**Основы программирования**

**программирование программа алгоритм**

Программа. Этапы разработки программы. Спецификация. Разработка алгоритма. Кодирование. Отладка. Тестирование. Создание справочной системы. Создание установочного диска. Алгоритм и программа. Компиляция. Язык программирования Delphi. Тип данных. Переменная. Константы. Инструкция присваивания. Выражение. Тип выражения. Выполнение инструкции присваивания. Стандартные функции. Математические функции. Функции преобразования. Ввод данных. Вывод результатов. Процедуры и функции. Структура процедуры. Структура функции. Запись инструкций программы. Стиль программирования

Программа

Программа, работающая на компьютере, нередко отождествляется с самим компьютером, т. к. человек, использующий программу, "вводит в компьютер" исходные данные, как правило, при помощи клавиатуры, а компьютер "выдает результат" на экран, на принтер или в файл. На самом деле, преобразование исходных данных в результат выполняет процессор компьютера. Процессор преобразует исходные данные в результат по определенному алгоритму, который, будучи записан на специальном языке, называется программой. Таким образом, чтобы компьютер выполнил некоторую работу, необходимо разработать последовательность команд, обеспечивающую выполнение этой работы, или, как говорят, написать программу.

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.

6. Создание справочной системы.

7. Создание установочного диска (CD-ROM).

Спецификация

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.

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 вы предполагаете, что вашей программой будут пользоваться другие. На этом этапе следует проверить, как ведет себя программа на как можно большем количестве входных наборов данных, в том числе и на заведомо неверных.

Создание справочной системы

Если разработчик предполагает, что программой будут пользоваться другие, то он обязательно должен создать справочную систему и обеспечить пользователю удобный доступ к справочной информации во время работы с программой. В современных программах справочная информация представляется в форме СНМ- или HLP-файлов. Помимо справочной информации, доступ к которой осуществляется из программы во время ее работы, в состав справочной системы включают инструкцию по установке (инсталляции) программы, которую оформляют в виде Readme-файла в одном из форматов: TXT, DOC или НТМ.

Создание установочного диска

Установочный диск или CD-ROM создаются для того, чтобы пользователь мог самостоятельно, без помощи разработчика, установить программу на свой компьютер. Обычно помимо самой программы на установочном диске находятся файлы справочной информации и инструкция по установке программы (Readme-файл). Следует понимать, что современные программы, в том числе разработанные в Delphi, в большинстве случаев (за исключением самых простых программ) не могут быть установлены на компьютер пользователя путем простого копирования, так как для своей работы требуют специальных библиотек и компонентов, которых может и не быть у конкретного пользователя. Поэтому установку программы на компьютер пользователя должна выполнять специальная программа, которая помещается на установочный диск. Как правило, установочная программа создает отдельную папку для устанавливаемой программы, копирует в нее необходимые файлы и, если надо, выполняет настройку операционной системы путем внесения дополнений и изменений в реестр.

Алгоритм и программа

На первом этапе создания программы программист должен определить последовательность действий, которые необходимо выполнить, чтобы решить поставленную задачу, т. е. разработать алгоритм. Алгоритм — это точное предписание, определяющее процесс перехода от исходных данных к результату.

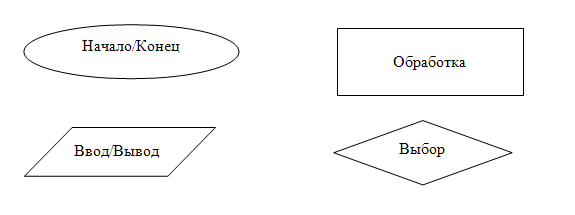


Рис. 1. Основные символы, используемые для представления алгоритма в виде блок-схемы

Алгоритм решения задачи может быть представлен в виде словесного описания или графически — в виде блок-схемы. При изображении алгоритма в виде блок-схемы используются специальные символы (рис. 1).

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

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

Компиляция

Программа, представленная в виде инструкций 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, схема работы которого приведена на рис. 2, выполняет последовательно две задачи:

1. Проверяет текст исходной программы на отсутствие синтаксических ошибок.

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

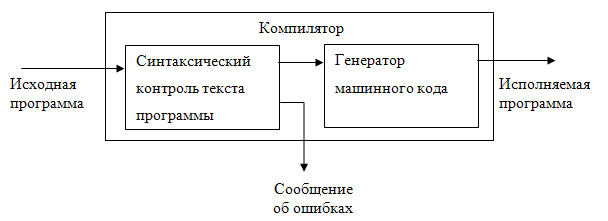


Рис. 2. Схема работы компилятора

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.

Язык программирования Delphi

В среде программирования Delphi для записи программ используется язык программирования Delphi. Программа на Delphi представляет собой последовательность инструкций, которые довольно часто называют операторами. Одна инструкция от другой отделяется точкой с запятой.

Каждая инструкция состоит из идентификаторов. Идентификатор может обозначать:

* Инструкцию языка **(:=,** if, while, for);
* переменную;
* константу (целое или дробное число);
* арифметическую (+, -,\*,/) или логическую (and, or, not) операцию;
* подпрограмму (процедуру или функцию);
* отмечать начало (procedure, function) или конец (end) подпрограммы ИЛИ блока (begin, end).

Тип данных

Программа может оперировать данными различных типов: целыми и дробными числами, символами, строками символов, логическими величинами.

Целый тип

Язык Delphi поддерживает семь целых типов данных: shortint, smailint, Longint, Int64, Byte, word и Longword, описание которых приведено в табл. 1.

Таблица 1. Целые типы

|  |  |  |
| --- | --- | --- |
| Тип | Диапазон | Формат |
| Shortint | -128-127 | 8 битов |
| Smallint | -32 768 - 32 767 | 16 битов |
| Longint | -2 147 483 648 - 2 147 483 647 | 32 бита |
| Int64 | -263- 263 - 1 | 64 бита |
| Byte | 0-255 | 8 битов, беззнаковый |
| Word | 0-65 535 | 16 битов, беззнаковый |
| Longword | 0 - 4 294 967 295 | 32 бита, беззнаковый |

Object Pascal поддерживает и наиболее универсальный целый тип - Integer, **который Эквивалентен** Longint.

Вещественный тип

Язык Delphi поддерживает шесть 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 (табл. 2).

Таблица 2. Вещественные (дробные) типы

|  |  |  |  |
| --- | --- | --- | --- |
| **Тип** | **Диапазон** | **Значащих цифр** | **Байтов** |
| Real48 | 2.9x 10-39-1.7x1038 | 11-12 | 06 |
| Single | 1.5 x 10-45-3.4х 1038 | 7-8 | 04 |
| Double | 5.0x10-324 -1.7x10308 | 15-16 | 08 |
| Extended | 3.6x10-4951 -1.1 х104932 | 19-20 | 10 |
| Comp | 263+1 - 263-1 | 19-20 | 08 |
| Currency | -922 337 203 685 477.5808 --922 337 203 685 477.5807 | 19-20 | 08 |

Язык Delphi поддерживает и наиболее универсальный вещественный тип - Real, который эквивалентен Double.

Символьный тип

Язык Delphi поддерживает два символьных типа: Ansichar и Widechar:

* тип Ansichar — это символы в кодировке ANSI, которым соответствуют числа в диапазоне от 0 до 255;
* тип widechar — это символы в кодировке Unicode, им соответствуют числа от 0 до 65 535.

Object Pascal поддерживает и наиболее универсальный символьный тип - Char, который эквивалентен Ansichar.

Строковый тип

Язык Delphi поддерживает три строковых типа: shortstring, Longstring

* WideString:
* тип shortstring представляет собой статически размещаемые в памяти компьютера строки длиной от 0 до 255 символов;
* тип Longstring представляет собой динамически размещаемые в памяти строки, длина которых ограничена только объемом свободной памяти;
* тип WideString представляет собой динамически размещаемые в памяти строки, длина которых ограничена только объемом свободной памяти. Каждый символ строки типа WideString является Unicode-символом.

В языке Delphi для обозначения строкового типа допускается использование идентификатора string. Тип string эквивалентен типу shortstring.

В языке Delphi для обозначения строкового типа допускается использование идентификатора string. Тип string эквивалентен типу shortstring.

Логический тип

Логическая величина может принимать одно из двух значений True (истина) или False (ложь). В языке Delphi логические величины относят к типу Boolean.

Переменная

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.

В качестве имени переменной можно использовать последовательность из букв латинского алфавита, цифр и некоторых специальных символов. Первым символом в имени переменной должна быть буква. Пробел в имени переменной использовать нельзя.

Следует обратить внимание на то, что компилятор языка Delphi не различает прописные и строчные буквы в именах переменных, поэтому имена SUMMA, Summa и summa обозначают одну и ту же переменную.

Желательно, чтобы имя переменной было логически связано с ее назначением. Например, переменным, предназначенным для хранения коэффициентов и корней квадратного уравнения, которое в общем виде традиционно записывают

ах2 + bх + с = 0

вполне логично присвоить имена а, b, с, x1 и х2. Другой пример. Если в программе есть переменные, предназначенные для хранения суммы покупки и величины скидки, то этим переменным можно присвоить имена TotalSumm и Discount или ObSumma и Skidka.

В языке Delphi каждая переменная перед использованием должна быть объявлена. С помощью объявления устанавливается не только факт существования переменной, но и задается ее тип, чем указывается и диапазон допустимых значений.

В общем виде инструкция объявления переменной выглядит так:

Имя : тип;

где:

* имя — имя переменной;
* тип — тип данных, для хранения которых предназначена переменная.

Пример:

а : Real; b : Real; i : Integer;

В приведенных примерах объявлены две переменные типа real и одна переменная типа integer.

В тексте программы объявление каждой переменной, как правило, помещают на отдельной строке.

Если в программе имеется несколько переменных, относящихся к одному типу, то имена этих переменных можно перечислить в одной строке через запятую, а тип переменных указать после имени последней переменной через двоеточие, например:

а,b,с : Real; x1,x2 : Real;

Константы

В языке Delphi существует два вида констант: обычные и именованные.

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

Числовые константы

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".

Ниже приведены примеры числовых констант:

123 0.0

-524.03 0

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.

В табл. 3 приведены примеры чисел, записанных в обычной форме, в алгебраической форме и форме с плавающей точкой.

Таблица 3. Примеры записи дробных чисел

|  |  |  |
| --- | --- | --- |
| Число | Алгебраическая форма | Форма с плавающей точкой |
| 1 000 000  -123.452  0,0056712 | 1х106  -1,23452x102  5,6712х10-3 | 1 .0000000000Е+06  -1 .2345200000Е+02  5,6712000000Е-03 |

Строковые и символьные константы

Строковые и символьные константы заключаются в кавычки. Ниже приведены примеры строковых констант:

'Язык программирования Delphi1 'Delphi 6'

'2.4'

'Д'

Здесь 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

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.

Инструкция присваивания

Инструкция присваивания является основной вычислительной инструкцией. Если в программе надо выполнить вычисление, то нужно использовать инструкцию присваивания.

В результате выполнения инструкции присваивания значение переменной меняется, ей присваивается значение.

В общем виде инструкция присваивания выглядит так: Имя : = Выражение;

где:

* Имя — переменная, значение которой изменяется в результате выполнения инструкции присваивания;
* : = — символ инструкции присваивания.
* Выражение — выражение, значение которого присваивается переменной, имя которой указано слева от символа инструкции присваивания.

Пример:

Surama := Сеnа \* Kol; Skidka := 10; Found := False;

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. Основные алгебраические операторы приведены в табл. 4.

Таблица 1.4. Алгебраические операторы

|  |  |
| --- | --- |
| Оператор | Действие |
| + | Сложение |
| - | Вычитание |
| \* | Умножение |
| / | Деление |
| DIV | Деление нацело |
| MOD | Вычисление остатка от деления |

При записи выражений между операндом и оператором, за исключением операторов DIV и MOD, пробел можно не ставить.

Результат применения операторов +, -, \* и / очевиден.

Оператор DIV позволяет получить целую часть результата деления одного числа на другое. Например, значение выражения is DIV i равно 2.

Оператор MOD, деление по модулю, позволяет получить остаток от деления одного числа на другое. Например, значение выражения 15 MOD 7 равно 1.

В простейшем случае выражение может представлять собой константу или переменную.Примеры выражений:

123 0.001 i+1

А + В/С Summa\*0.75 (В1+ВЗ+ВЗ)/3 Cena **MOD** 100

При вычислении значений выражений следует учитывать, что операторы имеют разный приоритет. Так у операторов \*, /, DIV, MOD более высокий приоритет, чем у операторов + и -.

Приоритет операторов влияет на порядок их выполнения. При вычислении значения выражения в первую очередь выполняются операторы с более высоким приоритетом. Если приоритет операторов в выражении одинаковый, то сначала выполняется тот оператор, который находится левее.

Для задания нужного порядка выполнения операций в выражении можно использовать скобки, например:

(r1+r2+r3)/(r1\*r2\*r3)

Выражение, заключенное в скобки, трактуется как один операнд. Это означает, что операции над операндами в скобках будут выполняться в обычном порядке, но раньше, чем операции над операндами, находящимися за скобками. При записи выражений, содержащих скобки, должна соблюдаться парность скобок, т. е. число открывающих скобок должно быть равно числу закрывающих скобок. Нарушение парности скобок — наиболее распространенная ошибка при записи выражений.

Тип выражения

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 о, 1 и -512 — целого типа (integer), а константы 1.0, 0.0 и 3.2Е-05 — вещественного типа (real).

В табл. 5 приведены правила определения типа выражения в зависимости от типа операндов и вида оператора.

Таблица 5. Правила определения типа выражения

|  |  |  |
| --- | --- | --- |
| Оператор | Тип операндов | Тип выражения |
| \*, +, - | Хотя бы один из операндов real | real |
| \*, +, - | Оба операнда integer | integer |
| / | real или integer | Всегда real |
| DIV, MOD | Всегда integer | Всегда integer |

Выполнение инструкции присваивания

Инструкция присваивания выполняется следующим образом:

1. Сначала вычисляется значение выражения, которое находится справа от символа инструкции присваивания.

2. Затем вычисленное значение записывается в переменную, имя которой стоит слева от символа инструкции присваивания.

Например, в результате выполнения инструкций:

* i:=0; — значение переменной i становится равным нулю;
* а:=b+с; — значением переменной а будет число, равное сумме значений переменных ь и с;
* j :=j+1; — значение переменной j увеличивается на единицу.

Инструкция присваивания считается верной, если тип выражения соответствует или может быть приведен к типу переменной, получающей значение. Например, переменной типа real можно присвоить значение выражения, тип которого real или integer, а переменной типа integer можно присвоить значение выражения только типа integer.

Так, например, если переменные i и n имеют тип integer, а переменная d — тип real, то инструкции

i:=n/10; i:=1.0;

неправильные, а инструкция

d:=i+1; правильная.

Во время компиляции выполняется проверка соответствия типа выражения типу переменной. Если тип выражения не соответствует типу переменной, то компилятор выводит сообщение об ошибке:

Incompatible types ... and ...

где вместо многоточий указывается тип выражения и переменной. Например, если переменная п целого типа, то инструкция n: = m/2 неверная, поэтому во время компиляции будет выведено сообщение :

Incompatible types 'Integer' and.'Extended'.

Стандартные функции

Для выполнения часто встречающихся вычислений и преобразований язык Delphi предоставляет программисту ряд стандартных функций.

Значение функции связано с ее именем. Поэтому функцию можно использовать в качестве операнда выражения, например в инструкции присваивания. Так, чтобы вычислить квадратный корень, достаточно записать k:=Sqrt(n), где Sqrt — функция вычисления квадратного корня, п — переменная, которая содержит число, квадратный корень которого надо вычислить.

Функция характеризуется типом значения и типом параметров. Тип переменной, которой присваивается значение функции, должен соответствовать типу функции. Точно так же тип фактического параметра функции, т. е. параметра, который указывается при обращении к функции, должен соответствовать типу формального параметра. Если это не так, компилятор выводит сообщение об ошибке.

Математические функции

Математические функции (табл. 6) позволяют выполнять различные вычисления.

Таблица 6. Математические функции

|  |  |
| --- | --- |
| Функция | Значение |
| Аbs (n) | Абсолютное значение n |
| Sqrt (n) | Квадратный корень из n |
| Sqr (n) | Квадрат n |
| Sin (n) | Синус n |
| Cos (n) | Косинус n |
| Arctan (n) | Арктангенс n |
| Ехр(n) | Экспонента n |
| Ln(n) | Натуральный логарифм n |
| Rardom(n) | Случайное целое число в диапазоне от 0 до n- 1 |

Величина угла тригонометрических функций должна быть выражена в радианах. Для преобразования величины угла из градусов в радианы используется формула (а\*з.141525б)/180, где: а— величина угла в градусах; 3.1415926 — число л. Вместо дробной константы 3.1415926 можно использовать стандартную именованную константу PI. В этом случае выражение пересчета угла из градусов в радианы будет выглядеть так: a\*Pi/180.

Функции преобразования

Функции преобразования (табл. 7) наиболее часто используются в инструкциях, обеспечивающих ввод и вывод информации. 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 х в поле Label1.

Использование функций

Обычно функции используют в качестве операндов выражений. Параметром функции может быть константа, переменная или выражение соответствующего типа. Ниже приведены примеры использования стандартных функций и функций преобразования.

n := Round((x2-x1)/dx);

x1:= (-b + Sqrt(d)) / (2\*а);

m := Random(10);

cena := StrToInt(Edit1.Text);

Edit2.Text := IntToStr(100);

mes := 'x1=' + FloatToStr(xl);

Ввод данных

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

Ввод из окна ввода

Окно ввода — это стандартное диалоговое окно, которое появляется на экране в результате вызова функции inputBox. Значение функции inputBox — строка, которую ввел пользователь.

В общем виде инструкция ввода данных с использованием функции inputBox выглядит так:

Переменная := InputBox(Заголовок, Подсказка, Значение);

где:

* Переменная — переменная строкового типа, значение которой должно быть получено от пользователя;
* Заголовок — текст заголовка окна ввода; Подсказка — текст поясняющего сообщения;
* Значение — текст, который будет находиться в поле ввода, когда окно ввода появится на экране.

Ниже в качестве примера приведена инструкция, используя которую можно получить исходные данные для программы пересчета веса из фунтов в килограммы. Окно ввода, соответствующее этой инструкции, приведено на рис. 3.

s:=InputBox('Пример', 'Введите что-нибудь', 'Привет');



Рис.3. Пример окна ввода

Если во время работы программы пользователь введет строку и щелкнет на кнопке **ОК,** то значением функции inputBox будет введенная строка. Если будет сделан щелчок на кнопке **Cancel,** то значением функции будет строка, переданная функции в качестве параметра значение.

Следует еще раз обратить внимание на то, что значение функции inputBox строкового (string) типа. Поэтому если программе надо получить число, то введенная строка должна быть преобразована в число при помощи соответствующей функции преобразования. Например, фрагмент программы пересчета веса из фунтов в килограммы, обеспечивающий ввод исходных данных из окна ввода, может выглядеть так:

s:=StrToInt(InputBox('Пример', 'Введите что-нибудь','10'));

Ввод из поля редактирования

Поле редактирования — это компонент Edit. Ввод данных из поля редактирования осуществляется обращением к свойству Text.

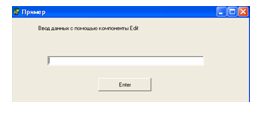


Рис. 4. Компонент Edit1 используется для ввода данных

На рис. 4 приведен вид диалогового окна программы в которой осуществляем ввод данных с помощью компонента Editl. Компонент Editl используется для ввода исходных данных. Инструкция ввода данных в этом случае будет иметь вид:

s := Edit1.Text;

Вывод результатов

Наиболее просто программа может вывести результат своей работы в окно сообщения или в поле вывода (компонент Label) диалогового окна.

Вывод в окно сообщения

Окна сообщений используются для привлечения внимания пользователя. При помощи окна сообщения программа может, к примеру, проинформировать об ошибке в исходных данных или запросить подтверждение выполнения необратимой операции, например, удаления файла.

Вывести на экран окно с сообщением можно при помощи процедуры ShowMessage или функции MessageDlg.

Процедура ShowMessage выводит на экран окно с текстом и командной кнопкой **ОК.**

В общем виде инструкция вызова процедуры ShowMessage выглядит так:

ShowMessage(Сообщение);

где сообщение — текст, который будет выведен в окне.

На рис. 5 приведен вид окна сообщения, полученного в результате выполнения инструкции:

ShowMessage('Пример процедуры.');

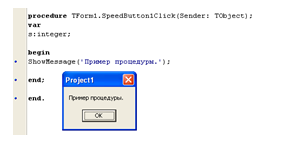


Рис. 5. Пример окна сообщения

Следует обратить внимание на то, что в заголовке окна сообщения, выводимого процедурой ShowMessage, указано название приложения, которое задается на вкладке **Application** окна **Project Options.** Если название приложения не задано, то в заголовке будет имя исполняемого файла.

Функция MessageDig более универсальная. Она позволяет поместить в окно с сообщением один из стандартных значков, например "Внимание", задать количество и тип командных кнопок и определить, какую из кнопок нажал пользователь. На рис. 6 приведено окно, выведенное в результате выполнения инструкции

r:=MessageDlg('Файл будет удален.', mtWarning, [mbOk,mbCancel] , 0) ;

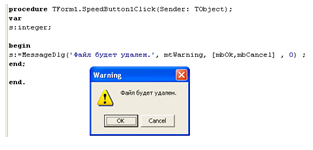


Рис. 6. Пример окна сообщения

Значение функции MessageDlg — число, проверив значение которого, можно определить, выбором какой командной кнопки был завершен диалог.

В общем виде обращение к функции MessageDig выглядит так:

Выбор: = MessageDlg( Сообщение, Тип, Кнопки, КонтекстСправки)

где:

* Сообщение — текст сообщения;
* Тип — тип сообщения. Сообщение может быть информационным, предупреждающим или сообщением о критической ошибке. Каждому типу сообщения соответствует определенный значок. Тип сообщения задается именованной константой (табл. 8);

Кнопки — список кнопок, отображаемых в окне сообщения. Список может состоять из нескольких разделенных запятыми именованных констант (табл.9). Весь список заключается в квадратные скобки.

Таблица 8. Константы функции MessageDlg

|  |  |
| --- | --- |
| Константа | Тип сообщения |
| mtWarning | Внимание |
| mtError | Ошибка |
| mtInformation | Информация |
| mtConfirmation | Подтверждение |
| mtCustom | Обычное |

Таблица 9. Константы функции MessageDlg

|  |  |  |  |
| --- | --- | --- | --- |
| Константа | Кнопка | Константа | Кнопка |
| mbYes | Yes | mb Abort | Abort |
| mbNo | No | mbRetry | Retry |
| mbOK | OK | mblgnore | Ignore |
| mbCancel | Cancel | mbAll | All |
| mbHelp | Help |  |  |

Например, для того чтобы в окне сообщения появились кнопки **ОК** и **Cancel,** список Кнопки должен быть таким:

[mbOK,mbCancel]

Кроме приведенных констант можно использовать константы: mbokcancel, mbYesNoCancel и mbAbortRetryIgnore. Эти константы определяют наиболее часто используемые в диалоговых окнах комбинации командных кнопок.

Контекст справки — параметр, определяющий раздел справочной системы, который появится на экране, если пользователь нажмет клавишу <F1>. Если вывод справки не предусмотрен, то значение параметра КонтекстСправки должно быть равно нулю.

Значение, возвращаемое функцией MessageDig (табл. 10), позволяет определить, какая из командных кнопок была нажата пользователем.

Таблица 10. Значения функции MessageDlg

|  |  |
| --- | --- |
| Значение функции MessageDig | Диалог завершен нажатием кнопки |
| mrAbort | Abort |
| mrYes | Yes |
| mrOk | Ok |
| mrRetry | Retry |
| mrNo | No |
| mrCancel | Cancel |
| mrIgnore | Ignore |
| mrAll | All |

Вывод в поле диалогового окна

Часть диалогового окна, предназначенная для вывода информации, называется полем вывода, или полем метки. Поле вывода — это компонент Label.

Содержимое поля вывода определяется значением свойства Caption. Изменить значение свойства Caption, как и большинства свойств других компонентов, можно как во время разработки формы приложения, так и во время работы программы.

Для того чтобы во время работы программы изменить содержимое поля вывода, например, вывести в поле результат работы программы, нужно присвоить свойству новое значение.

Свойство Caption символьного типа. Поэтому для того, чтобы во время работы программы вывести в поле метки числовое значение, нужно преобразовать число в строку, например, при помощи функции FloatToStr или IntToStr.

Процедуры и функции

При программировании в Delphi работа программиста заключается в основном в разработке процедур (подпрограмм) обработки событий.

При возникновении события автоматически запускается процедура обработки события, которую и должен написать программист. Задачу вызова процедуры обработки при возникновении соответствующего события берет на себя Delphi.

В языке Object Pascal основной программной единицей является подпрограмма. Различают два вида подпрограмм: процедуры и функции. Как процедура, так и функция, представляют собой последовательность инструкций, предназначенных для выполнения некоторой работы. Чтобы выполнить инструкции подпрограммы, надо вызвать эту подпрограмму. Отличие функции от процедуры заключается в том, что с именем функции связано значение, поэтому имя функции можно использовать в выражениях.

Структура процедуры

Процедура начинается с заголовка, за которым следуют: П раздел объявления констант;

* раздел объявления типов;
* раздел объявления переменных;
* раздел инструкций.

В общем виде процедура выглядит так:

**procedure** Имя (СписокПараметров);

**const**

// здесь объявления констант

**type**

// здесь объявления типов var

// здесь объявления переменных

**begin**

// здесь инструкции программы

**end;**

Заголовок процедуры состоит из слова procedure, за которым следует имя процедуры, которое используется для вызова процедуры, активизации ее выполнения. Если у процедуры есть параметры, то они указываются после имени процедуры, в скобках. Завершается заголовок процедуры символом "точка с запятой".

Если в процедуре используются именованные константы, то они объявляются в разделе объявления констант, который начинается словом const.

За разделом констант следует раздел объявления типов, начинающийся словом type.

После раздела объявления типов идет раздел объявления переменных, в котором объявляются (перечисляются) все переменные, используемые в программе. Раздел объявления переменных начинается словом var.

За разделом объявления переменных расположен раздел инструкций. Раздел инструкций начинается словом begin и заканчивается словом end, за которым следует символ "точка с запятой". В разделе инструкций находятся исполняемые инструкции процедуры.

Ниже в качестве примера приведен фрагмент программы вычисления стоимости покупки — процедура Summa.

**procedure** Summa;

**var**

cena: real; // цена

kol: integer; // количество

s: real; // сумма

mes: string[255]; // сообщение

**begin**

cena := StrToFloat(Form1.Edit1.Text);

kol := StrToInt(Form1.Edit2.Text);

s := cena \* kol; **if s >** 500 **then**

**begin**

s := s \* 0.9;

mes := 'Предоставляется скидка 10%'

+ #13; end; mes := mes+ 'Стоимость покупки: '

+ FloatToStrF(s,ffFixed,4,2) +' руб.';

Forml.Label3.Caption := mes; end;

Структура функции

Функция начинается с заголовка, за которым следуют разделы объявления констант, типов и переменных, а также раздел инструкций.

Объявление функции в общем виде выглядит следующим образом:

**function** Имя (Список Параметров) : Тип;

**const** // начало раздела объявления констант

**type** // начало раздела объявления типов

var // начало раздела объявления переменных

**begin** // начало раздела инструкций

result := Значение; // связать с именем функции значение

**end;**

Заголовок функции начинается словом function, за которым следует имя функции. После имени функции в скобках приводится список параметров, за которым через двоеточие указывается тип значения, возвращаемого функцией (тип функции). Завершается заголовок функции символом "точка с запятой".

За заголовком функции следуют разделы объявления констант, типов и переменных.

В разделе инструкций, помимо переменных, перечисленных в разделе описания переменных, можно использовать переменную result. По завершении выполнения инструкций функции значение этой переменной становится значением функции. Поэтому среди инструкций функции обязательно должна быть инструкция, присваивающая переменной result значение. Как правило, эта инструкция является последней исполняемой инструкцией функции.

Ниже в качестве примера приведена функция FuntToKg, которая пересчитывает вес из фунтов в килограммы:

// Пересчет веса из фунтов в килограммы

**function** FuntToKg(f:real):real;

**const**

// в России 1 фунт равен 409,5 гр.

К=0.4095; // коэф. Пересчета

**begin**

result:=f\*K;

**end;**

Запись инструкций программы

Одну инструкцию от другой отделяют точкой с запятой или, другими словами, в конце каждой инструкции ставят точку с запятой.

Хотя в одной строке программы можно записать несколько инструкций, как правило, каждую инструкцию программы записывают в отдельной строке.

Некоторые инструкции (if, case, repeat, while и др.) принято записывать в несколько строк, используя для выделения структуры инструкции отступы. Ниже приведен пример инструкции, которая записана в несколько строк и с использованием отступов:

**if** d >= 0 **then begin**

x1:=(-b+Sqrt(d))/(2\*a);

x2:=(-b-Sqrt(d))/(2\*a);

ShowMessage('x1='+FloatToStr(xl) +

'x2='+FloatToStr(x2)) ;

**end**

**else**

ShowMessage('Уравнение не имеет корней.');

Следует обратить внимание на то, что слова then и else записаны одно под другим (с одинаковым отступом) и с отступом относительно слова if. Слово end располагается под словом begin, а инструкции между begin и end размещаются одна под другой, но с отступом относительно begin.

Приведенную выше инструкцию можно записать и так:

**if d >= 0 then begin**

x1:=(-b+Sqrt(d))/(2\*a);

x2:=(-b-Sqrt(d))/(2\*a);

ShowMessage('x1='+FloatToStr(x1)+'x2='+FloatToStr(x2));

**end**

**else** ShowMessage('Уравнение не имеет корней.');

Однако первый вариант лучше, т. к. он отражает структуру алгоритма, реализуемого инструкцией. С первого взгляда видна группа инструкций, которая будет выполнена, если условие d >= о выполняется (в этом случае будут вычислены значения переменных xl и х2), и инструкция, которая будет выполнена, если условие d >=o не выполняется.

Длинные выражения тоже могут быть записаны в несколько строк. Разорвать выражение и перенести оставшуюся часть на следующую строку можно практически в любом месте. Нельзя разрывать имена переменных, числовые и строковые константы, а также составные операторы, например, оператор присваивания.

Ниже приведен пример записи выражения в несколько строк:

st:= 'Корни уравнения'+ #13

+'x1=' + FloatToStr(x1)+ #13 +'х2=' + FloatToStr(x2);

Еще один момент, на который следует обратить внимание. Компилятор игнорирует "лишние" пробелы и пустые строки. Так, он игнорирует все пробелы в начале строки. Кстати, это и позволяет записывать инструкции с отступами. Не требуются пробелы при записи арифметических и логических выражений (условий), списков параметров. Однако при их использовании программа легче воспринимается. Сравните два варианта записи инструкции присваивания:

x1:=(-b+Sqrt(d))/(2\*a);

x1 := (-b + Sqrt(d)) / (2 \* а);

**программирование программа алгоритм**

Очевидно, что второй вариант воспринимается лучше.

Для облегчения понимания логики работы программы в текст программы нужно включать поясняющий текст — комментарии. В общем случае комментарии заключают в фигурные скобки. Открывающая скобка помечает начало комментария, закрывающая — конец. Если комментарий однострочный или находится после инструкции, то перед комментарием ставят две наклонные черты.

Ниже приведен пример раздела объявления переменных, в котором использованы оба способа записи комментариев:

**var**

{ коэффициенты уравнения }

a:real; // при второй степени неизвестного

b:real; // при первой степени неизвестного

с:real; // при нулевой степени неизвестного

{ корни уравнения } x1,x2:real;

Стиль программирования

Работая над программой, программист, особенно начинающий, должен хорошо представлять, что программа, которую он разрабатывает, предназначена, с одной стороны, для пользователя, с другой — для самого программиста. Текст программы нужен прежде всего самому программисту, а также другим людям, с которыми он совместно работает над проектом. Поэтому для того, чтобы работа была эффективной, программа должна быть легко читаемой, ее структура должна соответствовать структуре и алгоритму решаемой задачи. Как этого добиться? Надо следовать правилам хорошего стиля программирования. Стиль программирования — это набор правил, которым следует программист (осознано или потому, что "так делают другие") в процессе своей работы. Очевидно, что хороший программист должен следовать правилам хорошего стиля.

Хороший стиль программирования предполагает:

* использование комментариев;
* использование несущих смысловую нагрузку имен переменных, процедур и функций;
* использование отступов;
* использование пустых строк.

Следование правилам хорошего стиля программирования значительно уменьшает вероятность появления ошибок на этапе набора текста, делает программу легко читаемой, что, в свою очередь, облегчает процессы отладки и внесения изменений.

Четкого критерия оценки степени соответствия программы хорошему стилю программирования не существует. Вместе с тем достаточно одного взгляда, чтобы понять, соответствует программа хорошему стилю или нет.

Сводить понятие стиля программирования только к правилам записи текста программы было бы неверно. Стиль, которого придерживается программист, проявляется во время работы программы. Хорошая программа должна быть прежде всего надежной и дружественной по отношению к пользователю.Надежность подразумевает, что программа, не полагаясь на "разумное" поведение пользователя, контролирует исходные данные, проверяет результат выполнения операций, которые по какой-либо причине могут быть не выполнены, например, операций с файлами. Дружественность предполагает хорошо спроектированные диалоговые окна, наличие справочной системы, разумное и предсказуемое, с точки зрения пользователя, поведение программы.