные выражения.
6. Переменные. Их объявление.
7. Форматы представления чисел (с фиксированной и плавающей точкой).
8. Какие типы целых чисел без знака поддерживаются в языке Pascal? Каков их диапазон и формат?
9. Какие типы целых чисел со знаком поддерживаются в Pascal? Каков их диапазон и формат?
10. Какие логические типы данных поддерживаются в языке Pascal?
11. Какие типы вещественных чисел поддерживаются в языке Pascal? Каков их диапазон и формат и точность?
12. Какие символьные типы поддерживаются в языке Pascal?
13. Что такое оператор?
14. Для чего необходим оператор присваивания?
15. Совместимость типов при присвоениях.
16. Последовательность действий при выполнении оператора присваивания.
17. Приоритетность выполнения операций в выражениях.

Лабораторная работа 2. Простейшие программы

Теоретический материал

Консоль — это монитор и клавиатура, рассматриваемые как единое устройство. *Консольное приложение* — программа, предназначенная для работы в операционной системе MS-DOS (или в окне DOS), для которой устройством ввода является клавиатура, а устройством вывода — монитор, работающий в режиме отображения символьной информации (буквы, цифры и специальные знаки).

Достоинство консольных приложений – относительная простота использования. Кроме того, исполняемый exe-файл консольной программы намного меньше по размеру (десятки килобайт) по сравнению с исполняемым файлом windows-приложения такой же программы (сотни килобайт).

Консольные приложения удобны как иллюстрации при рассмотрении общих вопросов программирования, когда надо сосредоточиться на сути проблемы, а также как небольшие утилиты «для внутреннего потребления».

Этапы разработки программы. Выражение «написать программу» отражает только один из этапов создания компьютерной программы, когда разработчик программы (программист) действительно пишет команды (инструкции) на бумаге или при помощи текстового редактора.

Программирование — это процесс создания (разработки) программы, который может быть представлен последовательностью следующих шагов:

1. Спецификация (определение, формулирование требований к программе).
2. Разработка алгоритма.
3. Кодирование (запись алгоритма на языке программирования).
4. Отладка.
5. Тестирование.

<u>Спецификация</u>

Спецификация, определение требований к программе — один из

важнейших этапов, на котором подробно описывается исходная информация, формулируются требования к результату, поведение программы в особых случаях (например, при вводе неверных данных), разрабатываются диалоговые окна, обеспечивающие взаимодействие пользователя и программы.

Разработка алгоритма

На этапе разработки алгоритма необходимо определить последовательность действий, которые надо выполнить для получения результата. Если задача может быть решена несколькими способами и, следовательно, возможны различные варианты алгоритма решения, то программист, используя некоторый критерий, например, скорость решения алгоритма, выбирает наиболее подходящее решение. Результатом этапа разработки алгоритма является подробное словесное описание алгоритма или его блок-схема.

При изображении алгоритма в виде блок-схемы используются специальные символы (рис. 3.1).

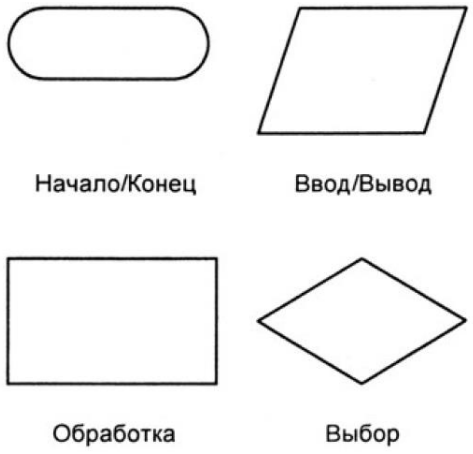


Рис 3.1. Основные символы, используемые для представления алгоритма в виде блок-схемы

Кодирование

После того как определены требования к программе и составлен алгоритм решения, алгоритм записывается на выбранном языке программирования. В результате получается *исходная* программа.

Отладка

Отладка — это процесс поиска и устранения ошибок. Ошибки в программе разделяют на две группы: синтаксические (ошибки в тексте) и алгоритмические. Синтаксические ошибки — наиболее легко устраняемые. Алгоритмические ошибки обнаружить труднее. Этап отладки можно считать законченным, если программа правильно работает на одном-двух наборах входных данных.

Тестирование

Этап тестирования особенно важен, если предполагается, что программой будут пользоваться другие люди. На этом этапе следует проверить, как ведет себя программа на как можно большем количестве входных наборов данных, в том числе и на заведомо неверных.

Перед тем как приступить к созданию консольного приложения, рассмотрим инструкции, обеспечивающие вывод информации на экран и ввод данных с клавиатуры.

Инструкции write и writeln. Инструкция *write* предназначена для вывода на экран монитора сообщений и значений переменных. После слова write в скобках задается список переменных, значения которых должны быть выведены. Кроме имен переменных в список можно включить сообщение — текст, заключенный в одиночные кавычки.

Например:

```
write(Summa);
write('Результат вычислений');
write('Корни уравнения: xl=', xl, ' x2=', x2);
```

После имени переменной через двоеточие можно поместить описание (формат) поля вывода значения переменной.

Для переменной типа Integer формат — это целое число, которое задает ширину поля вывода (количество позиций на экране).

Например, инструкция

write(d:5);

показывает, что для вывода значения переменной d используется 5 позиций.

Если значение переменной такое, что его изображение занимает меньше позиций, чем указано в формате, то перед первой цифрой числа будут выведены пробелы так, чтобы общее количество выведенных символов было равно указанному в формате.

Например, если значение переменной Kol типа integer равно 15, то в результате выполнения инструкции

write('Всего изделий:', Kol:5);

на экран будет выведено:

Всего изделий: 15

Для переменных типа Real формат представляет собой два целых числа, разделенных двоеточием. Первое число определяет ширину поля вывода, второе — количество цифр дробной части числа. Если задать только ширину поля, то на экране появится число, представленное в формате с плавающей точкой.

Например, пусть переменные x1 и x2 типа real имеют значения 13.25 и -0.3401, тогда в результате выполнения инструкции

write('xl=',x1:5:2,' x2=',x2:12)

на экран будет выведено:

x1=13.25 x2=-3.40100E-01

Если ширины поля, указанной в формате, недостаточно для вывода значения переменной, то выводится число в формате с плавающей точкой и десятью цифрами после запятой (все поле вывода в этом случае занимает 17 позиций).

После выполнения инструкции write курсор остается в той позиции экрана, в которую он переместился после вывода последнего символа, выведенного этой инструкцией. Следующая инструкция write начинает вывод именно с этой позиции. Например, в результате выполнения инструкций

```
x:=-2.73;
write('Значение перем');
write('енной:');
write('x=');
write(x:8:5);
```

на экран будет выведено:

Значение переменной: x=-2.73000

Инструкция *writeln* отличается от инструкции write только тем, что после вывода сообщения или значений переменных курсор переводится в начало следующей строки. Например, если значением переменной x1 является число -3.561, а значением переменной x2 — число 10.345, то результатом выполнения инструкций

```
writein('Значения корней уравнения:');
writeln('x1=',x:7:3);
writein('x2=',x:7:3);
```

на экран будет выведено:
Значения корней уравнения:
xl=-3.5610
x2= 10.345

Инструкции read и readln. Инструкция read предназначена для ввода с клавиатуры значений переменных (исходных данных). В общем виде инструкция выглядит следующим образом:

read (Переменная1, Переменная2, ... ПеременнаяN)

где *переменная* — имя переменной, значение которой должно быть введено с клавиатуры во время выполнения программы.

Примеры записи инструкции read: read(a); read(Cena,Kol);

При выполнении инструкции read происходит следующее:

1. Программа приостанавливает свою работу и ждет, пока на клавиатуре будут набраны нужные данные и нажата клавиша <Enter>.

2. После нажатия клавиши <Enter> введенное значение присваивается переменной, имя которой указано в инструкции.

Например, в результате выполнения инструкции read(Temperat); и ввода с клавиатуры строки 21, значением переменной Temperat будет число 21.

Одна инструкция read позволяет получить значения нескольких переменных. При этом вводимые числа должны быть набраны в одной строке и разделены пробелами. Например, если тип переменных a, b и c — real, то в результате выполнения инструкции read(a,b,c); и ввода с клавиатуры строки:

4.5 23 0.17

переменные будут иметь следующие значения: a = 4,5; b = 23,0; c = 0,17.

Если в строке набрано больше чисел, чем задано переменных в инструкции read, то оставшаяся часть строки будет обработана следующей инструкцией read. Например, в результате выполнения инструкций

read(A,B); read(C);

и ввода с клавиатуры строки

10 25 18

переменные получат следующие значения: A=10, B = 25. Инструкция read(C); присвоит переменной C значение 18.

Инструкция readln отличается от инструкции read тем, что после выделения очередного числа из введенной с клавиатуры строки и присваивания его последней переменной из списка инструкции readln, оставшаяся часть строки теряется, и следующая инструкция read или readln будет требовать нового ввода.

Например, в результате выполнения инструкции readln(A,B); read(C); и вводе с клавиатуры строки

10 25 18

переменные получат следующие значения: A=10, B = 25. После чего программа будет ожидать ввода нового числа, чтобы присвоить его переменной C.

Перед каждой инструкцией read или readln следует располагать инструкцию write, для того чтобы подсказать пользователю, какие данные ожидает от него программа. Например, фрагмент программы вычисления стоимости покупки может иметь вид:

```
writeln('Введите исходные данные.');
write('Цена изделия:');
readln(Cena);
write('Количество в партии:');
readln(Kol);
write('Скидка:');
```

```
readln(Skidka);
```

Если тип данных, вводимых с клавиатуры, не соответствует или не может быть приведен к типу переменных, имена которых указаны в инструкции read (readln), то программа аварийно завершает работу (инструкции, следующие за read, не выполняются), и на экран выводится сообщение об ошибке.

Создание консольного приложения.

Начинается консольное приложение инструкцией program, за которой следует имя программы. Сначала оно совпадает с именем проекта «по умолчанию». В момент сохранения проекта оно будет автоматически заменено на имя, под которым программист сохранит проект.

Пример консольного приложения:

```
program demo;
{$APPTYPE CONSOLE}
uses SysUtils;
     begin
     writeln('Проверка');
     end.
```

Следует обратить внимание на то, что консольное приложение создается в Windows, а выполняется как программа DOS. В DOS используется кодировка ASCII, а в Windows — ANSI, буквы русского алфавита в которых имеют разные коды. Это приводит к тому, что вместо сообщений на русском языке консольное приложение выводит «абракадабру». Решить данную проблему можно либо выводя все сообщения на английском языке (что не всегда удобно), либо разработав функцию перекодировки ANSI-строки в строку ASCII.

Компиляция и выполнение проекта. Программа, представленная в видеинструкций языка программирования, называется исходной программой. Она состоит из инструкций, понятных человеку, но не понятных процессору компьютера. Чтобы процессор смог выполнить работу в соответствии с инструкциями исходной программы, исходная программа должна быть переведена на машинный язык — язык команд процессора.

Задачу преобразования исходной программы в машинный код выполняет

специальная программа — *компилятор*.

Компилятор выполняет последовательно две задачи:

1. Проверяет текст исходной программы на отсутствие синтаксических ошибок.

2. Создает (генерирует) исполняемую программу — машинный код.

Следует отметить, что генерация исполняемой программы происходит только в том случае, если в тексте исходной программы нет синтаксических ошибок.

Генерация машинного кода компилятором свидетельствует лишь о том, что в тексте программы нет синтаксических ошибок. Убедиться, что программа работает правильно можно только в процессе ее тестирования — пробных запусках программы и анализе полученных результатов. Например, если в программе вычисления корней квадратного уравнения допущена ошибка в выражении (формуле) вычисления дискриминанта, то, даже если это выражение будет синтаксически верно, программа выдаст неверные значения корней.

При запуске консольного приложения на экране появляется стандартное окно DOS-программы.

Практическое задание1.

1. Создайте новое консольное приложение.
2. Сохраните его в отдельном каталоге Lab3 под именем Zadanie1.dpr. Введите раздел переменных.
4. Опишите две переменных: perem1 типа integer и perem2 типа real.

Указание: вводить по 1-й переменной на каждой строчке, для того, чтобы написать комментарии.

5. Напишите комментарии разными способами к каждой из строк.
6. Введите в тело программы оператор, выводящий на экран предложение (на русском языке), предлагающее пользователю ввести целое число и оператор, считывающий введенное значение в переменную perem1.
7. Откомпилируйте приложение. В случае обнаружения ошибок исправьте их.

8. Запустите программу на выполнение.

Указание. Окно приложения закрывается слишком быстро. Для устранения этого эффекта можно добавить следующую строку в конец программы:

ReadLn; *//задержка экрана до нажатия Enter*

Как видно, вместо русскоязычного предложения на экране появляется непонятный набор символов. Объясните почему?

9. Замените предложение на русском языке англоязычным. Запустите программу на выполнение, введите целое число, завершите выполнение программы.

10. Снова запустите программу на выполнение и введите целое число, выходящее за диапазон значений типа integer. Какова реакция программы? 11. Дополните тело программы оператором, выводящим на экран предупреждение о том, что сейчас будет выведено значение переменной «n» и оператор, выводящий это значение. Запустите программу на выполнение. Какова реакция программы? На каком этапе она происходит?

11. Добавьте в раздел описания целочисленных переменных переменную n. Снова запустите программу на выполнение. Какое значение появилось на экране? Чем это можно объяснить? Какой вывод из этого следует?

Практическое задание2. Реализовать консольное приложение, осуществляющее перевод введенного пользователем веса из функтов в килограммы.

Реализация.

Этап1. Спецификация.

Необходимо разработать программу, позволяющую пользователю вводить вес в фунтах и выводить для него результат перевода этого веса в килограммы. При этом вывод ответа целесообразно реализовать с отдельным указанием количества килограмм и грамм.

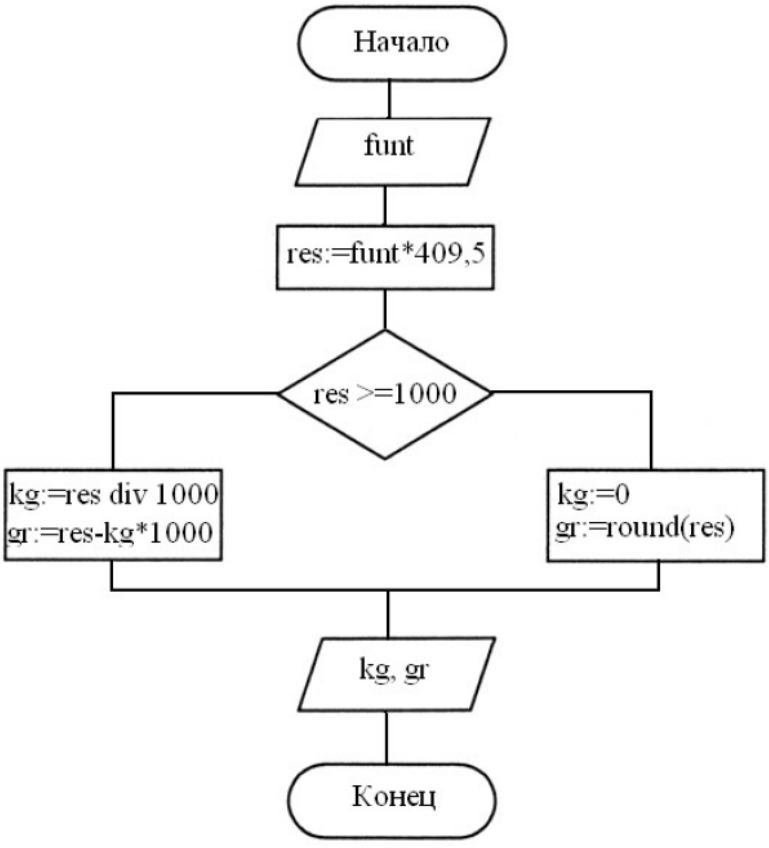
Этап2. Разработка алгоритма.

Для решения задачи введем следующие обозначения: funt – вес в фунтах, res –

результат перевода фунтов в граммы, kg – количество целых килограмм в результате перевода, gr – количество грамм в результате перевода. При этом вводу подлежит величина funt (без этого значения задача не может быть решена), а выводу – величины kg и gr (это искомый результат). Величина res является промежуточной.

Для перевода фунтов в граммы воспользуемся следующим соотношением: один фунт — это 409,5 гр. Следовательно, для перевода введенного пользователем значения из фунтов в килограммы, сначала это значение необходимо умножить на 409,5. Если полученное количество грамм больше 1000, значит, из него необходимо выделить целое количество килограмм, взяв целую часть от деления на 1000, и выделить количество грамм – разность общего количества грамм и количества грамм, входящего в выделенное количество килограмм.

Блок схема алгоритма приведенного словесного алгоритма будет иметь следующий вид:



Этап3. Кодирование.

Создадим новое консольное приложение. Сохраним его под именем Funt_kg.dpr в отдельном каталоге с именем Lab3_Zadanie2.

Как известно, при работе в консольном приложении существуют трудности с выводом на экран сообщений на русском языке. Для решения этой проблемы разработаем функцию перекодировки ANSI-строки в строку ASCII. Назовем эту функцию RUS.

Введем следующие переменные funt и res типа real, kg и gr типа integer. Дополнив основной алгоритм выводом необходимых сообщений для удобства пользователя, основную программу можно записать следующим образом:

Этап4. Отладка.

Откомпилируем созданное приложение. Исправим все найденные ошибки. Сохраним изменения, внесенные в проект.

Этап5. Тестирование.

Запустим созданное приложение на выполнение, введем некоторое значение веса в фунтах, полученный результат проверим с помощью калькулятора.

Успешное завершение этапа5 свидетельствует о том, что созданное приложение готово к использованию.

Доработайте приложение так, чтобы оно удовлетворяло всем требованиям, предъявляемым к программам при сдаче.

Практическое задание3. Создать консольное приложение, реализующее взаимные преобразования типов численных данных в соответствии с вариантами, приведенными в табл.3.1. Нужно преобразовывать численные данные из типа Start в типы Fin1 и Fin2 для вещественных и целых чисел.

Указания. Для хранения исходных числовых значений использовать типизированные константы. Например:

```
const
real_0:real=123.456789;
```

```
int_0:smallint=-1111;
```

Каждый раз данные должны выводиться в виде значения, размера в байтах (отводимого для хранения данного значения) и типа данных.

Числа для наглядности должны отображаться в формате с фиксированной точкой. При этом задается число позиций под выводимое значение, что позволяет позиционировать выводимые данные в строке.

Пояснения. Отличие типизированных констант от нетипизированных состоит в том, что они могут менять свои значения в ходе работы программы, то есть они ведут себя как инициализированные переменные. Поведение типизованных констант может регулироваться директивой компилятора {$J}. По умолчанию действует {$J+}, и типизованная константа может получить новое значение в ходе выполнения программы. Директива {$J-} «превращает» типизованную константу в истинную.

Процесс преобразования можно реализовать используя оператор присваивания, в результате выполнения которого приведение типов левой и правой частей осуществляется автоматически. Для этого достаточно описать переменные необходимых типов и присвоить им значения указанных типизированных констант.

Для контроля размеров данных существует встроенная функция **SizeOf**, возвращающая размер аргумента в байтах.

Типы вещественных чисел

	Start	*Fin_1*	*Fin_2*	*Fin_3*
1	Single	Currency	Extended	Real48
2	Real48	Single	Currency	Double
3	Double	Real48	Single	Real
4	Real	Double	Real48	Single
5	Extended	Real	Double	Real48
6	Comp	Extended	Real	Double
7	Currency	Comp	Extended	Real
8	Single	Currency	Comp	Extended
9	Real48	Single	Currency	Comp
10	Double	Real48	Single	Currency
11	Real	Double	Real48	Single
12	Extended	Real	Double	Real48

13	Comp	Extended	Real	Double
14	Currency	Comp	Extended	Real
15	Real	Currency	Comp	Extended

Типы целых чисел

	Start	*Fin_1*	*Fin_2*	*Fin_3*
1	Byte	Cardinal	Integer	Smallint
2	Word	Byte	Cardinal	Integer
3	LongWord	Word	Byte	LongInt
4	Cardinal	LongWord	Word	Byte
5	Shortint	Cardinal	LongWord	Word
6	Smallint	Shortint	Cardinal	LongWord
7	Integer	Smallint	Shortint	Cardinal
8	LongInt	Integer	Smallint	Shortint
9	Int64	LongInt	Integer	Smallint
10	Word	Int64	LongInt	Integer
11	LongWord	Byte	Int64	LongInt
12	Cardinal	Word	Int64	Int64
13	Shortint	LongWord	Byte	Word
14	Smallint	Cardinal	Word	LongWord
15	Integer	Shortint	LongWord	Cardinal

Выводы

1. Программа на языке Pascal имеет следующую структуру:
   - Заголовок.
   - Разделы объявлений констант, переменных, типов, меток, подпрограмм (процедур и функций).
   - Раздел операторов, помещенный в операторные скобки **begin**…**end**.
   - Точка, как признак конца программы.

2. Для чисел возможны типы: вещественные, целые без знака и со знаком, 16-ричные.

3. Вещественные числа запоминаются в формате с плавающей точкой. Для них лучше всего использовать тип **real**. При переходе к типу с меньшим размером теряется точность. При использовании типов с максимальными размерами требуемая память увеличивается, но точность может не измениться. Для округления результатов используются типы вещественных чисел с ограниченной

дробной частью: **comp** (0 разрядов) и **currency** (4 разряда).

4. Целые числа запоминаются в формате с фиксированной точкой. Для них лучше всего использовать тип **integer**. При переходе к типу с меньшим размером возможна грубая ошибка, если число не умещается в отведенную под него память. Целые числа без знака целесообразно использовать для уменьшения занимаемой памяти.

Контрольные вопросы.

1. Какие приложения называют консольными?
2. В каких случаях целесообразно использовать консольные приложения?
3. Что такое программа?
4. Структура программы.
5. Основные шаги разработки программ.
6. Что такое алгоритм?
7. Чем отличается отладка от тестирования?
8. Константы. Типизированные константы. Константные выражения.
9. Обязательна ли инициализация переменных в программе?
10. Как организован ввод и вывод данных в консольных приложениях?
11. Основные возможности и отличия инструкций ввода.
12. Основные возможности и отличия инструкций вывода.\
13. Как можно организовать вывод предложений на русском языке?
14. Что такое компилятор?
15. Что представляет собой процесс компиляции? Как выполнить компиляцию проекта?

Лабораторная работа 3. Вычисление выражений

Теоретический материал.

Логические операции. Операции отношения. Булевы выражения.

Для данных логического типа справедливы следующие условия:

1) False < True;
2) Succ(False) = True;
3) Pred(True) = False;
4) Ord (False) = 0;
5) Ord (True) = 1.

К величинам логического типа применимы *логические операции*.

Примечание. Результат логической операции имеет логический тип.

Not (*отрицание*) – унарная логическая операция, имеющая самый высокий приоритет выполнения, изменяет значение логического типа на противоположное;

And (*И*) – бинарная логическая операция, которая возвращает значение True, только в том случае, когда оба ее операнда имеют значение True;

Or (ИЛИ) - бинарная логическая операция, которая возвращает значение False, только в том случае, когда оба ее операнда имеют значение False;

Xor (исключающее ИЛИ) - бинарная логическая операция, которая возвращает значение False, когда оба ее операнда имеют значение False, либо оба ее операнда имеют значение True. Операция возвращает True в том случае, когда истинен только один из ее операторов.

Примечание. Приоритет операции And выше, чем приоритет операций Or и Xor.

Операцию And иногда называют логическим умножением, а операцию Or – логическим сложением.

Самый низкий приоритет выполнения имеют операции отношения: <> (*неравно*), = (*равно*), <= (*меньше или равно*), < (*меньше*), >= (*больше или равно*), > (*больше*).

Примечание. Определенный по умолчанию приоритет выполнения

операций можно изменить расставленными круглыми скобками.

Булево выражение – это запись, которая может содержать арифметические выражения, круглые скобки, логические операции и операции отношений.

Значение булево выражения логического типа.

Переменной логического типа может быть присвоено значение булево выражения.

Составной оператор. Условный оператор.

Составной оператор – совокупность последовательно выполняемых операторов, заключенная в операторные скобки.

Begin Оператор1, Оператор2, … , Оператор N **End**;

Составной оператор часто используется для оформления блока операторов внутри другого структурированного оператора. Например, внутри условного оператора.

Условный оператор ***IF*** предназначен для изменения порядок выполнения операторов в зависимости от истинности или ложности некоторого условия. Он предписывает выполнять некоторое действие только в том случае, когда выполняется заданное условие. Это условие записывается в виде логического выражения, а действие, которое нужно выполнить, задается в виде последовательности операторов.

Существует две конструкции оператора ветвления – полная и сокращенная (неполная).

Полное ветвление:

if(<логическое выражение>**) then** <оператор 1> **else** <оператор 2>**;**

В условном операторе сначала вычисляется значение булево выражения. Если значение выражения есть True, то выполняется Оператор1, а Оператор2 пропускается. Если значение булево выражения есть False, то выполняется Оператор2, а Оператор1 пропускается.

Примечание.

1) Оператор1 и Оператор2 часто являются составными операторами.

2) В операторе IF перед ELSE точка с запятой не ставится.

3) Условный оператор управляет только одним оператором, поэтому, если после ключевых слов Then и ELSE требуется произвести более одного действия, то необходимо использовать операторные скобки Begin End.

4) Внутри операторных скобок после каждого оператора точка с запятой ставится.

Если вторая ветвь отсутствует, тогда имеет место сокращенное ветвление.

Сокращенное ветвление:

if(<логическое выражение>**) then** <оператор>**;**

Примечание. Условные операторы могут быть вложенными. При использовании вложенных условных операторов в сокращенной и полной формах, необходимо помнить, что каждая часть Else будет относиться к ближайшей к ней сверху части Then.

Оператор выбора

В случае необходимости разветвить вычислительный процесс в зависимости от выполнения или невыполнения того или иного условия на более чем две ветви используется оператор выбора (случая, селектора, переключателя). Его использование оказывается более удобным по сравнению с использованием оператора if.

Оператор **case** работает следующим образом. Сначала вычисляется значение выражения-селектора, затем обеспечивается реализация того оператора, константа выбора которого равна текущему значению селектора. Если ни одна из констант не равна текущему значению селектора, выполняется оператор, стоящий за словом **ELSE,** при его отсутствии выполняется оператор, стоящий за словом **end**.

```
Case S of
     C1:   <Оператор1>
     C2:   <Оператор2>
     . . . . . .
     CN: <ОператорN>

     Else
     <Оператор>
     End;
```

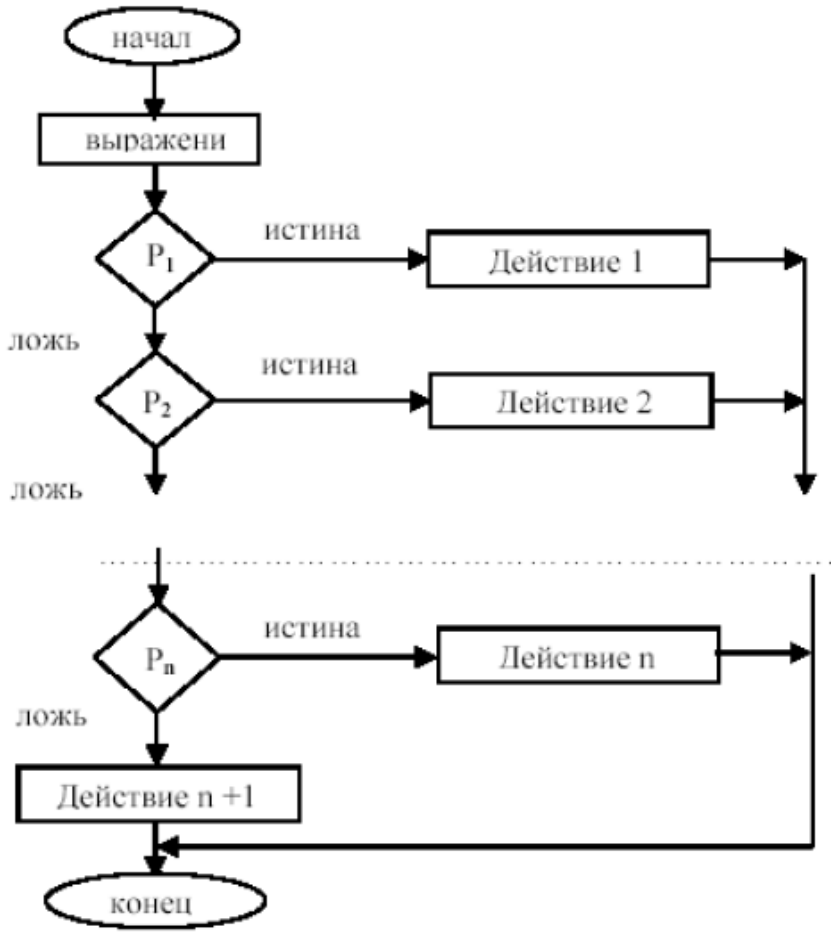
S - выражение порядкового типа значение которого вычисляется;

C1, C2,…,CN – константы, с которыми сравнивается значение выражения S; <Оператор1>, <Оператор2>, <Оператор N> - операторы, из которых выполняется тот, с константой которого совпадает значение выражения S.

Ветвь оператора *else* является необязательной. Если она отсутствует и значение выражения S не совпадает ни с одной константой, весь оператор рассматривается как пустой.

Если для нескольких констант нужно выполнить один и тот же оператор, их можно перечислить через запятую, сопроводив их одним оператором.

Схематически такую конструкцию можно изобразить следующим образом:



Стандартные функции и операции

Abs (x) соответствует **|x|**;

ArcTan (x) соответствует **arctg(x)**;

Cos (x) соответствует **cos(x)**;

Sin (x) соответствует **sin(x)**;

Exp (x) соответствует **ex**;

Ln (x) соответствует **ln(x)**;

Sqr (x) соответствует $\mathbf{x}^2$;

Sqrt (x) соответствует $\sqrt{x}$;

Frac (x) - дробная часть **x**;

Int (x) возвращает целую часть числа;

Round (x) возвращает число, округленное по правилам арифметики;

Trunc(x) возвращает целое число, отбрасывая дробную часть числа x.

Практическое задание1. Реализовать консольное приложение для вычисления значения выражения в соответствии с вариантом.

Указания. Все параметры и переменные являются вещественными числами и подлежат вводу.

Величина π должна быть оформлена в виде константы.

Необходимо учесть, что значение выражения может быть посчитано не для всех значений параметров и переменных.

Все значения вводимых исходных данных и результаты должны выводиться с указанием имен, с четырьмя знаками после запятой (например, Res=3.7854).

Варианты задания

1)
$$y = \begin{cases} |\, x^{t/x} - \sqrt{x/t}\,|, & \text{при } t < 1 \\ \pi\, at^2 \ln t, & \text{при } 1 \le t \le 2 \\ e^{at}\cos bt, & \text{при } t > 2 \end{cases}$$

2) $$q = \begin{cases} \pi x^2 - 7/x^2, & \text{при } x < 1.3 \\ a + x + \dfrac{x^2}{2!} + \dfrac{x^3}{3!} + \dfrac{x^4}{4!}, & \text{при } x = 1.3 \\ tg\left(x + 7\sqrt{a}\right), & \text{при } x > 1.3 \end{cases}$$

3) $$w = \begin{cases} e^{-bx}\sin(ax+b) - \sqrt{|bx+a|}, & \text{при } x < 1.2 \\ \pi a / x + \sqrt{x^2 - 1}, & \text{при } x = 1.2 \\ (a+bx)/\sqrt{x^2 + c}, & \text{при } x > 1.2 \end{cases}$$

4) $$r = \begin{cases} \sqrt{x^2 + b} - \dfrac{b^2 \sin^3(x+a)}{x}, & \text{при } x < 1.4 \\ ax^3 + 7\pi\sqrt{x^2 - 1}, & \text{при } x = 1.4 \\ (a+bx)/\sqrt{x^2 + a}, & \text{при } x > 1.4 \end{cases}$$

5) $$t = \begin{cases} \dfrac{a}{(x-b)}\cos^2 x, & \text{при } x \le 1 \\ \dfrac{bx^2 - a}{e^{ax} - 1}, & \text{при } 1 < x < 2 \\ \pi\, tgx, & \text{при } x > 2 \end{cases}$$

6) $$u = \begin{cases} \pi\, x\sqrt{x-a}, & \text{при } x > a \\ e^{-ax}\sin(ax), & \text{при } x < a \\ \dfrac{x^2(x+1)}{b} - \sin^2(x+a), & \text{при } x = a \end{cases}$$

7) $$p = \begin{cases} bx + \pi\ln(bx), & \text{при } bx < 1 \\ \sin^3\left(\left(x^2 + a\right)^2\right) - \sqrt{\dfrac{x}{b}}, & \text{при } bx = 1 \\ bx + \ln(bx), & \text{при } x > 1 \end{cases}$$

8) $$s = \begin{cases} b\sin(\ln x), & \text{при } x > 3.5 \\ \cos^2(\sqrt{x - \pi}), & \text{при } x < 3.5 \\ btg^2 x - \dfrac{a}{\sin^2(x/a)}, & \text{при } x = 3.5 \end{cases}$$

9) $$f = \begin{cases} \ln(x - b), & \text{при } x > 1 \\ \pi\sin^2\left(\sqrt{|ax|}\right), & \text{при } x < 1 \\ \ln(a + x^2) + \sin^2(x/b), & \text{при } x = 1 \end{cases}$$

10) $$d = \begin{cases} a\ln x + \sqrt{|x|}, & \text{при } x < 1 \\ 2b\cos x + 3\pi x^2, & \text{при } x > 1 \\ \dfrac{a^{2x} + b^{-x}\cos((a+b)x)}{x+1}, & \text{при } x = 1 \end{cases}$$

11) $$g = \begin{cases} \pi\left(\ln^2(x) + x^2\right)/\sqrt{x+1}, & \text{при } x < 0.5 \\ \sqrt{ax\sin(2x) + e^{-2x}(x+b)}, & \text{при } x = 0.5 \\ 1/a\cos x + b\sin^2 x, & \text{при } x > 0.5 \end{cases}$$

12) $$h = \begin{cases} \dfrac{a+b}{e^x + \cos x}, & \text{при } x < 2.8 \\ xctg^2(x-a) + \dfrac{b}{\sqrt{x+b}}, & \text{при } 2.8 \le x \le 5 \\ e^x/a + \pi\sin x, & \text{при } x > 5 \end{cases}$$

13) $$k = \begin{cases} \dfrac{a}{i} + bi^2 + c, & \text{при } i < 4 \\ \dfrac{a^2 i + e^{-x}\cos(bi)}{bi - e^{-i}\sin(bi) + 1}, & \text{при } 4 \le i \le 6 \\ \pi\, ai/(i-8) + bi^2, & \text{при } i > 6 \end{cases}$$

14) $$v = \begin{cases} b\pi\sqrt{\sin\left(\dfrac{i^2+1}{n}\right)}, & \text{при } \dfrac{i^2+1}{n} < 0 \\ \cos(i + 1/n) + 3i^2, & \text{при } 0 < \dfrac{i^2+1}{n} < 1 \\ \dfrac{2\cos(i - \pi/a)}{1/2 + \sin^2 n}, & \text{при } \dfrac{i^2+1}{n} > 1 \end{cases}$$

15) $$z = \begin{cases} \sqrt{at^2 + b\sin t + 1}, & \text{при } t < 0.1 \\ e^{-at}\sqrt{t+1} + e^{-bt}\sqrt{t+1.5}, & \text{при } t = 0.1 \\ \pi\sqrt{a/bt^2 + \cos t}, & \text{при } t > 0.1 \end{cases}$$

16) $$y = \begin{cases} ctg(z), & \text{при } t < 1 \\ at^2\ln(\pi t), & \text{при } 1 \le t \le 2 \\ (t-x)\dfrac{t - \dfrac{z}{t-x}}{1 + (t-x)^2}, & \text{при } t > 2 \end{cases}$$

17) $$q = \begin{cases} x^2 - 7\pi/x^2, & \text{при } x < 1.3 \\ ax^3 + 7\sqrt{x}, & \text{при } x = 1.3 \\ x(\sin x - \cos a), & \text{при } x > 1.3 \end{cases}$$

18) $$w = \begin{cases} (a + bx)/\sqrt{x^2 - 1}, & \text{при } x < 1.2 \\ b\sin(ax^2\cos(ax)) - 1, & \text{при } x = 1.2 \\ ax^2 + bx + \pi c, & \text{при } x > 1.2 \end{cases}$$

19) $$r = \begin{cases} \cos^2(x^3) - x/\sqrt{a^2+b^2}, \text{при } x < 1.4 \\ \pi\, ax^3 + \sqrt{x^2-1}, \text{при } x = 1.4 \\ (a+bx)/\sqrt{x^2-b}, \text{ при } x > 1.4 \end{cases}$$

20) $$t = \begin{cases} \pi \cos^2 x, \text{ при } x \le 1 \\ (x-2)^2 + 6, \text{при } 1 < x < 2 \\ x^3 tg^2\left((x+b)^2\right) + \dfrac{a}{\sqrt{x+b}}, \text{при } x > 2 \end{cases}$$

21) $$u = \begin{cases} \sqrt{\dfrac{xb}{a}} + \cos\left((x+b)^3\right), \text{ при } x > a \\ e^{-ax}\sin(ax), \text{при } x < a \\ \pi\, x\sin(ax), \text{ при } x = a \end{cases}$$

22) $$p = \begin{cases} bx/a - \ln(bx), \text{ при } bx < 1 \\ \pi\, b, \text{при } bx = 1 \\ \dfrac{x^2}{a} + \cos\left((x+b)^3\right), \text{ при } x > 1 \end{cases}$$

23) $$s = \begin{cases} \sqrt{\sin(\ln x)}, \text{ при } x > 3.5 \\ \pi\, \cos^2(bx), \text{при } x < 3.5 \\ ae^{\sqrt{a}}\cos(bx/a), \text{при } x = 3.5 \end{cases}$$

24) $$f = \begin{cases} \pi\, \ln(x+1), \text{ при } x > 1 \\ \sin^2(|\,bx\,|), \text{при } x < 1 \\ e^{-ax}\dfrac{x+\sqrt{x+a}}{x-\ln(|\,x-b\,|)}, \text{при } x = 1 \end{cases}$$

25) $$i = \begin{cases} b\ln x + \pi\sqrt{|\,x\,|}, \text{ при } x > 1 \\ 2a\cos x + 3x^2, \text{при } x = 1 \\ \sqrt{x^2+b} - b^2\sin(x+a)/x, \text{при } x < 1 \end{cases}$$

Практическое задание2. Реализовать консольное приложение для вывода произвольных сообщений для различных дней недели.

Указания. Программа должна содержать запрос у пользователя номера дня недели в пределах 1..7. После ввода номера в этом интервале с помощью оператора выбора формируется текстовое сообщение об этом дне недели. Если пользователь вводит номер 0, работа приложения завершается. Ввод чисел, отличных от 0..7, программа рассматривает, как ошибку, и просит повторить ввод.

Выводы

1. Для ветвлений используется оператор

if <условие> **then** <оператор1> **else** <оператор2>.

2. Возможен сокращенный оператор ветвлений

if <условие> **then** <оператор>.

3. Для множественного ветвления можно применить оператор выбора **case… of**. Селектор этого оператора должен быть порядкового типа (чаще всего целочисленного).

4. Если в ветвях **then** или **else** нужно выполнить несколько операторов, то их нужно объединить в один составной оператор с помощью операторных скобок **begin…end.**

Контрольные вопросы.

1. Что такое стандартная функция?
2. Какие стандартные математические функции реализованы в Pascal? Как их можно использовать для вычисления выражений?
3. Что представляет собой составной оператор? Как ограничиваются операторы объединенные в составной оператор?
4. Что собой представляют порядковые типы данных?
5. Что такое перечисляемые типы?
6. Какие значения могут иметь данные логического типа?
7. Какое служебное слово используется для описания данных логического типа?
8. Как осуществляется программирование логических выражений?
9. Назначение, формы записи и порядок выполнения условного оператора if.
10. Особенности использования вложенных условных операторов.
11. Каковы отличия оператора выбора case от оператора условия if?
12. Для чего служит ключ выбора и какого он может быть типа?
13. Сколько меток может быть перед оператором в списке выбора?

Лабораторная работа 4. Операторы цикла

Теоретический материал.

При реализации многократного повторения некоторых операций линейной конструкцией необходимо снова и снова повторять одни и те же операторы. Для более компактной реализации этих операций во всех языках используются циклические конструкции, суть которых заключается в том, что вместо многократного переписывания одних и тех же строк программы управление в нужном месте передается предыдущим операторам с тем, чтобы они повторялись.

Существует три оператора циклов.

Цикл с предусловием

Цикл с предусловием используется, как правило, в тех случаях, когда заранее неизвестно число повторений цикла.

Форма записи оператора цикла с предусловием:

- до начала цикла необходимо указать начальное значение управляющей переменной;
- тело цикла заключается в операторные скобки;
- необходимо предусмотреть изменение управляющей переменной.

Выход из цикла осуществляется, если некоторое логическое выражение окажется ложным. Так как истинность логического выражения проверяется в начале каждого повтора, ***тело цикла может не выполняться ни разу.*** Структура оператора цикла имеет

while <условие> **do** <тело цикла>; Блок-

схема цикла с предусловием:

Здесь WHILE (Пока), DO (выполнить) – служебные слова.

Цикл с постусловием

Оператор цикла **REPEAT** организует выполнение цикла, состоящего из любого числа операторов, с неизвестным заранее числом повторений.

Форма записи оператора цикла с постусловием:

- необходимо предусмотреть задание начального значения управляющей переменной;
- необходимо предусмотреть изменение управляющей переменной.

Тело цикла выполняется ***хотя бы один раз***. Выход из цикла осуществляется при истинности некоторого логического выражения.

Структура оператора:

repeat <тело цикла> **until** <условие>;

Здесь REPEAT(повторять) , UNTIL (до тех пор пока) – служебные слова.

Цикл с параметром

Оператор цикла **FOR** организует выполнение одного оператора заранее известное число раз. Существует два варианта оператора.

for <переменная>:=<нач. значение> **to** <кон. значение> **do**<оператор>;

for <переменная>:=<нач. значение> **downto** <кон. значение> **do** <оператор>;

В операторе цикла с параметром сначала вычисляются начальное и конечное значения параметра. Затем параметру цикла присваивается начальное значение. После проводится проверка необходимости продолжения цикла или его остановки. Цикл прекращается, когда значение параметра цикла превышает конечное значение. Если цикл не заканчивается, то выполняются операторы из тела цикла. Затем переменной цикла присваивается следующее большее значение (в первом случае) или следующее меньшее (во втором варианте).

При программировании циклов с параметром необходимо помнить следующие ***правила организации цикла***:

1) параметр цикла, начальное и конечное значения, должны быть одинакового типа, их тип может быть любым скалярным типом (стандартным, перечисляемым, ограниченным), кроме вещественного;

2) очередное значение параметра вычисляется автоматически, для целого типа шаг изменения значения параметра цикла равен 1 при TO и – 1 при DOWNTO;

3) запрещено изменять внутри тела цикла значение управляющей переменной цикла;

4) цикл не выполняется вообще, если начальное значение больше (при DOWNTO – меньше), чем конечное;

5) после служебного слова DO может стоять только один оператор; если в цикле нужно выполнить группу операторов, то их заключают в скобки BEGIN … END;

Циклы называются ***вложенными***, если в тело одного из них входит другой оператор цикла. Цикл, содержащий в себе другой цикл, называется ***внешним***. Цикл, содержащийся в другом цикле, называется ***внутренним***.

Внутренний цикл должен завершиться раньше, чем внешний. Внутренний и внешний циклы могут быть как одного так и разного вида. В качестве параметров вложенных циклов for недопустимо использовать одну и ту же переменную.

Практическое задание1. Реализовать консольное приложение для вычисления значения суммы (произведения) членов бесконечного ряда с заданной точностью ε в соответствии с вариантом. На печать вывести значение суммы (произведения) и число членов ряда, вошедших в сумму (произведение).

Указания. Для вычисления значения суммы (произведения) использовать оператор цикла с предусловием.

Значение точности вычислений ε должно быть оформлено в виде константы.

Для получения значения π использовать встроенную константу pi.

Варианты заданий

1) $s = 2\left(\frac{\sin x}{1} - \frac{\sin 2x}{2} + \frac{\sin 3x}{3} + \ldots\right)$ $\varepsilon = 10^{-5}$

2) $s = ch(x) = 1 + \frac{x^2}{2!} + \frac{x^4}{4!} + \ldots + \frac{x^{2n}}{(2n)!} + \ldots$ $\varepsilon = 0.5 \cdot 10^{-4}$

3) $s = 1 + \frac{x}{1!}\cos\frac{\pi}{4} + \frac{x^2}{2!}\cos\frac{2\pi}{4} + \ldots + \frac{x^n}{n!}\cos\left(\frac{\pi}{4}n\right) + \ldots$ $\varepsilon = 0.3 \cdot 10^{-3}$

4) $s = \frac{\pi}{2} - \frac{4}{\pi}\left(\cos x + \frac{\cos 3x}{3^2} + \frac{\cos 5x}{5^2} + \ldots\right)$ $\varepsilon = 10^{-4}$

5) $s = 1 + \frac{\cos x}{1!} + \frac{\cos 2x}{2!} + \ldots + \frac{\cos nx}{n!} + \ldots$ $\varepsilon = 10^{-6}$

6) $s = \frac{x\cos(\pi/3)}{1} + \frac{x^2\cos(2\pi/3)}{2} + \ldots + \frac{x^n\cos(n\pi/3)}{n} + \ldots$ $\varepsilon = 0.2\cdot 10^{-5}$

7) $s = 4\left(x - \frac{x}{3} + \ldots + (-1)^n\frac{x}{2n+1} + \ldots\right)$ $\varepsilon = 0.1\cdot 10^{-3}$

8) $s = \cos\left(\frac{\pi}{6}\right) = 1 - \frac{(\pi/6)^2}{2!} + \frac{(\pi/6)^4}{4!} - \ldots + (-1)^n\frac{(\pi/6)^{2n}}{(2n)!} + \ldots$ $\varepsilon = 10^{-7}$

9) $s = x - \frac{x^3}{3} + \frac{x^5}{5} - \ldots + (-1)^{n-1}\frac{x^{2n-1}}{(2n-1)} + \ldots$ $\varepsilon = 0.5\cdot 10^{-4}$

10) $s = 1 + \frac{x^2}{2!} - \frac{3x^4}{4!} + \ldots + (-1)^n\frac{(2n-1)x^{2n}}{(2n)!} + \ldots$ $\varepsilon = 10^{-8}$

11) $s = \frac{x^3}{5} - \frac{x^5}{17} + \ldots + (-1)^{n+1}\frac{x^{2n+1}}{(4n^2+1)} + \ldots$ $\varepsilon = 0.4\cdot 10^{-3}$

12) $s = arctg(x) = \frac{1}{x} + \frac{1}{3x^3} - \ldots + (-1)^n\frac{1}{(2n+1)x^{2n+1}} + \ldots$ $\varepsilon = 10^{-6}$

13) $s = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + (-1)^n\frac{(x/2)^{2n}}{n!(n+3)!} + \ldots$ $\varepsilon = 10^{-7}$

14) $s = \left(1 - \frac{(x-1)}{2}\right)\cdot\left(1 - \frac{(x-1)}{6}\right)\cdot\left(1 - \frac{(x-1)}{12}\right)\cdot\ldots$ $\varepsilon = 10^{-8}$

15) $s = \cos(x) = \left(1 - \frac{4x^2}{\pi}\right)\cdot\left(1 - \frac{4x^2}{9\pi}\right)\cdot\left(1 - \frac{4x^2}{25\pi}\right)\cdot\ldots$ $\varepsilon = 10^{-9}$

16) $s = \exp(x) = 1 + \frac{x}{1!} + \frac{x^2}{2!} + \ldots + \frac{x^n}{n!} + \ldots$ $\varepsilon = 10^{-5}$

17) $s = sh(x) = x + \frac{x^3}{3!} + \frac{x^5}{5!} + \ldots + \frac{x^{2n+1}}{(2n+1)!} + \ldots$ $\varepsilon = 0.5\cdot 10^{-4}$

18) $s = \frac{\sin 3x}{3!} + \frac{\sin 5x}{5!} + \frac{\sin 7x}{7!} + \ldots + \frac{\sin((2n+1)x)}{(2n+1)!} + \ldots$ $\varepsilon = 0.3\cdot 10^{-3}$

19) $s = \frac{2}{\pi} - \frac{4}{\pi}\left(\frac{\cos 2x}{1\cdot 3} + \frac{\cos 4x}{3\cdot 5} + \frac{\cos 6x}{5\cdot 7} + \ldots\right)$ $\varepsilon = 10^{-4}$

20) $s = \frac{\cos x}{1} - \frac{\cos 2x}{4} + \frac{\cos 3x}{9} - \ldots + (-1)^{n+1}\frac{\cos nx}{n^2} + \ldots$ $\varepsilon = 10^{-6}$

21) $s = \frac{2}{3}\sin 2x - \frac{3}{8}\sin 3x + \ldots + (-1)^n\frac{n}{n^2-1}\sin nx + \ldots$ $\varepsilon = 0.2\cdot 10^{-5}$

22) $s = \cos(x) = 1 - \frac{x^2}{2!} + \frac{x^{42}}{4!} - \ldots + (-1)^n\frac{x^{2n}}{(2n)!} + \ldots$ $\varepsilon = 0.1\cdot 10^{-3}$

23) $s = -\frac{(2x)^2}{2!} + \frac{(2x)^4}{4!} - \ldots + (-1)^n \frac{(2x)^{2n}}{(2n)!} + \ldots$ $\varepsilon = 10^{-7}$

24) $s = \frac{\sin(x)}{x} = 1 - \frac{x^2}{3!} + \frac{x^4}{5!} - \ldots + (-1)^n \frac{x^{2n}}{(2n+1)!} + \ldots$ $\varepsilon = 0.5 \cdot 10^{-4}$

25) $s = \frac{x^3}{3} - \frac{x^5}{15} + \ldots + (-1)^{n+1} \frac{x^{2n+1}}{(4n^2 - 1)} + \ldots$ $\varepsilon = 10^{-8}$

Практическое задание2. Реализовать консольное приложение для вычисления значения суммы (произведения) членов бесконечного ряда из практического задания 1 с использованием оператора цикла с постусловием. На печать вывести значение суммы (произведения) и число членов ряда, вошедших в сумму.

Указания. Значение точности вычислений ε должно быть оформлено в виде константы.

Для получения значения π использовать встроенную константу pi.

Практическое задание3. Реализовать консольное приложение для вычисления значения суммы (произведения) членов ряда из практического задания 1, сделав его конечным. На печать вывести значение суммы (произведения) членов ряда.

Указания. Количество членов ряда задает пользователь.

Для вычисления значения суммы (произведения) использовать оператор цикла с параметром.

Для получения значения π использовать встроенную константу pi.

Практическое задание4. Реализовать консольное приложение для вычисления таблицы значений суммы (произведения) членов ряда из практического задания 3 для x, изменяющегося в диапазоне 0..A с шагом h.

Указания. Количество членов ряда, значение A и шаг h задает пользователь. Для получения значения π использовать встроенную константу pi.

Выводы

1. Для организации циклов с неизвестным числом повторений применяются

операторы:

while...do,

repeat...until.

2. Для выполнения в цикле **while...do** группы операторов необходимо превратить ее в один составной оператор с помощью операторных скобок **begin…end**.

3. При выполнении цикла **repeat...until** в теле цикла может находиться множество операторов. Составной оператор не требуется.

4. Для организации циклов с известным числом повторений применяются операторы:

Инкрементный:

for i:=<начальное значение> **to** <конечное значение> **do** <оператор>.

В нем переменная индекса цикла I увеличивается с шагом 1.

Декрементный:

for i:=<начальное значение> **downto** <конечное значение> do <оператор>.

В нем переменная индекса цикла I уменьшается с шагом 1.

5. Переменная индекса цикла должна быть порядкового типа (например, **Integer**).

6. Если в теле цикла используется много операций, то их нужно оформить, как составной оператор, с помощью операторных скобок **begin…end.**

Контрольные вопросы.

1. Какие программы называются циклическими?

2. Как исполняется цикл с предусловием?

3. Выполнится ли оператор в теле цикла с предусловием, если изначально значение булева выражения ложно?

4. В каких случаях используется цикл с предусловием?

5. Почему в цикле с постусловием серия операторов из тела цикла будет выполнена хотя бы один раз?

6. В каком случае выполнение цикла с постусловием прекращается?

7. Переменная какого типа может служить параметром цикла?

8. Какой тип должны иметь начальное и конечное значения в цикле с параметром?

9. Можно ли в оператор цикла for включать действия по изменению параметра?

10. Чему равно значение параметра цикла for после завершения цикла?

11. Как исполняется цикл по убыванию с параметром?

12. Какие циклы называются вложенными?

13. Какой цикл называется внутренним? Внешним?

14. Какому правилу должны удовлетворять вложенные циклы?

15. Могут ли внутренний и внешний циклы быть циклами одного вида? Разного вида?

16. Прямое вычисление значения суммы членов бесконечного ряда.

17. Условие выхода из цикла при вычислении значения суммы членов бесконечного ряда.

Лабораторная работа 5. Поиск ошибок. Отладка

Классификация ошибок

Ошибки, которые могут быть в программе, принято делить на три группы:

- синтаксические;
- ошибки времени выполнения;
- алгоритмические.

Синтаксические ошибки, их также называют *ошибками времени компиляции* (Compile-time error), наиболее легко устранимы. Их обнаруживает компилятор, а программисту остается только внести изменения в текст программы и выполнить повторную компиляцию.

Ошибки времени выполнения (run-time errors) или *исключения* (exception), тоже, как правило, легко устранимы. Они обычно проявляются уже при первых запусках программы и во время тестирования. В большинстве случаев причинами исключений являются неверные исходные данные.

При возникновении ошибки в программе, среда разработки прерывает работу программы, и на экране появляется диалоговое окно, которое содержит сообщение об ошибке и информацию о типе (классе) ошибки.

После возникновения ошибки программист может либо прервать выполнение программы, либо продолжить ее выполнение, например, наблюдая результат выполнения каждой инструкции.

При разработке программы необходимо постараться предусмотреть все возможные варианты некорректных действий пользователя, которые могут привести к возникновению ошибок времени выполнения (исключения), и обеспечить способы защиты от них.

С *алгоритмическими ошибками* дело обстоит иначе. Компиляция программы, в которой есть алгоритмическая ошибка, завершается успешно. При пробных запусках программа ведет себя нормально, однако при анализе результата выясняется, что он неверный. Для того чтобы устранить алгоритмическую ошибку, приходится анализировать алгоритм, вручную «проводить» его выполнение.

Отладчик

Интегрированная среда разработки предоставляет программисту мощное средство поиска и устранения ошибок в программе — отладчик. Отладчик позволяет выполнять *трассировку* программы, наблюдать значения переменных, контролировать выводимые программой данные.

Наблюдение значений переменных

Во время отладки, в частности, при выполнении программы по шагам, довольно часто бывает полезно знать, чему равно значение той или иной переменной. Отладчик позволяет наблюдать значения переменных программы.

Для того чтобы во время выполнения программы по шагам иметь возможность контролировать значение переменной, нужно добавить имя этой переменной в список наблюдаемых элементов (Watch List).

Практическое задание1. На примере консольного приложения с помощью отладочной печати проконтролировать значения последовательности s[i], а также продемонстрировать встроенные средства отладки среды Delphi.

Указания. В работе использовать учебную программу, которая по паре вводимых чисел c и d в цикле для i=1..10 вычисляет значения пары других параметров a=(c+d)*i, b=(c-d)*i. Функция **Ample** вычисляет квадратный корень из суммы квадратов целых частей от a и b, и эти значения суммируются в s. По завершении цикла вычисляется среднее арифметическое sm от выходных параметров функции **Ample**. Функция **Ample** оформлена в виде подпрограммы.
Отладочная печать - вывод значений переменных в окно приложения при прогоне программы.

Листинг программы:

```
program Prg_16_1;
     {$APPTYPE CONSOLE}
SysUtils, RusTrans; // Ссылка на модули

var
     a,b,c,d,s,f,sm: real; i:
     integer;

function Ample(x,y:real):real;
  var x1,y1,f: real;
```

```
            begin
(m5)          x1:=Int(x);
(m6)          y1:=Int(y);
             Result:=sqrt(x1*x1+y1*y1);
           end;

         begin
              writeln(Rus('Введите c,d'));
              readln(c,d);
              writeln;
              s:=0;
              for i:=1 to 10 do
              begin
(m1)               a:=(c+d)*i; (m2)
                   b:=(c-d)*i; (m3)
                   f:=Ample(a,b);
(m4)               s:=s+f;
                    writeln('s[',i,']=',s:8:3);
              end;
              sm:=s/10;
              writeln;
              writeln('Resultat=',sm:8:3);
              readln;
              end.
```

Варианты заданий:

№	Переменные окна Watches	Строки останова	Выражение для окна Evaluate/ Modify
1	a, b, f	m1,m2,m3	a + b
2	c, d, s	m4,m5,m6	a * b
3	b, d, f	m2,m3,m4	a / b
4	a, d, c	m1,m5,m6	a + b
5	b, a, s	m3,m4,m5	a * b
6	c, d, f	m1,m2,m6	a – b
7	d, c, a	m4,m5,m6	a / b
8	f, d, b	m1,m2,m3	a + b
9	s, f, c	m5,m6,m1	a * b
10	a, s, f	m4,m2,m3	a – b
11	b, a, s	m6,m1,m2	a / b
12	c, d, f	m3,m4,m5	a * b
13	s, d, c	m1,m3,m5	a – b
14	a, c, f	m4,m5,m3	a / b

Выводы

1. Данные о состоянии программы можно получить, используя отладочную

печать или отладочные средства ИСР.

2. Для контроля значений переменных можно использовать окно отладки «Watches». В нем при каждом останове программы можно наблюдать текущие значения переменных или выражений с их использованием.

3. Для трассировки программы предназначено пошаговое выполнение программы в версиях «Step Over» и «Trace Into». В первом случае подпрограмма исполняется за один шаг, а во втором происходит построчное исполнение кодов подпрограммы.

4. Для контроля вызова подпрограмм используется окно «Call Stack».

Контрольные вопросы.

1. Ошибки. Классификация ошибок.
2. Синтаксические ошибки.
3. Ошибки времени выполнения.
4. Алгоритмические ошибки.
5. Отладчик.
6. Отладочная печать и ее использование при тестировании программ.
7. Пошаговое выполнение программы. Режимы трассировки.
8. Трассировка в режиме «Trace to Next Source Line».
9. Трассировка в режиме «Run Until Return».
10. Отладка методом точек останова.
11. Способы наблюдения значений переменных.
12. Способы оценки и модификации значения выражений.

8. Структура и использование окна «**Call stack**».

10. Назначение протокола событий и его использование при отладки приложений.

Лабораторная работа 6. Обработка исключительных ситуаций

Теоретический материал.

В программе во время ее работы могут возникать ошибки, причиной которых, как правило, являются действия пользователя. Например, пользователь может ввести неверные данные или удалить нужный программе файл.

Нарушение в работе программы называется *исключением*. Обработку исключений (ошибок) берет на себя автоматически добавляемый в выполняемую программу код, который обеспечивает, в том числе, вывод информационного сообщения.

Структурная обработка исключительных ситуаций

Структурная обработка исключительных ситуаций - это система, позволяющая программисту при возникновении ошибки (исключительной ситуации) связаться с кодом программы, подготовленным для обработки такой ошибки. Это выполняется с помощью языковых конструкций, которые как бы «охраняют» фрагмент кода программы и определяют обработчики ошибок, которые будут вызываться, если что-то пойдет не так в «охраняемом» участке кода. В данном случае понятие исключительной ситуации относится к языку и не нужно его путать с системными исключительными ситуациями (hardware exceptions), такими как General Protection Fault. Исключительные ситуации независимы от аппаратных средств, не используют прерываний и используются для обработки ошибочных состояний, с которыми подпрограмма не готова иметь дело. Системные исключительные ситуации могут быть перехвачены и преобразованы в языковые исключительные ситуации, но это только одно из применений языковых исключительных ситуаций.

Структурная обработка исключительной ситуации замещает ручную обработку ошибок автоматической, сгенерированной компилятором системой уведомления.

Данная система называется *структурной,* поскольку обработка ошибок определяется областью «защищенного» кода; такие области могут быть вложенными. Выполнение программы не может перейти на произвольный участок

кода; выполнение программы может перейти только на обработчик исключительной ситуации активной программы.

Модель исключительных ситуаций

Модель исключительных ситуаций в Object Pascal является невозобновляемой (*non-resumable*). При возникновении исключительной ситуации уже нельзя будет вернуться в точку, где она возникла, для продолжения выполнения программы (это позволяет сделать возобновляемая (*resumable*) модель). Невозобновляемые исключительные ситуации разрушают стек, поскольку они сканируют его в поисках обработчика; в возобновляемой модели необходимо сохранять стек, состояние регистров процессора в точке возникновения ошибки и выполнять поиск обработчика и его выполнение в отдельном стеке. Возобновляемую систему обработки исключительных ситуаций гораздо труднее создать и применять, нежели невозобновляемую.

Синтаксис обработки исключительных ситуаций

Новое ключевое слово, добавленное в язык Object Pascal - **try**. Оно используется для обозначения первой части защищенного участка кода. Существует два типа защищенных участков:

try..except

try..finally

Первый тип используется для обработки исключительных ситуаций. Его синтаксис:

```
try
    // здесь инструкции, выполнение которых может вызвать исключение
except      // начало секции обработки исключений
on TunИсключения1 do Обработка1;
on TunИсключения2 do Обработка2;
on TunИсключенияJ do ОбработкаJ;
else
    // здесь инструкции обработки остальных исключений
end;
```

где:

try — ключевое слово, обозначающее, что далее следуют инструкции, при выполнении которых возможно возникновение исключений, и что обработку этих исключений берет на себя программа;

except — ключевое слово, обозначающее начало секции обработки исключений. Инструкции этой секции будут выполнены, если в программе возникнет ошибка;

on — ключевое слово, за которым следует тип исключения, обработку которого выполняет инструкция, следующая за do. Если возникшая исключительная ситуация подходит по типу к указанному после *on*, то выполнение программы переходит сюда (на код после *do*). Исключительная ситуация подходит в том случае, если она того же класса, что указан в *on*, либо является его потомком. Например, в случае *on EFileNotFound* обрабатываться будет ситуация, когда файл не найден. А в случае *on EFileIO* – все ошибки при работе с файлами, в том числе и предыдущая ситуация.

else — ключевое слово, за которым следуют инструкции, обеспечивающие обработку исключений, тип которых не указаны в секции except.

После того, как найден подходящий обработчик ошибки, поиск оканчивается. После выполнения кода обработчика, программа продолжает выполняться с оператора, стоящего после слова *end* блока *try..except*.

Конструкцию try...finally используют в тех случаях, когда существуют действия, которые обязательно надо выполнить до завершения программы. Код, расположенный в части finally, выполняется в любом случае, даже если возникает исключительная ситуация. Если ошибки не возникло, то последовательно выполняются все операторы секций.

```
try
    // здесь инструкции, выполнение которых может вызвать исключение
finally
//операторы, которые должны быть выполнены даже в случае ошибки
end;
```

Конструкцию try...finally можно включить в блок try...except.

Это позволяет выполнить обязательные операторы секции finally и обработать исключение операторами секции except. Оба типа конструкций можно использовать в любом месте, допускается вложенность любой глубины.

Пример обработки исключительных ситуаций

Рассмотрим программу вычисления силы тока (i) по введенным значениям напряжения (u) и сопротивления (r).

В данной программе исключения, например, могут возникнуть при вычислении величины тока. Если пользователь задаст, что сопротивление равно нулю, то при выполнении инструкции i:=u/r возникает Исключение EZeroDivide.

Реализуем обработку исключения в виде вывода соответствующего сообщения с помощью оператора try..exept.

Листинг 1. Обработка исключения типа *EZeroDivide*

```
program SilaToka;
{$APPTYPE CONSOLE}
Uses  SysUtils;
var
u: real; // напряжение
r: real; // сопротивление
i: real; // ток
begin
write('Введите через пробел напряжение и сопротивление:');
 try     // инструкции, которые могут вызвать исключение (ошибку)
  read(u,r);
  i := u/r;
 except   // секция обработки исключений
  on EZeroDivide do // деление на ноль
    Writeln('Сопротивление не может быть равно нулю');
 end;
writeln(i,' A');
readln;
end.
```

Примечание. При тестировании приложений желательно пользоваться созданным exe-файлом. Запускать приложения из среды разработки можно, но при этом надо учитывать, что при возникновении исключительной ситуации прежде всего сработает система защиты среды. При появлении системного сообщения его надо прочитать, окно сообщения закрыть и выполнить команду **Run** для

продолжения работы.

В связи с тем, что после выполнения кода обработчика, программа продолжает выполняться с оператора, стоящего после слова *end* блока *try..except* в приведенном примере вывод значения силы тока будет осуществляться вне зависимости от того, была исключительная ситуация или нет. Для того, чтобы избежать вывод значения силы тока в случае возникновения исключительной ситуации внесем следующие изменения в программу:

Листинг 2. Обработка исключения типа *EZeroDivide*

```
program SilaToka;
{$APPTYPE CONSOLE}
Uses  SysUtils;
var
u: real; // напряжение
r: real; // сопротивление
i: real; // ток
f: boolean; //флаг
begin
write('Введите через пробел напряжение и сопротивление:');
f:=true;
 try     // инструкции, которые могут вызвать исключение (ошибку)
  read(u,r);
  i := u/r;
 except   // секция обработки исключений
  on EZeroDivide do // деление на ноль
    begin
     Writeln('Сопротивление не может быть равно нулю!');
     f:=false;
    end;
 end;
if f then writeln(i,' A');
readln;
end.
```

Реализуем обработку исключения в виде вывода соответствующего сообщения с помощью оператора try..finally.

Листинг 3. Обработка исключения типа *EZeroDivide*

```
program SilaToka;
{$APPTYPE CONSOLE}
Uses SysUtils;
var
u: real; // напряжение
r: real; // сопротивление
i: real; // ток
f: boolean; //флаг
```

```
begin
write('Введите с новой строки напряжение и сопротивления:');
f:=true;
 try    // инструкции, которые могут вызвать исключение (ошибку)
  read(u);
  read(r);
  if r< >0 then i := u/r
      else
       begin
        f:=false;
        Writeln('Сопротивление не может быть равно нулю!');
       end;
 finally
  if f then writeln(i,' A');
 end;
end.
```

Оператор **try**…**finally** не обрабатывает исключение. Поэтому при возникновении ошибки срабатывает системный обработчик исключения и выдает свое сообщение.

Практическое задание 1. Реализовать консольное приложение для вычисления значений функций $f_1(x)$ и $f_2(x)$ при x=i*h, где i – целое число, принимающее значения из интервала I_{min} .. I_{max}, h – шаг вычислений. Необходимо отслеживать следующие исключения: деление на 0 (исключение **EZeroDivide**), некорректный ввод пользователя (например, вместо числа введена буква) и исключение **EOverflow** с использованием оператора **try**… **except.**

Указания. Встроенный отладчик **Delphi** перехватывает все исключительные ситуации и формирует свои сообщения о них в диалоговом окне, которое пользователь перед продолжением программы должен закрыть. Чтобы избежать этого, следует отменить использование отладчика, сбросив флаг «Встроенный отладчик» в окне команды Сервис => Опции отладчика.

Чтобы отслеживать исключения в операциях с числами, необходимо в окне команды Проект => Опции... на вкладке «Компилятор» установить флаги директив компилятора:

- проверка области
- проверка ввода/вывода
- проверка переполнения

Величины I_{min} и I_{max} вводятся пользователем.

Обработка исключительной ситуации реализуется в виде вывода сообщения с указанием того, какая именно произошла ошибка и при каком значении i.

При работе с программой рекомендуется менять h_1 и h_2, чтобы получать разные результаты и условия возможного возникновения исключений.

Для получения значения использовать встроенную константу pi.

Варианты заданий:

№	Функция 1			Функция 2		
	$f_1(x)$	h_1	Тип	$f_2(x)$	h_2	Тип
1	1/x	π/5	real	200+x	10	byte
2	1/sin(x)	π /5	single	(300+x)*50	100	word
3	1/cos(x)	π /5	extended	300+x	10	shortint
4	tg(x)	π /5	real	(200+x)*200	10	smallint
5	ctg(x)	π /5	single	100+x	15	byte
6	1+1/x	π /10	extended	(300-x)*50	20	word
7	2+1/sin(x)	π /10	real	50+x	30	shortint
8	3+1/cos(x)	π /10	single	200+x	10	smallint
9	4+tg(x)	π /10	extended	250+x	50	byte
10	5+ctg(x)	π /10	real	200-x	30	word
11	1-1/x	π /20	single	200+x	10	shortint
12	2-1/sin(x)	π /20	extended	(200+x)*200	10	smallint
13	3-1/cos(x)	π /20	real	400+x	100	byte
14	4- tg(x)	π /20	single	200+x	10	word
15	5-ctg(x)	π /20	extended	100-x	30	shortint

Практическое задание 2. Реализовать консольное приложение для вычисления значений функций $f_1(x)$ и $f_2(x)$ из практического задания 1 с возможностью отслеживать те же исключения с использованием оператора **try**… **finally.**

Выводы

1. При исполнении программ возможны исключения (ситуации с ошибками). Примеры исключений - деление на ноль **EZeroDivide**, выход значений за пределы допустимого диапазона **ERangeError**.

2. Встроенный отладчик среды разработки перехватывает все исключительные ситуации и формирует свои сообщения о них в диалоговом окне, которое

пользователь перед продолжением программы должен закрыть. Чтобы избежать этого, следует отменить использование отладчика.

3. Для обработки исключений используются обработчики исключений (глобальные и локальные). Глобальные обработчики определены в ИСР и вызываются автоматически, если отладчик подключен. Локальные обработчики создает пользователь.

4. Для локальной обработки исключений определен оператор **try**…**except**…**end**. В секции **try** размещаются операторы, в которых могут возникнуть исключения, а в секции **except** обработчики исключений. При обнаружении исключения исполнение операторов секции **try** прекращается, управление передается нужному обработчику в секции **except**.

5. Обработчик исключения имеет формат **on** <исключение> **do** <оператор>

6. Для обнаружения исключений определен оператор **try**…**finally**. В секции **try** размещаются операторы, в которых могут возникнуть исключения, а в секции **finally** операторы, исполняемые всегда. При возникновении исключения исполнение операторов секции **try** прекращается, управление передается первому оператору секции **finally**. После исполнения секции finally в случае ошибки появиться сообщение системного обработчика исключений.

Контрольные вопросы

1. Что такое исключение?

2. Оператор **try**…**except**. Назначение, структура и применение.

3. В каких случаях в код включают блок **try...finally**?

4. В процессе тестирования проект был запущен на выполнение из среды разработки. Проверялось поведение проекта при некорректном вводе данных. В программе была предусмотрена обработка такой ситуации: должно было появиться окно с сообщением об ошибке и предложением повторить ввод. Однако появилось окно с другим сообщением, причём на английском языке. Почему?

5. Предопределенные исключения.

6. Как отключить перехват исключения системой.

Лабораторная работа 7. Создание windows-приложений

Создание приложения

Можно считать хорошим приложение, которое:

имеет простой, удобный, интуитивно понятный интерфейс, со всеми присущими Windows атрибутами: окнами, кнопками, меню и т.д.;

- управляется как мышью, так и клавиатурой;
- отказоустойчиво, корректно обрабатывает любые ошибки пользователя;
- работает быстро;
- имеет хорошую справочную систему;
- обеспечивает обмен данными с другими приложениями; может (при необходимости) использовать средства мультимедиа; может работать с базами данных.

Этапы создания приложения. Разработка windows-приложения состоит из двух этапов: создания интерфейса приложения и определения его функциональности.

Интерфейс определяет способ взаимодействия пользователя и приложения: какие применяются окна, каким образом пользователь управляет приложением. Интерфейс создаётся путём размещения на форме компонентов. Функциональность определяется процедурами, которые выполняются при возникновении определённых событий, происходящих при действиях пользователя.

Создание интерфейса приложения. Интерфейс приложения определяется компонентами, которые разработчик выбирает из палитры компонентов и помещает на форму. При проектировании интерфейса приложения действует принцип WYSIWYG (What You See Is What You Get) – что видите, то и получите. Так как компонент очень много, то они в палитре распределены по группам.

Определение функциональности проекта. Приложения Windows управляются событиями. Программа Windows способна управлять многими альтернативными действиями, которые может совершать пользователь. Windows переводит эти действия, например, нажатие кнопки мыши, вызов меню опций и

нажатия клавиш, в события. Программа Windows в дальнейшем имеет дело с этим событием.

Функциональность приложения определяется реакциями на возможные события. Для этого разрабатывают процедуры, которые вызываются при наступлении соответствующих событий.

Формирование кода. Код пишется в окне Редактора кода, который имеет встроенные средства, существенно облегчаюшие работу. К инструментальным средствам, используемым при записи кода, относятся: **Code Insight**, **Code Explorer**, **Object Browser**, механизм навигации в коде.

Возможности Подсказчика **Code Insight** целесообразно использовать даже в самых простых программах, так как это сокращает время записи и, главное, гарантирует корректность введённых выражений.

Если записать имя компонента, поставить точку и немного подождать, то появится длинный список всех свойств и методов класса, к которому принадлежит компонент. Чтобы быстрее найти нужное название, желательно записать первые символы. Затем надо выбрать строку щелчком мыши и нажать на клавишу **Enter** (или дважды щёлкнуть левой кнопкой мыши по нужной строке): соответствующее имя будет вставлено в код. Повторный вывод Подсказчика выполняется комбинацией клавиш **Ctrl+пробел**. Эта же клавиатурная комбинация позволяет вывести **Code Insight** на экран при отключении автоматического режима работы. По умолчанию строки списка упорядочены по категориям. При желании их можно отсортировать по алфавиту, выбрав команду **Sort by Name** из контекстного меню. Комбинацию клавиш **Ctrl+пробел** можно использовать после записи переменной, оператора присваивания, в пустой строке. Во всех случаях появится подсказка.

Если **Code Insight** работает в автоматическом режиме, то при записи процедур и функций после того, как поставлена открывающаяся круглая скобка, появится список параметров и их типов. Повторный вывод подсказки выполняется комбинацией клавиш **Shift+Ctrl+пробел**.

Иногда полезно воспользоваться имеющимися в **Code Insight** шаблонами стандартных структур языка Pascal. Перечень шаблонов выводится клавишами

Ctrl+J.

Для быстрого перемещения по коду можно использовать закладки, которые устанавливаются командой контекстного меню **Toggle Bookmark**. Переход на нужную закладку выполняется командой контекстного меню **Goto Bookmarks**. Очень удобно организован переход между объявлениями методов в интерфейсной секции и их реализацией. Для этой цели используются клавиши **Ctrl+Shift** и клавиша со стрелкой вверх или вниз (в зависимости от нужного направления перемещения).

Практическое задание 1. Ознакомиться с компонентами групп Standard, Additional, Win32, System, Samples, их свойствами, методами и событиями на которые они могут реагировать.

Практическое задание 2. Реализовать приложение, эмулирующее внешний вид заданной экранной формы.

Указание. Для выполнения данного задания необходимо лишь разместить необходимые компоненты на форме так, чтобы добиться максимального сходства с указанным стандартным окном.

Созданное приложение должно иметь иконку, отличающуюся от стандартной.

Варианты задания:

1. Пункт меню OpenOffice Writer: Формат-Страница (Format-Page). Закладки: Organizer, Page, Background, Header, Footer.
2. Пункт меню OpenOffice Writer: Настройка-Параметры (Tools-Options).
3. Пункт меню Turdo Delphi: «Свойства проекта» (Project-Options). Закладки: Forms, Application, Compiler, Version Info.
4. Пункт меню Turbo Delphi: «Добавить в проект» (Add to Project).
5. Пункт меню OpenOffice Writer: Формат-Стили-Новый (Format-Styles and Formatting-New). Закладки: Numbering, Tabs, Drop Caps, Background, Condition.
6. Пункт меню OpenOffice Writer: Формат- Абзац (Format-Paragraph). Закладки: Indents & Spacing, Alignment, Text Flow, Numbering, Tabs.

7. Пункт меню OpenOffice Writer: Вставка- Символ (Insert – Special Characters).

8. Пункт меню OpenOffice Writer: Вставка- Оглавление и указатели (Insert – Indexes and Tables - Indexes and Tables).

9. Калькулятор с расширенными возможностями.

10. Пункт меню Winrar: Параметры (Options - Settings). Закладки: General, Compression, Paths, Viewer, Security.

11. Пункт меню OpenOffice Writer: Файл – Печать (File - Print).

12. Свойства «Мой компьютер». Закладки: Общие, Имя компьютера, Оборудование, Дополнительно, Восстановление системы.

13. Свойства «Сетевое окружение».

14. Свойства экрана.

15. Свойства ярлыка.

Практическое задание 3. Дополнить приложение, реализованное в практическом задании1 необходимыми обработчиками событий, для реализации указанных действий в соответствии с вариантом.

Варианты задания:

1. В результате установки флажка в переключателе «Header on» на закладке «Header» все компоненты становятся активными, в результате снятия флажка – неактивными. Каждое перемещение курсора мышки над прямоугольником, расположенным на закладке Background приводит к увеличению его текущей высоты и ширины на 10 пикселей. Одинарный щелчок правой кнопкой мыши по этому прямоугольнику приводит к уменьшению его текущей высоты и ширины на 10 пикселей. Одинарный щелчок левой клавишей мыши по кнопке «Cancel» приводит к исчезновению раздела «Contains» закладки «Organizer» и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению раздела и возврату прежнего заголовка кнопки. Двойной щелчок левой клавишей мыши по форме приводит к завершению работы приложения.

2. Одинарный щелчок левой клавишей мыши по любому узлу

иерархического дерева приводит к замене информации об OpenOffice.org на информацию, содержащую краткие сведения о компоненте, используемом для создания таблиц (сеток). Перемещение курсора мыши над изображением приводит к возврату информации об OpenOffice.org. Двойной щелчок левой кнопкой мыши по справочной информации приводит к ее смещению относительно предыдущего расположения вниз на 5 пикселей. Щелчок правой кнопкой мыши по справочной информации приводит к ее смещению относительно текущего расположения вверх на 5 пикселей. Одинарный щелчок левой клавишей мыши по кнопке «Cancel» приводит к исчезновению всех компонентов кроме дерева и кнопок, изменению размеров формы так, чтобы были видны (без полосы прокрутки) все оставшиеся компоненты, но не было пустого места (при необходимости, с переносом этих компонентов) и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к возврату прежнего вида формы и заголовка кнопки. Щелчок правой кнопкой мыши по форме приводит к завершению работы приложения.

3. Одинарный щелчок левой клавишей мыши по кнопке «>» на закладке «Forms» приводит к исчезновению всего текста из поля «Auto-create forms» и его появлению в поле «Available forms». Одинарный щелчок левой клавишей мыши по кнопке «<» приводит к исчезновению всего текста из поля «Available forms» и его появлению в поле «Auto-create forms». Ввод любого символа в поле «Title» на закладке «Application» приводит к следующему изменению изображения на этой же закладке: если оно было сжато и отображалось целиком, то необходимо отменить сжатие, если же сжатия не было и при этом видна лишь часть изображения, то его необходимо сжать. Одинарный щелчок левой клавишей мыши по кнопке «Cancel» приводит к исчезновению раздела «Syntax options» на закладке «Compiler» и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению раздела и возврату прежнего заголовка кнопки. Установка флажка в переключателе «Default» приводит к завершению работы приложения.

4. Одинарный щелчок правой клавишей мыши по пункту «Мой компьютер» приводит к выводу в центральной части формы списка всех логических дисков, существующих в локальной сети СамГУ с указанием рода информации, которая хранится на каждом из них (информация о каждом диске должна выводиться в отдельной строке). Двойной щелчок левой клавишей мыши по пункту «Мой компьютер» приводит к очищению центральной части формы. В результате выбора нового элемента в выпадающем списке «Имя файла» высота и ширина изображения, соответствующего пункту «Недавние документы» должны уменьшаться относительно их текущих значений на 5 пикселей. Перемещение курсора мыши над изображением «Мои документы» приводит к увеличению высоты и ширины изображения, соответствующего пункту «Недавние документы» относительно их текущих значений на 5 пикселей. Одинарный щелчок левой клавишей мыши по кнопке «Cancel» приводит к исчезновению элементов, находящихся в верхней части окна: надписи «Папка», выпадающего списка, находящегося рядом с ней, а также четырех пиктограмм, обеспечивающих переходы к последней просмотренной папке, на один уровень вверх, к меню «Вид», а также создание новой папки, и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению всех этих элементов и возврату прежнего заголовка кнопки. Щелчок правой кнопки мыши по центральной части окна приводит к завершению работы приложения.

5. В результате установки флажка в поле «Condition Style» на закладке «Condition» все компоненты становятся активными, в результате снятия флажка – неактивными. Выбор конкретного цвета из палитры цветов приводит к изменению цвета прямоугольника, расположенного рядом с ней. Одинарный щелчок левой клавишей мыши по кнопке «Cancel» приводит к исчезновению надписи «Numbering Style» и расположенного рядом выпадающего списка на закладке «Numbering», а также к замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к

появлению указанных элементов на 10 пикселей ниже их последнего местоположения и возврату прежнего заголовка кнопки. Перемещение курсора мыши над надписью «Numbering Style» приводит к ее перемещению и расположенного рядом выпадающего списка вверх относительно их текущего местоположения на 10 пикселей. Двойной щелчок левой клавишей мыши по форме приводит к завершению работы приложения.

6. В свободном поле на закладке «Tabs» должны содержаться краткие сведения о компоненте, используемом для создания «бегунка». При установке переключателя на значении Left этот текст должен выравниваться по левому краю, а на значении Right – по правому. Перемещение курсора над изображением на закладке «Indents & Spacing» приводит к его смещению относительно предыдущего положения вниз на 10 пикселей. Двойной щелчок левой клавишей мыши по этому изображению приводит его смещению относительно предыдущего положения вверх на 10 пикселей. Одинарный щелчок левой клавишей мыши по кнопке «Cancel» приводит к исчезновению раздела «Contents» закладки «Drop Caps» и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению раздела и возврату прежнего заголовка кнопки. Установка флажка в переключателе «Activate» на закладке «Indents & Spacing» приводит к завершению работы приложения.

7. Выбор любого символа в таблице приводит к появлению под таблицей краткой информации о компоненте, используемом для создания иерархического дерева (без использования свойства Visible). Перемещение курсора мыши над появившейся информацией приводит к ее стиранию. Выбор любого значения из списка «Sabset» приводит к увеличению высоты и ширины изображения символа, находящегося справа от таблицы, на 10 пикселей, повторный выбор значения из этого списка приводит к уменьшению его размеров на 5 пикселей. Выбор любого значения из списка «Font» приводит к смещению изображения символа, находящегося справа от таблицы относительно его начального положения вниз и влево на 5 пикселей, повторный выбор значения из этого

списка приводит к возвращению элемента в первоначальную позицию. Одинарный щелчок левой клавишей мыши по кнопке «Help» приводит к исчезновению таблицы символов и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению таблицы и возврату прежнего заголовка кнопки. Двойной щелчок левой клавишей мыши по форме приводит к завершению работы приложения.

8. В результате одинарного щелчка левой клавишей мыши по кнопке «Help» все компоненты раздела «Create Form» на закладке «Index/Table» становятся неактивными, а заголовок кнопки заменяется на «Активизировать». Повторный щелчок по этой кнопке делает все компоненты активными и возвращает прежний заголовок кнопки. В результате установки флажка в поле «Preview» изображение слева исчезает, все остальные компоненты перемещаются к левому краю формы и размер формы уменьшается так, чтобы без полосы прокрутки были видны все оставшиеся компоненты, и не было пустого места. Снятие флажка возвращает все в начальное состояние. Двойной щелчок левой клавишей мыши по изображению приводит к завершению работы приложения.

9. В результате установки флажка в поле «Радианы» в строке ввода появляется сообщение, содержащее информацию о том с помощью какого компонента можно реализовать выпадающий список. В результате установки флажка в поле «Градусы» строка ввода очищается. Нажатие любой из кнопок, на которых изображены цифры, приводит к увеличению размеров формы на то количество пикселей, которое указано на кнопке. Нажатие кнопки «Backspace» возвращает начальные размеры формы. Выбор пункта меню «Вид – Обычный» приводит к исчезновению всех кнопок, на которых указаны какие-либо операции или функции кроме +, -, *, / и смене заголовка формы на «Калькулятор Обыкновенный», повторный выбор этого пункта меню приводит появлению всех компонентов и возврату прежнего заголовка формы. Выбор пункта меню «Справка – О» программе приводит к завершению программы.

10. Нажатие кнопки «Browse» на закладке «Compression» приводит к

появлению в находящемся рядом поле сообщения о том, с помощью какого компонента можно объединить несколько компонентов в одну группу. Ввод любого символа в это поле приводит к его очищению. Установка или снятие флажка в переключателе «Restore last working folder on start-up» на закладке «Paths» приводит к смещению всей группы «Start-up folder» относительно ее текущего положения вниз на 5 пикселей. Нажатие на кнопку «Browse» возвращает всю группу в начальное положение. Одинарный щелчок левой клавишей мыши по кнопке «Cancel» приводит к исчезновению раздела «ToolBar» закладки «General» и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению раздела и возврату прежнего заголовка кнопки. Двойной щелчок левой клавишей мыши по форме приводит к завершению работы приложения.

11. В результате установки флажка в поле «Print to File» в центральной части окна (без использования свойства Visible) выводится надпись, содержащая краткую информацию о компоненте «ColorBox». Снятие флажка стирает эту надпись. Одинарный щелчок правой клавишей мыши по кнопке «Options» приводит к смещению выпадающего списка и кнопки «Properties» вниз относительно их текущего положения на 10 пикселей. Ввод любого символа в поле «Pages» возвращает оба компонента в начальное положение. Одинарный щелчок левой клавишей мыши по кнопке «Cancel» приводит к исчезновению раздела «Copies» и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению раздела и возврату прежнего заголовка кнопки. Перемещение курсора мыши над надписью «Printer» приводит к завершению работы приложения.

12. Щелчок любой клавишей мышки в произвольном месте закладки «Общие» приводит к появлению (без использования свойства Visible) на свободном месте краткой информации о компоненте, с помощью которого реализуются кнопки-переключатели. Перемещение курсора мыши над появившейся информацией приводит к ее стиранию. Попытка изменения

положения «бегунка» на закладке Восстановление системы приводит к уменьшению ширины и высоты раздела Состояние на 2 пикселя. Двойной щелчок левой клавишей мыши по форме приводит к увеличению ширины и высоты этого раздела на 2 пикселя. Одинарный щелчок левой клавишей мыши по кнопке «Изменить» на закладке «Имя компьютера» приводит к исчезновению изображения компьютера на этой закладке и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению изображения и возврату прежнего заголовка кнопки. Щелчок правой кнопкой мыши по кнопке «ОК» приводит к завершению работы приложения.

13. Одинарный щелчок левой клавишей мыши по надписи «Создание нового подключения» в группе «Сетевые задачи» приводит к тому, что эта надпись выводится в одну строку, повторный щелчок по ней возвращает вывод надписи в две строки. Одинарный щелчок правой клавишей мыши по кнопке «Вперед» приводит к увеличению центральной части окна относительно ее текущего размера на 10 пикселей. Одинарный щелчок правой клавишей мыши по кнопке «Назад» приводит к уменьшению центральной части окна относительно ее текущего размера на 10 пикселей. Одинарный щелчок левой клавишей мыши по кнопке «Переход» (находящейся рядом с адресной строкой) приводит к исчезновению меню и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению меню и возврату прежнего заголовка кнопки. Выбор пункта меню «Файл – Закрыть» приводит к завершению работы приложения.

14. Перемещение курсора мыши над кнопкой «Сохранить» на закладке «Темы» делает форму прозрачной. Двойной щелчок левой клавишей мыши по изображению на закладке «Темы» возвращает первоначальный вид формы. Щелчок правой кнопкой мыши по «бегунку» на закладке «Параметры» приводит к уменьшению ширины и высоты изображения на этой закладке на 2 пикселя, а щелчок левой клавишей мыши приводит к увеличению ширины и высоты изображения на 2 пикселя. Одинарный щелчок левой клавишей мыши по

кнопке «Отмена» приводит к исчезновению изображения монитора на закладке «Рабочий стол» и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению изображения и возврату прежнего заголовка кнопки. Двойной щелчок левой клавишей мыши по форме приводит к завершению работы приложения.

15. Ввод любой буквы в поле «Объект» на закладке «Ярлык» приводит к выводу в поле «Комментарий» на этой же закладки сообщения о том, с помощью какого компонента можно сделать несколько закладок на форме, а двойной щелчок левой клавишей мыши в поле «Комментарий» приводит к его очищению. Одинарный щелчок правой клавишей мыши по кнопке «Дополнительно» на закладке «Общие» приводит к смещению всех компонентов, находящихся левее нее вниз относительно их текущего положения на 3 пикселя, а перемещение курсора мыши над изображением на этой закладке приводит к смещению этих компонентов вверх относительно их текущего положения на 3 пикселя. Одинарный щелчок левой клавишей мыши по кнопке «Отмена» приводит к исчезновению раздела «Режим совместимости» на закладке «Совместимость» и замене существующего заголовка кнопки на заголовок «Показать» («Show»), повторный щелчок по этой кнопке приводит к появлению изображения и возврату прежнего заголовка кнопки. Установка флажка в переключателе «Скрытый» на закладке «Общие» приводит к завершению работы приложения.

Контрольные вопросы

1. Из каких файлов состоит приложение?
2. Какую структуру имеет файл проекта?
3. Какую структуру имеет модуль?
4. Можно ли в среде разработки одновременно открыть два проекта?
5. Как удалить из проекта форму?
6. Как добавить в хранилище созданную форму?
7. Какое расширение может иметь исполняемый файл, полученный после обработки проекта?

8. Какая форма по умолчанию становится главной формой проекта?

9. Как сделать главной нужную форму?

10. Что понимают под разработкой интерфейса приложения?

11. Что понимают под функциональностью приложения?

12. Что такое компонент?

13. Чем отличаются визуальные компоненты от невизуальных?

14. Как можно задать точное расположение компонентов?

15. Какие используются команды для выравнивания группы компонент?

16. Для чего предназначен Инспектор объектов? Как пользоваться Инспектором объектов?

17. Что такое свойство? Какие существуют типы свойств?

18. Почему некоторых свойств нет в Инспекторе объектов?

19. Почему свойство Name обязательно должно иметь какое-либо значение?

20. Перечислите и охарактеризуйте общие свойства компонентов группы Standard.

21. Как сделать компонент невидимым? Недоступным?

22. Какой тип имеет свойство Visible?

23. Что задаёт свойство TabOrder?

24. Как определить номер выбранной записи в списке компонента ListBox?

25. Как создать многострочную надпись?

26. В чем преимущество компонента GroupBox перед RadioGroup?

27. Какие компоненты фиксируют нажатое состояние?

28. Какой индикатор допускает два или три состояния?

29. Как определить какая строка выбрана и какой индикатор нажат в CheckBox?

30. Перечислите и охарактеризуйте общие свойства компонентов группы Samples.

31. Что определяет свойство меню GroupIndex?

32. Как реагирует раздел меню со свойством Enabled=false на событие OnClick?

33. Перечислите и охарактеризуйте общие свойства компонентов группы Additional.

34. Какое свойство управляет возможностью редактирования on-line в компонентах Edit и MaskEdit?

35. Чем отличается компонент BitButton от Button?

36. Перечислите и охарактеризуйте общие свойства компонентов группы Win32.

37. Что задает свойство PageControl.PageIndex?

38. Перечислите и охарактеризуйте общие свойства компонентов группы System.

39. Как сделать компонент невидимым через 10 с после его активизации, используя компонент Timer?

40. Как создать всплывающую подсказку для компонента?

41. Чем отличаются управляющие элементы-окна (компоненты контейнеры) и рисуемые управляющие элементы? Приведите примеры.

42. Способен ли получить фокус ввода компонент **Memo** (**ListBox, ComboBox**, **DrawGrid, StaticTerxt, UpDown, Timer**)?

43. Что такое событие? Обработчик события? Обработка события?

44. Как создать заготовку процедуры обработчика события?

45. Как формируется имя процедуры-обработчика?

46. Что произойдёт, если в инспекторе объектов изменить свойство Name компонента, для которого созданы обработчики событий?

47. Какой параметр всегда присутствует в процедурах обработчиках событий?

48. Что такое событие по умолчанию? Как для такого события можно создать обработчик?

49. Чем отличаются события OnChanging от OnChange?

50. Когда и для каких компонентов фиксируется событие OnChange?

51. Из каких событий состоит событие OnClick?

52. Из каких событий состоит событие OnDblClick?

53. Когда фиксируется событие OnEnter?

54. Когда фиксируется событие OnExit?

55. Чем отличаются события OnKeyDown от OnKeyPress?

56. Что такое форма?

57. Перечислите и охарактеризуйте основные свойства формы.

58. Какое значение имеет по умолчанию свойство FormStyle?

59. Какое окно появится на экране, если заданы приведённые ниже свойства?

```
form1.BorderStyle:=bsSingle;
form1.BorderIcons:=[biSystemMenu, biHelp];
```

60. В каком из приведённых вариантов будет присутствовать кнопка для сворачивания окна:

1) `form1.BorderStyle:=bsNone;`

2)
```
form1.BorderStyle:=bsNone;
form1.BorderIcons:=[biSystemMenu, biMinimize];
```

3)
```
form1.BorderStyle:=bsSizeable;
form1.BorderIcons:=[biSystemMenu, biMinimize];
```

4) `form1.BorderStyle:=bsSizeableform1.BorderIcons:=[biMinimize];`

5)
```
form1.BorderStyle:=bsDialog;
form1.BorderIcons:=[biSystemMenu, biMinimize];
```

6) `form1.BorderStyle:=bsDialog;`

61. Перечислите и охарактеризуйте основные события, на которые может реагировать форма.

62. В обработчике какого события задают программно размеры формы?

63. Как создать не прямоугольную форму?

64. Что такое файл формы? Какова его структура?

65. Проанализируйте приведённый ниже код. Выясните, что и как делает программа. Подумайте, как можно избавиться от замеченных недостатков.

```
unit Unit1;
interface
uses
Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms, Dialogs, StdCtrls, ExtCtrls;
type
TForm1 = class(TForm)
Label1: TLabel;
Image1: TImage;
Memo1: TMemo;
procedure Image1MouseDown(Sender:TObject; Button:TMouseButton; Shift: TShiftState; X, Y:
Integer);
procedure FormCreate(Sender: TObject);
```

```
var
Form1: TForm1;
implementation
{$R *.dfm}
procedure TForm1.Image1MouseDown(Sender: TObject; Button:TMouseButton; Shift: TShiftState; X,
Y: Integer);
Var Lx,Ly:integer;
begin
Lx:=Image1.Width div 2; Ly:=Image1.Height div 2;
if (x<Lx) and (y<Ly) then
                        begin Label1.Caption:='Меланхолик';
                                Memo1.Lines.LoadFromFile('f2.txt')
                        end;
if (x>Lx) and (y<Ly) then
                        begin Label1.Caption:='Холерик';
                                Memo1.Lines.LoadFromFile('f1.txt')
                        end;
if (x<Lx) and (y>Ly) then
                                begin Label1.Caption:='Флегматик';
                                Memo1.Lines.LoadFromFile('f3.txt')
                                end;
if (x>Lx) and (y>Ly) then
                        begin Label1.Caption:='Сангвиник';
                                Memo1.Lines.LoadFromFile('f4.txt')
                                end;
end;
procedure TForm1.FormCreate(Sender: TObject);
begin
Image1.Picture.LoadFromFile('Pic.bmp');
Label1.Caption:='';
Memo1.Clear;
end;
end.
```

Лабораторная работа 8. Ввод-вывод данных. Работа с датой и временем

Теоретический материал.

Ввод данных

Для ввода данных можно использовать:

- реализующие диалоги функции **InputBox** и **InputQuery**;
- редакторы **Edit**, **LabeledEdit**, **MaskEdit**, **Memo**;
- компоненты для ввода целых чисел **UpDown**, **SpinEdit**;
- компоненты для работы со списками строк **ListBox**, **ComboBox**;
- компоненты-таблицы **StringGrid**;
- переключатели **CheckBox**, **RadioGroup**, **RadioButton**;

Данные можно вводить с клавиатуры, загружать из файла, формировать программно (случайным образом или по определённым правилам).

Наиболее просто программа может получить исходные данные из окна ввода или из поля редактирования (компонент **Edit**).

Контроль вводимых данных. При вводе с клавиатуры необходимо учитывать, что многие компоненты, предназначенные для ввода данных, работают со строками. Поэтому при вводе чисел необходимо использовать функции для перевода строки в число. До использования функций преобразования строки в число желательно проверить, задана ли строка, так как эти функции с пустой строкой не работают.

Обязательное требование: к обработке данных приступать только после корректного ввода.

Наиболее общим решением проблемы контроля вводимых данных является использование имеющихсяв среде разработки средств обработки исключительных ситуаций, то есть формирование защищённого блока и обработка возникающих исключений. Причём, защищённые блоки надо формировать так, чтобы было понятно, где возникла ошибка. Нельзя все поля ввода заключать в один блок. Однако этот универсальный подход не всегда удобен, так

как предусматривает прерывание естественного хода обработки данных. В ряде случаев желательно контролировать данные непосредственно при вводе, то есть разрешать вводить только определённые символы.

При использовании нескольких полей ввода или больших объёмов данных необходимо так организовать их контроль, чтобы пользователю не пришлось полностью повторять ввод, если ошибка была сделана где-то в конце ввода.

В любом случае желательно использовать такие компоненты, которые уменьшают количество ошибок. Если есть возможность организовать выбор значений из списка, то предпочтение следует отдать компонентам **ListBox, ComboBox** и их разновидностям**.**

К сожалению, полностью отказаться от набора данных на клавиатуре, как правило, не удаётся. Тогда следует рассмотреть возможность пользования компонент, задающих маску ввода (например, **MaskEdit**) или разрешающих ввод определённых значений (например, **SpinEdit**).

Иногда известны ограничения на вводимые значения. Например, для возраста и стажа работника, скорости движения поезда, грузоподъёмности пассажирского лифта и др. можно указать минимальное и максимальное значения, для шифра – количество символов. Часто в формулировке задачи допустимые значения указываются явно и, используя свойства MaxValue, MinValue, MaxLength, можно задать ограничения на значения. Хорошими возможностями для организации корректного ввода обладает компонент **ValueListEditor.**

При вводе данных с клавиатуры в обычные поля надо учесть возможность случайных ошибок: нажатие «неверной» клавиши (например, буквы вместо цифры) ни в коем случае не должно приводить к нарушению работы приложения.

Контролировать данные можно на разных уровнях. Прежде всего, выполняется контроль по формальным признакам (синтаксический).

Например, если вводится число, то во вводимой последовательности не должно быть букв, если число целое положительное, то должны вводиться только цифры и т.п.

Для анализа введённых символов целесообразно создавать обработчики события нажатия клавиш **OnKeyPress** и события **OnChange,** возникающего при любых изменениях в содержимом редактора. Иногда полезно контролировать свойство **Modified**, которое при изменении содержимого редактора принимает значение true. Это свойство можно использовать, например, для принятия решения о сохранении изменённых данных.

Иногда хорошие результаты даёт использование традиционного подхода к обработке ошибок. Так, для ввода целых и особенно вещественных чисел целесообразно использовать процедуру **Val**. Формируемый в этой процедуре код ошибки позволяет оперативно реагировать на некорректный ввод (не прерывая работы программы). При вводе вещественных чисел и особенно при использовании процедуры **Val** необходимо проверить, какой используется разделитель между целой и дробной частью.

Вывод результатов

Для вывода данных можно использовать:

- реализующие окна сообщений функцию MessageDlg и процедуру ShowMessage;
- поля вывода **Label, StaticText**;
- компоненты-таблицы **StringGrid**.

Форматирование результатов расчетов.

Функции преобразования. Функции преобразования наиболее часто используются в инструкциях, обеспечивающих ввод и вывод информации, так как свойства практически всех компонентов, обеспечивающие эти процессы являются строкового типа. Например, для того чтобы вывести в поле вывода (компонент Label) диалогового окна значение переменной типа real, необходимо преобразовать число в строку символов, изображающую данное число. Это можно сделать при помощи функции FloatToStr, которая возвращает строковое представление

значения выражения, указанного в качестве параметра функции. Например, инструкция Label1.caption:= FioatTostr(x) выводит значение переменной x в поле Label1.

Таблица. Функции преобразования

Функция	***Значение функции***
IntToStr (k)	Строка, являющаяся изображением целого k
FloatToStr (n)	Строка, являющаяся изображением вещественного n
FloatToStrF(n, f , k,m)	Строка, являющаяся изображением вещественного n. При вызове функции указывают: f — формат (способ изображения); k — точность (нужное общее количество цифр); m — количество цифр после десятичной точки
StrToInt (s)	Целое, изображением которого является строка s
StrToFloat (s)	Вещественное, изображением которого является строка s

Точность представления чисел вещественного типа очень высока. Результат вычислений может иметь более десятка значащих цифр. Предположим, что в результате обработки результатов эксперимента на лабораторной работе по физике, мы получили значение некоторой величины равное 9.85432132, хотя исходные данные для расчета имели точность значительно меньшую. Такой результат лучше отформатировать перед выводом, округлив до требуемого количества цифр после точки. Это можно сделать с помощью функции format(). Функция format() не только округляет результаты, но и предоставляет программисту много других возможностей по представлению результатов вычислений при выводе их на экран или принтер. В общем виде обращение к функции выглядит так:

Format (<строка форматов>, <список объектов форматирования>).

Работа с датой и временем.

Для работы с датой и временем в предусмотрен специальный тип данных TDateTime (тип дата-время). Он предназначен для одновременного хранения и даты, и времени. Во внутреннем представлении он занимает 8 байтов и, подобно Currency, представляет собой вещественное число с фиксированной дробной частью: в целой части числа хранится дата,

в дробной – время. Дата определяется как количество суток, прошедших с 30 декабря 1899 года, а время - как часть суток, прошедших с 0 часов, так что значение 36444,837 соответствует дате 11.10.1999 и времени 20:05. Количество суток может быть и отрицательным, однако значения, меньшие – 693594 (соответствует дате 00.00.0000 от рождества Христова), игнорируются функциями преобразования даты к строковому типу.

Над данными типа TdateTime определены те же операции, что и над вещественными числами, а в выражениях этого типа могут участвовать константы и переменные целого и вещественного типов.

Для работы с датой и временем предусмотрено большое количество стандартных процедур и функций.

Практическое задание1. Ознакомиться со следующими функциями форматирования: StrToInt, StrToFloat, IntToStr, FloatToStr, FloatToStrF, Format.

Практическое задание2. Ознакомиться с функциями, редназначенными для работы с датой и временем: Date, Time, Now, DateToStr, TimeToStr, DateTimeToStr, FormatDateTime и другими функциями, определенными в модуле DateUtils.

Практическое задание3. Создать windows-приложение для вычисления значения выражения $y = \frac{\sin ax}{n}$.

Решение.

Этап1. Спецификация.

Необходимо создать приложение, позволяющее пользователю вводить значения параметров a и n, а также значение x с указанием единиц измерения (градусы / радианы) и выводить для него результат, при этом предыдущие результаты расчетов должны быть также сохранены.

Этап2. Разработка алгоритма.

Для решения задачи введем следующие обозначения: a, n, x – вводимые пользователем данные, y – результат вычислений, который будет подлежать выводу.

При вычислении y воспользуемся стандартной функцией sin() и операцией вещественного деления. Однако, прежде чем вычислять значение y необходимо выяснить в каких единицах измерения введена величина x. Если величина x введена в радианах, то просто вычисляем величину y по даннойформуле. Для случая введения x в градусах необходимо предусмотреть их перевод в радианы по формуле: x*pi/180. После окончания расчетов величина y будет выводиться.

Этап3. Создание интерфейса.

Создадим новый проект. Сохраним в отдельной папке файл модуля под именем unRaschet, а файл проекта под именем prFormula. Установим значение свойства Name появившейся формы как frmFormula, а свойства Caption как «Расчет по формуле».

1) Для того чтобы пользователь имел представление о том, по какой формуле будет производиться расчет, расчетную формулу необходимо поместить на форму. Для этого воспользуемся специальным компонентом OLE-контейнером, который используется для того, чтобы из приложения можно было обращаться к стандартным приложениям Microsoft.

Компонент OLE-контейнер расположен в группе System, палитры компонентов. Расположим его в том месте формы, где должна быть формула и растянем до нужных размеров. Для того чтобы вызвать список объектов, которые можно вставить в контейнер, сделаем двойной щелчок в поле контейнера, или вызовем из контекстного меню функцию Insert Object. В появившемся списке выберем построитель формул (Microsoft Equation 3.0) и нажмем ОК. В результате, нужный объект помещается в контейнер, хотя вид контейнера при этом может не измениться.

Замечание. Если построитель формул не появился на экране, сделайте двойной щелчок в поле контейнера, или вызовите из контекстного меню

функцию «Открыть».

Далее, используя средства построителя, наберем расчетную формулу и закроем построитель. Формула должна появиться в поле контейнера.

2) Большинство компонентов, которые будут находиться на форме, можно разделить на две группы: компоненты для ввода данных и компоненты для отображения результатов. Поэтому для улучшения внешнего вида интерфейса осуществим группировку всех компонентов с помощью компонента GroupBox (хотя группировка компонентов не является обязательной). Этот компонент находится в группе Standard палитры компонентов. Расположим на форме два объекта GroupBox и запишем в свойстве Caption одного их них – «Исходные данные к расчету», а для другого – «Протокол выполненных расчетов».

3) Для ввода значений параметров a, n и x будем использовать компоненты LabeledEdit, расположенные в группе Additional (можно было бы использовать и обычный компонент Edit, однако в состав компонента LabeledEdit уже входит надпись, параметры которой можно настраивать). Расположим три таких компонента в группе «Исходные данные к расчету». Присвоим этим компонентам имена: lbEdtA, lbEdtX, lbEdtN. Свойству Caption, которое находится внутри свойства EditLabel, присвоим соответственно значения «a», «x», «n (целое)».

4) В связи с тем, что размерность величины x может принимать всего два значения (градусы / радианы), причем по умолчанию установленным считается значение радианы, для ввода размерности этой величины используем компонент CheckBox, расположенный в группе Standard (вместо CheckBox можно использовать RadioButton, можно было бы использовать два таких компонента: один для градусов, другой для радиан или использовать RadioGroup).

LabeledEdit уже входит надпись, параметры которой можно настраивать). Расположим этот компонент в группе «Исходные данные к расчету». Присвоим ему именя chGradus, а свойству Caption присвоим соответственно значения «Градусы».

5) Для отображения результата расчетов можно было бы, например, использовать компонент Edit или Label. Однако, в связи с тем, что нам необходимо не только выводить новый результат, но и сохранить предыдущие, для вывода будем использовать компонент Memo, в котором каждый результат будет выводиться с новой строки.

Расположим компонент Memo, находящийся в группе Standard внутри группы «Протокол выполненных расчетов», и присвоим ему имя memProtocol. Присвоим свойству ScrollBars значение ssVertical. В результате в поле должна появиться полоса вертикальной прокрутки. Свойству ReadOnly присвоим значение True. Уберем надпись из компонента с помощью его свойства Lines.

6) В качестве управляющих компонентов, которые будут отвечать за выполнение каких-либо действий, будем использовать кнопки (также в качестве такого компонента мог выступать MainMemu, что с точки создания интерфейса является даже более предпочтительным). Основным действием является выполнение расчета, такжедля удобства пользователя предусмотрим возможность очистки полей ввода и очистки протокола расчетов, а также возможность завершения приложения, поэтому расположим на форме четыре кнопки с надписями «Выполнить расчет», «Очистить поля ввода», «Очистить протокол», «Выход» (для кнопки завершения приложения можно было бы использовать не обычную кнопку, а BitBtn). Присвоим этим кнопкам имена btnCalculate, btnClearEdit, BtnClearProtocol, btnClose.

Этап4. Определение возможных событий и реакций на них.

В нашем приложении будет предусмотрена реакция на следующие события:

Событие	***Реакция***
Щелчок левой клавишей мыши по кнопке «Выполнить расчет»	Считывание введенных пользователем значений, выполнение расчета и вывод результат.
Щелчок левой клавишей мыши по кнопке «Очистить поля ввода»	Очищение трех полей ввода в группе «Исходные данные к расчету»

Щелчок левой клавишей мыши по кнопке «Очистить протокол»	Очищение компонента Memo, в группе «Протокол выполненных расчетов».
Щелчок левой клавишей мыши по кнопке «Выход»	Завершение приложения.

Этап5. Кодирование (создание процедур обработки событий).

1) Процедура выполнения расчета по формуле.

Создадим шаблон процедуры и приступим к ее написанию.

Прежде всего, нужно присвоить имена исходным данным, результату и определить их типы. В нашей формуле все переменные имеют имена, удовлетворяющие требованиям языка, поэтому имена оставляем без изменения.

По условию, переменная n должна быть целого типа, остальные переменные будут вещественного типа, поэтому тип результата будет тоже вещественным.

Помимо чисел, нам понадобится переменная, которая будет содержать строку с результатами расчета для протокола. Назовем ее res, а тип у нее должен быть String.

Написание операторной части процедуры можно начать с формирования строки исходных данных, которая понадобится для протокола и, возможно, для вывода сообщения об ошибке в исходных данных. Эту строку можно создать, объединяя строки с названиями переменных и тексты, написанные в полях ввода данных.

Следующим этапом будет считывание исходных данных и преобразования их в числа требуемого типа (для этого воспользуемся функциями форматирования StrToFloat и StrToInt, хотя для вещественных чисел также можно было использовать функцию val). Так как при вводе данных возможны ошибки, то следует добавить обработку исключительной ситуации, возникающей при вводе. Для этого воспользуемся конструкцией try … except.

В этом фрагменте должна производится попытка преобразования текстов в числа заданного типа путем выполнения инструкций, записанных после служебного слова try. Если такая попытка оказывается неудачной, то выполняются

инструкции, записанные после служебного слова except. В этом случае изменяется значение строки res, в которой записаны значения исходных данных. Эта строка объединяется со строкой 'Не могу прочесть исходные данные', а между ними вставляется символ перевода строки chr(13).

Инструкция showMessage(res) во время выполнения создаст на экране окно с сообщением, записанным в строке res. С целью привлечения внимания пользователя сообщение об ошибке можно сопровождать звуковым сигналом с помощью процедуры MessageBeep($FFFFFFFF).

Если же текст в полях ввода удалось преобразовать в числа, начинает выполняться вторая часть процедуры, обеспечивающая расчет по формуле. Прежде чем вычислять значение величины y необходимо узнать размерность величины x. Если пользователь хотел ввести ее в градусах, он должен был установить переключатель в компоненте CheckBox. Узнать сделал ли он это можно с помощью логического свойства этого компонента Checked. Если оно равно значению true, то необходимо перевести значение x из градусов в радианы.

Кроме того, в этой части также возможно возникновение исключительной ситуации, так как величина n, на которую производится деление, может оказаться равной нулю. В результате обработки этой исключительной ситуации также изменяется значение строки res, в которой записаны значения исходных данных. Эта строка объединяется со строкой 'Не могу произвести вычисление: деление на 0, а между ними вставляется символ перевода строки chr(13).

Инструкция showMessage(res) во время выполнения создаст на экране окно с сообщением, записанным в строке res.

После осуществления попытки ввода исходных данных и вычисления результата, необходимо проверить были ли эти попытки удачными. Если все прошло хорошо, то можно сформировать строку результата, путем объединения строки исходных данных с полученным результатом и добавить ее в протокол расчетов (в связи с тем, что компонент Memo может отображать только строковые данные полученный результат необходимо перевести из

числового типа в строковый. Предпочтительнее всего делать это с помощью функций Format или FloatToStrF, которые позволяют регулировать количество знаков после запятой). В случае неудачи на одном из этапов в переменной y не содержится результат и выводить ее бессмысленно, поэтому в протокол расчетов добавляется строка res, содержащая исходные данные и сообщение информацию о причине невозможности осуществления вычислений.

Для возможности проверки того была ли попытка удачной или нет введем логическую переменную flag_error. Установим в качестве ее начального значения true и будем изменять его на false в случае возникновения исключительной ситуации.

2) Процедура очистки протокола.

Для очистки поля Memo используем событие onClick для кнопки «Очистить протокол» и метод Clear компонента Memo.

3) Процедура очистки полей ввода.

Данная процедура будет вызываться при нажатии кнопки «Очистить поля ввода». Саму очистку полей ввода можно осуществить путем присваивания их свойству Text пустой строки.

4) Процедура выхода.

Данная процедура будет вызываться при нажатии кнопки «Выход». Завершение приложения будет осуществляться с помощью процедуры Close.

Этап 6. Отладка.

Откомпилируем приложение. Устраним все синтаксические ошибки, указанные компилятором.

Этап 7. Тестирование.

Для возможности обнаружения всех алгоритмических ошибок для данного приложения необходимо минимум четыре тестовых примера: вместо одного из параметров вводится символ, значение n равно 0 и корректные значения параметров, при этом x вводится сначала в градусах, а затем в радианах. В последнем случае необходимо сравнить результат, полученный программой с

результатом вычислений с помощью калькулятора.

Также необходимо проверить правильность реакции программы на нажатие клавиш «Очистить поля ввода», «Очистить протокол» и «Выход».

В случае «непонятного» или некорректного поведения программы следует воспользоваться средствами отладки (например, пошаговой трассировкой программы).

Практическое задание 4. Реализовать windows-приложение, которое с использованием введенных пользователем целого числа k, вещественного числа a (с 4 знаками после запятой) и даты d, выводит 2 строки, содержащих информацию в соответствии с вариантом.

Указание. Для ввода даты должен быть использован компонент DateTimePicker.

Должны быть реализованы проверка правильности ввода и обработка некорректного ввода.

Для вывода результата должен быть использован компонент Memo.

Для получения результата в требуемом виде должны быть использованы функции форматирования.

Варианты заданий:

1. Строка 1: «Введенное число ***a***, в научном формате представления с ***k*** цифрами в мантиссе имеет вид: … » (если символьное представление содержит больше k символов, то оно должно быть округлено по первой отбрасываемой цифре).

Строка 2: «Введенная дата ***d*** раньше/позже (в зависимости от результата сравнения) текущей. Сегодня: число.месяц.год ч:мин:сек».

2. Строка 1: «Результатом умножения введенного числа ***a*** на k^{10} будет: … » (результат выводится с разделением на тысячи).

Строка 2: «Дате ***d*** соответствует день недели: …. Завтра будет: число месяц (полностью) год, ч:мин».

3. Строка 1: «Количество йен, равное ***a***, с учетом курса 1 йена – ***k*** руб., соответствует … руб.» (с использованием денежного формата).

Строка 2: «Порядковый номер дня для даты ***d*** в году равен: … Через ***k*** секунд будет: мин:сек».

4. Строка 1: «Округленное число ***a***, можно записать с использованием ***k*** позиций: … » (если символьное представление содержит меньше k цифр, оно дополняется слева символами 0).

Строка 2: «Между датой ***d*** и сегодняшней ровно … дней. Сейчас: ч:мин. Это равно … миллисекунд».

5. Строка 1: «Результат перевода округленного числа ***a***, в 16-ричную систему счисления с использованием ***k*** позиций равен: … » (если символьное представление содержит меньше k цифр, оно дополняется слева символами 0).

Строка 2: «Введена дата ***d***. В месяце … (из d) … дней. Через ***k*** минут будет: … (вывод времени осуществляется в 12 часовом формате, am – до полудня, pm - после)».

6. Строка 1: «Введенное число ***a***, можно представить с ***k*** цифрами после запятой: …, с ***k-1*** цифрой после запятой: …, с ***k-2*** цифрами после запятой: …».

Строка 2: «Введена дата ***d***. В … году (из d) … дней. Сейчас ч: мин. Это равно …мин.».

7. Строка 1: «Delphi представляет собой систему программирования. Систему программирования представляет собой Delphi. (и т.д. все осмысленные перестановки, причем слова предложения должны быть переданы в функцию форматирования только один раз). Было введено целое число ***k*** и вещественное число ***a***» (число a вывести с k знаками после запятой).

Строка 2: «Введенной дате ***d*** соответствует год …, … неделя (ее номер), день недели … Вчера было: число месяц (сокращенно: янв) год, ч:мин:с».

8. Строка 1: «Вещественные числа могут быть записаны в обычной форме: ***a*** с ***k*** знаками после запятой или в научной: … с ***k*** знаками в мантиссе».

Строка 2: «Сначала месяца до введенной даты ***d*** прошло … часов. Через ***k*** секунд будет: … (вывод времени осуществляется в 12 часовом формате, a – до полудня, p - после)».

9. Строка 1: «Если вещественное число ***a***, умножить на k^{13}, то получится большое число …, которое удобнее читать с разбиением на тысячи: … ».

Строка 2: «Введенная дата ***d*** принадлежит / не принадлежит (в зависимости от результата проверки) високосному году. Сейчас: ч:мин (без ведущего нуля). От начала дня прошло … мин.»

10. Строка 1: «Использование компонентов уменьшает строки разработки программ. Строки разработки программ уменьшает использование компонентов. (и т.д. все осмысленные перестановки, причем слова предложения должны быть переданы в функцию форматирования только один раз). Было введено целое число ***k*** и вещественное число ***a***» (число a вывести с k знаками после запятой).

Строка 2: «Разница между текущей датой с текущим временем … и введенной датой ***d*** с текущим временем … составляет: … (число) Сейчас: … мин.».

11. Строка 1: «Один килограмм конфет стоит ***a*** фунтов, а с учетом того, что 1 фунт стоит ***k*** руб. он стоит …руб.» (с использованием денежного формата).

Строка 2: «Была введена дата ***d***. Сегодня … число месяца … … года. Сейчас: … часов … минут… секунд … миллисекунд».

12. Строка 1: «Число ***a*** можно преобразовать к виду, в котором оно будет занимать нужное число позиций. Например, с использованием ***k*** позиций оно будет выглядеть как: …» (если символьное представление содержит меньше k цифр, оно дополняется слева символами 0).

Строка 2: «Вернуться в прошлое можно, заменив в текущей дате число на число … (из d) и уменьшив часы на 1. Получим: … Сейчас: ч:мин:сек:млс.» (исходная дата и время должны храниться в одной переменной, итоговая дата должна быть получена из исходной путем непосредственной замены числа и часов и также должна храниться в одной переменной).

13. Строка 1: «Любое целое число легко может быть переведено в 16-ричную систему счисления. Если под запись 16-ричного числа отвести ***k*** позиций, то результатом перевода округленного числа ***a*** будет: … » (если символьное

представление содержит меньше k цифр, оно дополняется слева символами 0).

Строка 2: «Введенная дата ***d*** совпадает / не совпадает (в зависимости от результата сравнения) с текущей, так как <день даты d>=<день текущей даты>, <месяц даты d >=<месяц текущей даты >, <год даты d >=<год текущей даты >. Завтра в это время тоже будет: ч:мин, а через 1ч 30мин будет: …».

14. Строка 1: «Вещественные числа могут быть записаны с разной точностью. Например, число ***a*** может быть записано с ***k*** знаками после запятой: …, или с ***k+1*** знаком: …, или с ***k+2*** знаками: …».

Строка 2: «Сегодня: число.месяц.год, а не ***d***. Первым моментом этой недели был: число.месяц.год ч:мин:сек. Сейчас: ч:мин:сек (без ведущих нулей). Через 25мин 40сек будет: ч:мин».

15. Строка 1: «Delphi предназначена для разработки программ. Для разработки программ предназначена Delphi. (и т.д. все осмысленные перестановки, причем слова предложения должны быть переданы в функцию форматирования только один раз). Было введено целое число ***k*** и вещественное число ***a***» (число a вывести с k знаками после запятой).

Строка 2: «Текущая дата … и дата ***d*** отстоят / не отстоят (в зависимости от результата сравнения) не более чем на ***k*** дней. Сейчас: ч:мин:сек, а 1ч 30мин назад было: ч:мин:сек».

Контрольные вопросы.

1. Назовите способы ввода данных.

2. В каких случаях для ввода данных целесообразно использовать переключатели (функции, реализующие диалоги)?

3. Что такое фокус ввода? Какие компоненты им обладают? Как его установить?

4. Перечислите способы, используемые для контроля вводимых данных.

5. Почему при использовании нескольких полей ввода предпочтительнее не заключать их в один защищённый блок?

6. Как с помощью функции random можно задать вещественные числа в диапазоне от 1 до 100?

7. Запишите оператор для занесения данных в ячейку компонента

StringGrid, расположенную во второй строке и третьем столбце.

8. Приведите пример ситуации, в которой целесообразно использовать компонент **SpinEdit (UpDown).**

9. Запишите оператор, очищающий свойство Caption компонента **Label**.

10. Как узнать, не является ли строка, записанная в компоненте **Edit (Memo, Label)** пустой?

11. Назовите способы вывода данных.

12. В каких случаях для вывода данных целесообразно использовать диалоговые окна?

13. Зачем при вводе и выводе значений необходимо их преобразование из строкового или в строковый тип?

14. Для чего необходимо форматирование результатов при выводе?

15. Поясните использование функций IntToStr, StrToInt, FloatToStr, FloatToStrF, StrToFloat и Format.

16. Для чего предназначен тип данных TdateTime? Какие значения могут принимать переменные этого типа? Какое представление используется для их хранения в памяти компьютера?

17. Поясните использование функций Date, Time, Now, DateToStr, TimeToStr, DateTimeToStr, FormatDateTime.

Лабораторная работа 9. Создание приложений для работы с графикой

Теоретический материал.

- Для построения изображений определены следующие графические примитивы:
- линия и метод **LineTo** его отображения.
- кусочно-ломаная линия и метод **PolyLine** и его отображения.
- хорда (линия поперек эллипса) и метод **Chord** его отображения.
- прямоугольная рамка и метод **FrameRect** его отображения.
- заполненный прямоугольник и метод **Rectangle** его отображения.
- заполненный многоугольник и метод **Polygon** его отображения.
- заполненный прямоугольник со скругленными углами и метод **RoundRect** его отображения.
- дуга **и** метод **Arc** его отображения.
- заполненный эллипс и метод **Ellipse** его отображения.
- пирог (сектор эллипса) и метод **Pie** его отображения.

Определен метод заполнения замкнутых фигур **FloodFill** (заполнить область).

Имеется метод вывода текста: **TextOut** (вставить строку текста). Замкнутые фигуры с помощью инструмента **Brush** могут быть залиты узором.

Для рисования изображений могут быть использованы компоненты **Timage, TpaintBox и TForm**.

При создании анимации образ движущегося объекта рисуется на холсте со смещением во времени. Перед очередным рисованием область фона, на которую накладывается текущий образ движущегося объекта, запоминается в буфере для последующего восстановления.

Практическое задание 1. Разработать приложение, которое будет рисовать на экране движущийся объект (или объекты) в соответствии с вариантом.

Указания. Должна быть предусмотрена возможность изменения скорости движения (с помощью компонента **TrackBar),** остановки и запуска

объекта.

Для реализации случайного поведения следует использовать тандартную функцию Random.

Все изображения, показываемые в приложении, должны быть нарисованы средствами Delphi. Не допускается использование готовых изображений (нарисованных в графических редакторах), а также использование компонента TShape.

Варианты заданий.

1. Домик в окне которого периодически гаснет и загорается свет.
2. Движущийся автомобиль (после того, как автомобиль достигнет края формы, он должен будет вернуться назад).
3. Ночное небо на котором через некоторое время падают звезды.
4. Плывущий кораблик (после того, как кораблик достигнет края формы, он должен будет вернуться назад).
5. Линия горизонта и дорога. По дороге должен идти человек так, чтобы при постепенном его удалении уменьшались и его размеры. На уровне горизонта он должен превратиться в точку.
6. Телевизор, отображающий жестикулирующего человека
7. Сначала отдельные части пирамиды, а затем постепенное ее построение в соответствии с рисунком.
8. Движущийся паровоз (после того, как паровоз достигнет края формы, он должен будет вернуться назад)
9. Часы, из которых через определенные промежутки времени вылетает кукушка.
10. Крепость, перед воротами которой передвигается охранник.
11. Постепенное увядание цветка.
12. Неваляшка, которая раскачивается до максимально возможной амплитуды, затем постепенно амплитуда колебаний уменьшается и она останавливается.

13. “Светофор”. Нарисовать модель светофора (следует учесть мигание желтого цвета).

14. “Закат” и “восход” солнца. По мере исчезновения солнца за линией горизонта оно должно менять свой цвет с желтого на красный, а небо должно становиться из голубого темно-синим.

15. “Гроза”. Небо меняет цвет с голубого на черный, затем появляется молния (она должна менять цвет с желтого до оранжевого) и идет дождь.

Контрольные вопросы.

1. Назначение графических примитивов.
2. Графический примитив «Линия» и метод **LineTo** его отображения.
3. Графический примитив «Кусочно-ломаная линия» и метод **PolyLine** и его отображения.
4. Графический примитив «Хорда» (линия поперек эллипса) и метод **Chord** его отображения.
5. Графический примитив «Прямоугольная рамка» и метод **FrameRect** его отображения.
6. Графический примитив «Заполненный прямоугольник» и метод **Rectangle** его отображения.
7. Графический примитив «Заполненный многоугольник» и метод **Polygon** его отображения.
8. Графический примитив «Дуга» **и** метод **Arc** его отображения.
9. Графический примитив «Заполненный эллипс» и метод **Ellipse** его отображения.
10. Графический примитив «Пирог» (сектор эллипса) и метод **Pie** его отображения.
11. Метод заполнения замкнутых фигур **FloodFill** (заполнить область).
12. Методы вывода текста: **TextOut** (вставить строку текста).
13. Компонент **TShape**.
14. Компоненты **TImage и TPaintBox**.
15. Принцип создания анимации.

Лабораторная работа 10. Подпрограммы. Динамические массивы

Практическое задание 1. Изучить процедуру Randomize и функцию Random. Повторить теоретический материал по динамическим массивам.

Практическое задание 2. Разработать приложение, которое будет выполнять описанные ниже действия над числами и многочленами (в соответствии с вариантом), которые могут как задаваться пользователем, так и генерироваться случайным образом.

Указания. Числа, для представления которых в стандартных компьютерных типах данных не хватает количества двоичных разрядов, называются иногда «многоразрядными». Одним из возможных способов представления таких чисел является представление их в виде одномерного массива, в каждой ячейке которого хранится одна цифра числа.

Размерность массивов задается с клавиатуры.

Использовать динамические массивы.

Варианты заданий.

1. Задан ряд последовательных натуральных чисел от n до m (n<m), из которого удаляют сначала все числа, стоящие на нечетных местах. Затем из оставшегося ряда удаляют все числа, стоящие в нем на четных местах. Эти действия повторяют до тех пор, пока не останется одно число. Требуется написать программу, которая находит оставшееся число.

2. Пусть A массив, состоящий из N элементов $A_1,...,A_N$. Обозначим его максимальное и минимальное значение как max(A) и min(A) соответственно. Вычислим сумму элементов S, $S=A_1+A_2+...+A_N$. Заменим каждый элемент массива на разницу S и этого элемента: $A_i:=S-A_i$, 1<=i<=N. Такое преобразование массива A назовем операцией Confuse. Напишите программу, которая по массиву B, полученному в результате K кратного применения операции Confuse к некоторому массиву A, вычислит разность max(A)-min(A).

3. Уравнение x!a=b. Операция «!» действует с целыми положительными числами побитово, т.е. надо представить числа в двоичном виде и для каждой пары

бит с одинаковыми номерами выполнить действие по следующим правилам: 0!0=0, 1!0=1, 0!1=1, 1!1=0. Требуется написать программу, которая по заданным a и b решит уравнение x!a=b.

4. Задано натуральное число N (1<=N<=9999). Требуется написать программу, определяющую последнюю ненулевую цифру числа N!=1*2*3*...*N.

5. Дано число в K-ичной системе счисления, состоящее из N цифр. Для записи этого числа кроме цифр 0, 1, … 9, если требуется, используются заглавные латинские буквы A, B, ,Z. Также задано натуральное число M. Требуется написать программу, которая находит остаток от деления числа N на число M.

6. Определить НОД двух многоразрядных чисел.

7. Факториалом натурального числа N называется произведение всех натуральных чисел от 1 до N включительно - $N! = 1*X_2*X_3*X*\ldots*X_N$. Требуется написать программу, которая определит каким количеством нулей заканчивается запись числа N! в K-ичной системе счисления.

8. Определить НОК двух многоразрядных чисел.

9. Рассмотрим числовую последовательность, первоначально состоящую из двух единиц: 1, 1. Далее на каждом последующем шаге будем вставлять между соседними элементами их сумму. Требуется написать программу, которая подсчитает сумму членов последовательности, построенной за K шагов.

10. Даны действительные числа s и t, многочлен P(x) степени n. Найти значение $\int_s^t P(x)dx$

11. Даны действительное число a, многочлен P(x) степени N. Получить многочлен $(x-a)P(x)$.

12. Даны действительное число a, многочлен P(x) степени N. Получить многочлен $(x^2+a^2)P(x)$.

13. Последовательность многочленов $T_0(x)$, $T_1(x)$,… определяется следующим образом

$$T_0(x)=1,\quad T_1(x)=x,\quad T_k(x)=2xT_{k-1}(x)-T_{k-2}(x)\quad (k=2,3,\ldots).$$

Получить $T_N(x)$.

14. Последовательность многочленов $L_0(x)$, $L_1(x)$,… определяется

следующим образом

$$L_0(x) = 1,\ L_1(x) = x,\ L_k(x) = xL_{k-1}(x) - \frac{(k-1)^2}{(2k-3)(2k-1)}L_{k-2}(x)\quad (k = 2,3,\ldots).$$

Даны действительные числа d_0, d_1,… d_N. Получить многочлен $d_0 + d_1L_1(x) + \ldots + d_NL_N(x)$.

15. Написать программу извлечения квадратного корня из многоразрядного числа.

16. Рассмотрим последовательности длины N, состоящие из 0 и 1. Требуется написать программу, которая по заданному натуральному числу N, выводит все последовательности, в которых никакие две единицы не стоят рядом, а также определяет их количество.

17. Числа Фибоначчи F_1, F_2, ... определяются начальными значениями и соотношением: $F_1=1$; $F_2=2$; $F_n=F_{n-1}+F_{n-2}$. Рассмотрим систему счисления с двумя цифрами 0 и 1, в которой, в отличие от двоичной системы, весами являются не степени двойки 1, 2, 4, 8, 16, ..., а числа Фибоначчи 1, 2, 3, 5, 8, 13, ... В этой системе счисления каждое положительное целое число единственном способом представляется в виде строки из нулей и единиц, которая начинается с 1 и в которой нет двух единиц, стоящих рядом. Требуется написать программу, которая по двум заданным строкам, представляющим числа A и B в такой системе счисления, находила строку, представляющую число A+B также в этой системе счисления.

Например, исходные строки 10101 и 100 представляют числа 1*8+0*5+1*3+0*2+1*1=8+3+1=12 и 1*3+0*2+0*1=3. Ответом является строка 100010, представляющая число 1*13+0*8+0*5+0*3+1*2+0*1=13+2=15=12+3.

18. Написать программу вычисления N!, при N≥100.

19. Написать программу вычисления a^N, при N≥100, 0<a<9.

20. Вычислять двойной факториал числа, не превосходящего 150.

21. Определить, являются ли два многоразрядных числа взаимно простыми.

22. Вычислить значение производной многочлена P(x) степени N в заданной точке.

23. Дан многочлен P(x) степени N. Получить многочлен P(x+1)-P(x).

24. Даны действительное число a, многочлен P(x) степени N. Получить многочлен $(x^2 + abx + c)P(x)$.

25. Даны действительные числа s и t, многочлен P(x) степени n. Получить Многочлен

$$(sx^2 + t)P(x) + P'(x)$$

где P`(x) – производная многочлена P(x).

26. Последовательность многочленов $H_0(x)$, $H_1(x)$,… определяется следующим образом

$$H_0(x) = 1,\ H_1(x) = x,\ H_k(x) = xH_{k-1}(x) - (k-1)H_{k-2}(x) \quad (k = 2,3,\ldots)$$

Даны действительные числа d_0, d_1,… d_N. Получить многочлен $a_0H_0(x) + \ldots + a_NH_N(x)$.

27. Последовательность многочленов $G_0(x)$, $G_1(x)$,… определяется следующим образом

$$G_0(x) = 1,\ G_1(x) = x - 1,$$
$$G_k(x) = (x - 2k + 1)G_{k-1}(x) - (k-1)^2G_{k-2}(x) \qquad (k = 2,3,\ldots).$$

Дано действительное число a. Вычислить $G_0(a) + G_1(a) + \ldots + G_N(a)$.

28. Факториалом числа N называется произведением 1*2*3*…*N. Требуется написать программу, которая найдет разложение факториала заданного числа на простые множители. При выводе простые числа должны идти в порядке возрастания их значений.

29. Дан многочлен P(x) степени N≤100. Получить многочлен $P^a(x)$, где $0<a\leq 10$.

30. По заданным коэффициентам многочлена P(x) и многочлена Q(x), заданных степеней, определить коэффициенты многочлена P(Q(x)).

Контрольные вопросы.

1. Что такое подпрограмма? Какие виды подпрограмм существуют? Чем они отличаются?

2. Какие способы передачи параметров в подпрограмму существуют?

3. Принцип работы с динамическими массивами.

Лабораторная работа 11. Работа с файлами

Практическое задание 1. Разработать приложение, позволяющее создавать типизированные файлы, заполненные случайным образом, выполнять обработку хранящейся в них информации в соответствии с вариантом и просматривать их содержимое.

Указания. Количество информации, записываемой в файл должно задаваться пользователем.

При отсутствии специальных указаний для обработки информации, хранящейся в файлах, не допускается использование структурированных типов данных (массивов, строк и т.д.).

Отображаться должно содержимое всех используемых в приложении файлов.

Варианты заданий.

1. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Если его размер меньше m, то дополнить его нулями до m элементов; если его размер больше m, то урезать его до m элементов. Разрешается использовать в качестве вспомогательного файл second.txt.

2. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Расположить в файле first.txt элементы в обратном порядке. Разрешается использовать в качестве вспомогательного файл second.txt.

3. Создать файл data1.txt целых чисел из диапазона -100..100, содержащий k элементов, файл data2.txt целых чисел из диапазона -50..50, содержащий m элементов, файл data3.txt целых чисел из диапазона -10..10, содержащий n элементов. Создать файл result.txt, в котором чередовались бы элементы исходных файлов с одним и тем же номером. «Лишние» элементы более длинных файлов в результирующий файл не записывать.

4. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Удалить из файла first.txt нечетные числа. Разрешается использовать в качестве вспомогательного файл second.txt.

5. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Записать в файл second.txt количество участков возрастания элементов файла first.txt.

6. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Записать в файл second.txt длины всех возрастающих последовательностей элементов файла first.txt.

7. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов (A_1, A_2 … A_k). Записать в файл second.txt все числа из файла first.txt, поменяв их исходное расположение на следующее: A_1, A_k, A_2, A_{k-1}, A_3, ...

8. Создать файл first.txt целых чисел из диапазона -50..50, содержащий k элементов. Продублировать в файле first.txt все числа, принадлежащие диапазону 5..10. Разрешается использовать в качестве вспомогательного файл second.txt.

9. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов, упорядоченных по возрастанию и файл целых чисел second.txt из диапазона -100..100, содержащий q элементов, также упорядоченных по возрастанию. Объединить эти файлы в новый файл с именем result.txt, сохранив упорядоченность элементов.

10. Создать символьный файл first.txt, содержащий k элементов, среди которых, по крайней мере, один символ является пробелом. Удалить из файла first.txt те из них которые, расположены после первого символа пробела, включая и сам этот пробел. Разрешается использовать в качестве вспомогательного файл second.txt.

11. Создать символьный файл first.txt, содержащий k элементов, среди которых, по крайней мере, один символ является пробелом. Удалить из файла first.txt те из них которые, расположены перед первым символом пробела, включая и сам этот пробел. Разрешается использовать в качестве вспомогательного файл second.txt.

12. Создать символьный файл first.txt, содержащий k элементов. Записать в файл second.txt символы из файла first.txt, упорядочив их по возрастанию их кодов. (Допускается использование массива).

13. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Числа из файла first.txt будем считать элементами прямоугольной

матрицы A размерности n x m (записанными по строкам). Записать в файл second.txt элементы матрицы (по строкам), транспонированной к A. Если элементов файла first.txt не достаточно для формирования матрицы A, то оставить файл second.txt пустым. (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

14. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Числа из файла first.txt будем считать элементами симметричной относительно главной диагонали матрицы A размерности n (записанными по строкам), расположенными выше главной диагонали. Записать в файл second.txt все элементы матрицы A (по строкам). Если элементов файла first.txt не достаточно для формирования матрицы A, то оставить файл second.txt пустым. (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

15. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов и файл целых чисел second.txt, содержащий q элементов. Числа из файла first.txt будем считать элементами ненулевой части верхней треугольной матрицы A размерности n (записанные по строкам), а числа из файла second.txt будем считать элементами ненулевой части верхней треугольной матрицы B размерности m (записанные по строкам). Записать в файл result.txt ненулевую часть произведения A·B исходных матриц (по строкам). Если матрицы A и B нельзя перемножать, то оставить файл result.txt пустым. (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

16. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Заменить в файле first.txt каждый элемент на его квадрат. Разрешается использовать в качестве вспомогательного файл second.txt.

17. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Заменить в файле first.txt каждый элемент, кроме начального и последнего, на его среднее арифметическое с предыдущим и последующим элементом. Разрешается использовать в качестве вспомогательного файл

second.txt.

18. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Записать в файл result1.txt числа из файла first.txt с четными номерами, а в файл result2.txt числа из файла first.txt с нечетными номерами.

19. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Удалить из файла first.txt положительные числа. Разрешается использовать в качестве вспомогательного файл second.txt.

20. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Записать в файл second.txt количество участков убывания элементов файла first.txt.

21. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Записать в файл second.txt длины всех убывающих последовательностей элементов файла first.txt.

22. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Поменять в файле first.txt местами минимальный и максимальный элементы. Разрешается использовать в качестве вспомогательного файл second.txt.

23. Создать файл first.txt целых чисел из диапазона -50..50, содержащий k элементов. Заменить в файле first.txt все отрицательные числа нулем. Разрешается использовать в качестве вспомогательного файл second.txt.

24. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов, упорядоченных по убыванию и файл целых чисел second.txt из диапазона -100..100, содержащий q элементов, также упорядоченных по убыванию. Объединить эти файлы в новый файл с именем result.txt, сохранив упорядоченность элементов.

25. Создать символьный файл first.txt, содержащий k элементов, среди которых, по крайней мере, один символ является пробелом. Записать в файл second.txt символы из файла first.txt, удалив те из них которые, расположены после последнего символа пробела, включая и сам этот пробел.

26. Создать символьный файл first.txt, содержащий k элементов, среди которых,

по крайней мере, один символ является пробелом. Записать в файл second.txt символы из файла first.txt, удалив те из них которые, расположенные перед последним символом пробела, включая и сам этот пробел.

27. Создать символьный файл first.txt, содержащий k элементов. Записать в файл second.txt символы из файла first.txt, упорядочив их по убыванию их кодов. (Допускается использование массива).

28. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Числа из файла first.txt будем считать элементами ненулевой части верхней треугольной матрицы A (записанными по строкам). Записать в файл second.txt размерность матрицы A и ее элемент, расположенный в i-й строке и j-м столбце (i и j задаются пользователем). Если требуемый элемент находится в нулевой части матрицы, то вывести 0; если элемент отсутствует, то вывести –1. Если элементов файла first.txt не достаточно для формирования матрицы A, то оставить файл second.txt пустым. При выводе на экран отображать элементы файла second.txt в виде матрицы.

29. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов. Числа из файла first.txt будем считать элементами симметричной относительно побочной диагонали матрицы A размерности n (записанными по строкам), расположенными выше побочной диагонали. Записать в файл second.txt все элементы матрицы A (по строкам). Если элементов файла first.txt не достаточно для формирования матрицы A, то оставить файл

second.txt пустым. (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

30. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k элементов и файл целых чисел second.txt из диапазона -50..50, содержащий q элементов. Числа из файла first.txt будем считать элементами ненулевой части нижней треугольной матрицы A размерности n (записанные по строкам), а числа из файла second.txt будем считать элементами ненулевой части нижней треугольной матрицы B размерности m (записанные по строкам). Записать в файл

result.txt ненулевую часть произведения A·B исходных матриц (по строкам). Если матрицы A и B нельзя перемножать, то оставить файл result.txt пустым. (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

Практическое задание 2. Разработать приложение, позволяющее создавать, заполнять и просматривать типизированный файл, элементами которого будут записи, содержащие введенную пользователем информацию о студентах (номер группы, ФИО, дата рождения, домашний адрес, телефон)

Указания. Для хранения информации об одном студенте, также как и для ее записи в файл и считывания из файла допускается использование одной переменной комбинированного типа.

Практическое задание 3. Разработать приложение, позволяющее создавать текстовые файлы, заполненные случайным образом, выполнять обработку хранящейся в них информации в соответствии с вариантом и просматривать их содержимое, а также добавлять конец файла строку, введенную пользователем.

Указания. Количество информации, записываемой в файл должно задаваться пользователем.

Для всех вариантов, кроме 10-12 и 25-27 элементы строки должны отделяться друг от друга пробелом.

При отсутствии специальных указаний для обработки информации, хранящейся в файлах, не допускается использование структурированных типов данных (массивов, строк и т.д.). Однако, рекомендуется использовать строки при необходимости копирования содержимого одного файла в другой, а также при выводе содержимого файла.

Отображаться должно содержимое всех используемых в приложении файлов.

Варианты заданий.

1. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Если размер строки меньше m, то дополнить ее нулями до m

элементов; если ее размер больше m, то урезать ее до m элементов. Разрешается использовать в качестве вспомогательного файл second.txt.

2. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Расположить в каждой строке файла first.txt элементы в обратном порядке. (Допускается использование одного одномерного массива). Разрешается использовать в качестве вспомогательного файл second.txt.

3. Создать файл data1.txt целых чисел из диапазона -100..100, содержащий t строк по k элементов, файл data2.txt целых чисел из диапазона -50..50, содержащий t строк по m элементов, файл data3.txt целых чисел из диапазона -10..10, содержащий t строк по n элементов. Создать файл result.txt, содержащий t строк, в каждой из которых чередовались бы элементы соответствующих строк исходных файлов с одним и тем же номером. «Лишние» элементы более длинных строк в строки результирующего файла не записывать.

4. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Удалить из каждой строки файла first.txt нечетные числа. Разрешается использовать в качестве вспомогательного файл second.txt.

5. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Записать в строках файла second.txt количество участков возрастания элементов соответствующей строки файла first.txt.

6. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Записать в строках файла second.txt длины всех возрастающих последовательностей элементов соответствующей строки файла first.txt.

7. Создать файл first.txt целых чисел из диапазона -100..100, содержащий t строк по k элементов (A_1, A_2 … A_k). Записать в файл second.txt все числа из файла first.txt, сохранив его структуру и поменяв исходное расположение чисел в каждой строке на следующее: A_1, A_k, A_2, A_{k-1}, A_3, ...

8. Создать файл first.txt целых чисел из диапазона -50..50, содержащий k строк по t элементов. Продублировать в каждой строке файла first.txt все числа, принадлежащие диапазону 5..10. Разрешается использовать в качестве вспомогательного файл second.txt.

9. Создать файл first.txt целых чисел из диапазона -100..100, содержащий t строк по k элементов, упорядоченных по возрастанию и файл целых чисел second.txt из диапазона -100..100, содержащий t строк по q элементов, также упорядоченных по возрастанию. Объединить эти файлы в новый файл с именем result.txt, содержащий t строк, сохранив упорядоченность элементов в каждой строке.

10. Создать файл first.txt, содержащий k строк по t символов, среди которых, по крайней мере, один символ является пробелом. Удалить из каждой строки файла first.txt те из них, которые, расположены после первого символа пробела, включая и сам этот пробел. Разрешается использовать в качестве вспомогательного файл second.txt.

11. Создать файл first.txt, содержащий k строк по t символов, среди которых, по крайней мере, один символ является пробелом. Удалить из каждой строки файла first.txt те из них, которые, расположены перед первым символом пробела, включая и сам этот пробел. Разрешается использовать в качестве вспомогательного файл second.txt.

12. Создать файл first.txt, содержащий k строк по t символов. Записать в файл second.txt символы из файла first.txt, сохранив его структуру и упорядочив символы в каждой строке по возрастанию их кодов. (Допускается использование одного одномерного массива).

13. Создать файл first.txt целых чисел из диапазона -100..100, содержащий n строк по m элементов. Числа из файла first.txt будем считать элементами прямоугольной матрицы A размерности n x m (записанными по строкам). Записать в файл second.txt элементы матрицы (по строкам), транспонированной к A. (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

14. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк. Числа из файла first.txt будем считать элементами симметричной относительно главной диагонали матрицы A размерности k (записанными по строкам), расположенными выше главной диагонали. Записать в файл second.txt все элементы матрицы A (по строкам). (Допускается использование массива). При

выводе на экран отображать элементы файлов в виде матриц.

15. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк и файл целых чисел second.txt, также содержащий k строк. Числа из файла first.txt будем считать элементами ненулевой части верхней треугольной матрицы A размерности k (записанные по строкам), а числа из файла second.txt будем считать элементами ненулевой части верхней треугольной матрицы B размерности k (записанные по строкам). Записать в файл result.txt ненулевую часть произведения A·B исходных матриц (по строкам). (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

16. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Заменить в каждой строке файла first.txt каждый элемент на его квадрат. Разрешается использовать в качестве вспомогательного файл second.txt.

17. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Заменить в каждой строке файла first.txt каждый элемент, кроме начального и последнего, на его среднее арифметическое с предыдущим и последующим элементом. Разрешается использовать в качестве вспомогательного файл second.txt.

18. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Записать в строки файла result1.txt числа из соответствующих строк файла first.txt с четными номерами, а в строки файла result2.txt числа из соответствующих строк файла first.txt с нечетными номерами.

19. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Удалить из каждой строки файла first.txt положительные числа. Разрешается использовать в качестве вспомогательного файл second.txt.

20. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Записать в строках файла second.txt количество участков убывания элементов соответствующей строки файла first.txt.

21. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк

по t элементов. Записать в строках файла second.txt длины всех убывающих последовательностей элементов соответствующей строки файла first.txt.

22. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк по t элементов. Поменять в каждой строке файла first.txt местами минимальный и максимальный элементы. Разрешается использовать в качестве вспомогательного файл second.txt.

23. Создать файл first.txt целых чисел из диапазона -50..50, содержащий k строк по t элементов. Заменить в каждой строке файла first.txt все отрицательные числа нулем. Разрешается использовать в качестве вспомогательного файл second.txt.

24. Создать файл first.txt целых чисел из диапазона -100..100, содержащий t строк по k элементов, упорядоченных по убыванию и файл целых чисел second.txt из диапазона -100..100, содержащий t строк по q элементов, также упорядоченных по убыванию. Объединить эти файлы в новый файл с именем result.txt, содержащий t строк, сохранив упорядоченность элементов в каждой строке.

25. Создать файл first.txt, содержащий k строк по t символов, среди которых, по крайней мере, один символ является пробелом. Записать в файл second.txt символы из файла first.txt, сохранив его структуру и удалив из каждой строки те из них которые, расположены после последнего символа пробела, включая и сам этот пробел.

26. Создать файл first.txt, содержащий k строк по t символов, среди которых, по крайней мере, один символ является пробелом. Записать в файл second.txt символы из файла first.txt, сохранив его структуру и удалив из каждой строки те из них которые, расположенные перед последним символом пробела, включая и сам этот пробел.

27. Создать файл first.txt, содержащий k строк по t символов. Записать в файл second.txt символы из файла first.txt, сохранив его структуру и упорядочив символы в каждой строке по убыванию их кодов. (Допускается использование массива).

28. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк. Числа из файла first.txt будем считать элементами ненулевой части верхней

треугольной матрицы A размерности k (записанными по строкам). Записать в файл second.txt элемент матрицы A, расположенный в i-й строке и j-м столбце (i и j задаются пользователем). Если требуемый элемент находится в нулевой части матрицы, то вывести 0; если элемент отсутствует, то вывести –1. При выводе на экран отображать элементы файла second.txt в виде матрицы.

29. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк. Числа из файла first.txt будем считать элементами симметричной относительно побочной диагонали матрицы A размерности k (записанными по строкам), расположенными выше побочной диагонали. Записать в файл second.txt все элементы матрицы A (по строкам). (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

30. Создать файл first.txt целых чисел из диапазона -100..100, содержащий k строк и файл целых чисел second.txt из диапазона -50..50, содержащий k строк. Числа из файла first.txt будем считать элементами ненулевой части нижней треугольной матрицы A размерности k (записанные по строкам), а числа из файла second.txt будем считать элементами ненулевой части нижней треугольной матрицы B размерности k (записанные по строкам). Записать в файл result.txt ненулевую часть произведения A·B исходных матриц (по строкам). (Допускается использование массива). При выводе на экран отображать элементы файлов в виде матриц.

Выводы

1. Связь с файлами может быть осуществлена через файловые переменные.
2. При применении файловых переменных для связи с файлами на диске используется процедура **AssignFile**.
3. В текстовом файле произвольный доступ к символу невозможен. Можно обращаться к строкам текста.

Контрольные вопросы.

1. Какие существуют типы файлов?
2. Способы связи с файлами.
3. Файловая переменная.

4. Как связать файловую переменную с дисковым файлом?

5. Процедуры создания, открытия, закрытия типизированных файлов.

6. Процедуры создания, открытия, закрытия текстовых файлов.

7. Процедуры создания, открытия, закрытия нетипизированных файлов.

8. Контроль ошибок работы с файлами.

ТРЕБОВАНИЯ К ПРОГРАММАМ

1. Каждый проект должен содержаться в отдельной папке.

2. Папка, содержащая проект, должна иметь осмысленное название, по которому можно составить представление о цели проекта.

3. Название файла проекта должно отличаться от «Project*.dpr» и должно быть осмысленными.

4. Любая программа должна содержать информацию о ее назначении (название или краткое описание задачи, на решение которой она направлена), а также информацию о ее авторе.

5. На одной строке не должно быть больше одного оператора.

6. Тела составных, условных, циклических операторов должны быть выделены отступами.

7. Программа должна содержать комментарии, поясняющие основные моменты программы.

8. Названия переменных и подпрограмм должны быть осмысленными.

9. При объявлении переменной или подпрограммы в комментариях должно быть описано ее назначение.

10. Программы, предусматривающие ввод данных с клавиатуры или любое другое управление программой с помощью клавиатуры, должны выводить на экран сообщения, поясняющие ожидаемые от пользователя действия, т.е. обеспечивать «диалоговый режим» работы.

11. При решении задач практического содержания необходимо указывать в каких единицах должны быть введены значения тех или иных величин. Вывод значений величин на экран также должен сопровождаться указанием единиц измерения.

ТРЕБОВАНИЯ К WINDOWS-ПРИЛОЖЕНИЯМ.

1. Каждый проект должен содержаться в отдельной папке.

2. Папка, содержащая проект, должна иметь осмысленное название, по которому можно составить представление о цели проекта.

3. Название файла проекта и файла модуля должны отличаться от «Project*.dpr» и «Unit*.pas» должно быть осмысленными.

4. Любое приложение должно содержать информацию о его назначении (название или краткое описание задачи, на решение которой она направлена), а также информацию о ее авторе.

5. Программа должна содержать комментарии, поясняющие основные моменты программы.

6. Названия переменных и подпрограмм должны быть осмысленными.

7. При объявлении переменной или подпрограммы в комментариях должно быть описано ее назначение.

8. Имена всех компонент должны отличаться от стандартных (Button1, Label1 и т.д.). Они должны отражать назначение компонента.

9. Приложение должно иметь дружественный интерфейс. Пользователь должен четко представлять, что ему нужно сделать для получения результата, и каковы будут последствия каждого из осуществленного им действий (например, все поля ввода и вывода должны быть подписаны, все компоненты должны осмысленно располагаться на форме и т.п.). При этом приложение должно содержать минимум кнопок, а все основные действия (насколько это возможно) должны быть помещены в меню (компонент MainMenu).

10. Программа должна содержать обработку возможных исключительных ситуаций.

11. При решении задач практического содержания, необходимо предусмотреть возможность указания единиц измерения вводимых величин. Вывод значений величин также должен сопровождаться указанием единиц измерения.

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Архангельский, А.Я. Язык Pascal и основы программирования в Delphi / А.Я. Архангельский. - М.: Бином-Пресс, 2008. - 496 с.

2. Емельянов, В.И. Основы программирования на Delphi. / В.И. Емельянов. - М.: Высшая школа, 2005. - 231 с.

3. Желонкин, А. Основы программирования в интегрированной среде Delphi / А. Желонкин. - М.: Бином. Лаборатория знаний, 2004. - 236 с.

4. Зыков, С.В. Основы современного программирования: Учебное пособие для вузов / С.В. Зыков. - М.: ГЛТ , 2012. - 444 с.

5. Колдаев, В.Д. Основы алгоритмизации и программирования: Учебное пособие / В.Д. Колдаев; Под ред. Л.Г. Гагарина. - М.: ИД ФОРУМ, ИНФРА-М, 2012. - 416 с.

6. Культин, Н.Б. Самоучитель. Основы программирования в Delphi 8 для MS.NET Framework / Н.Б. Культин. - СПб.: BHV, 2004. - 400 с.

7. Культин, Н.Б. Основы программирования в Delphi XE / Н.Б. Культин. - СПб.: BHV, 2011. - 416 с.

8. Культин, Н.Б. Основы программирования в Delphi 2006 для Windows / Н.Б. Культин. - СПб.: BHV, 2006. - 384 с.

9. Культин, Н.Б. Основы программирования в Turbo Delphi / Н.Б. Культин. - СПб.: BHV, 2012. - 384 с.

10. Культин, Н.Б. Основы программирования в Delphi 8 для Microsoft. NET Framework / Н.Б. Культин. - СПб.: BHV, 2004. - 400 с.

11. Окулов, С.М. Основы программирования / С.М. Окулов. - М.: Бином. Лаборатория знаний, 2012. - 336 с.

12. Юдин, Д.Б. Задачи и методы линейного программирования: Математические основы и практические задачи / Д.Б. Юдин, Е.Г. Гольштейн. - М.: КД Либроком, 2010. - 320 с.

Учебное издание

Семенова Ирина Владимировна

ОСНОВЫ ПРОГРАММИРОВАНИЯ

Методические указания