**Основы программирования в Visual FoxPro на 4 часа**

**Выражения**

*Выражением*называют какой-либо из элементов языка или их совокупность, которая в результате обработки (трансляции, выполнения, вычисления) приводит к одному оп­ределенному значению.

Примеры простейших выражений:

|  |  |
| --- | --- |
| **Выражение** | **Результат** |
| *1* | число *1* |
| 2+2 | число 4 |
| CTOD('28.12.02") | дата 28 декабря 2002 г. |
| "Сегодня 8 апреля" | символьное выражение |

Выражение может состоять из различных элементов Visual FoxPro: констант, пере­менных, массивов, функций и т.д. Элементы, входящие в выражение, связываются меж­ду собой арифметическими, логическими операторами и/или операторами сравнения.

В Visual FoxPro выражения бывают 4-х типов:

* символьные;
* числовые;
* типа даты и времени;
* логические.

Поля таблиц баз данных, функций, переменные памяти и элементы массива, комби­нируемые для создания выражения, обязательно должны иметь один тип данных. Если они имеют различные типы, то Visual FoxPro выдаст сообщение об ошибке.

**Символьные выражения**состоят из:

* полей таблиц баз данных символьного типа;
* функций, возвращающих символьное значение;
* переменных и элементов массивов, которые содержат символьные данные;
* констант символьного типа, называемых *строковыми литералами.*

Операторы, используемые при работе с символьными выражениями:

|  |  |
| --- | --- |
| **Оператор** | **Операция** |
| **+** | Слияние строковых выражений (две строки соединяются в одну) |
| — | Слияние строковых выражений (хвостовые пробелы первой строки переносятся в конец второй строки) |
| **$** | Сравнение символьных выражений |

**Числовые выражения**состоят из:

* полей таблиц базы данных числового типа;
* функций, возвращающих числовое значение;
* переменных и элементов массивов, содержащих числовые данные;
* числовых констант.

Операторы, работающие с числовыми выражениями:

|  |  |
| --- | --- |
| **Оператор** | **Операция** |
| ( ) | Скобки для группировки выражения |
| \*\*, ^ | Возведение в степень |
| \*, / | Умножение и деление |
| % | Остаток от деления двух чисел |
| +, - | Сложение и вычитание |

**Выражения типа даты и времени**состоят из:

* полей таблиц баз данных типа даты;
* функций, возвращающих значения типа даты;
* переменных и элементов массивов, содержащих данные типа даты;
* констант типа дата.

В выражениях типа даты и времени могут использоваться операторы сложения и вы­читания.

**Логические выражения**имеют одно из двух значений — истина и ложь.

В Visual FoxPro их обозначают соответственно (.Т.) и (.F.). Логические выражения состоят из:

* полей таблиц баз данных логического типа;
* функций, возвращающих логическое значение;
* переменных и элементов массивов, содержащих логические данные;
* выражений других типов (символьных, числовых или даты), разделенных специ­альными операторами отношения.

Операторы, работающие с логическими выражениями:

|  |  |
| --- | --- |
| **Оператор** | **Операция** |
| AND | Логическое **И** |
| OR | Логическое ИЛИ |
| !, NOT | Логическое отрицание |
| ( ) | Круглые скобки для группировки выражений |

Операции отношения, используемые в логических выражениях:

|  |  |
| --- | --- |
| **Оператор** | **Операция** |
| < | меньше, чем |
| > | больше, чем |
| = | равно |
| <>, !=, # | не равно |
| <= | меньше или равно |
| >= | больше или равно |
| $ | сравнение подстроки |
| == | сравнение точного совпадения строк символов |

**Переменные**

*Переменная*представляет собой ячейку временного хранения данных. Каждая пере­менная имеет собственное имя. После того как переменной присвоено значение, в программе вместо самого значения можно использовать эту переменную. В Visual FoxPro переменной может быть присвоено значение любого допустимого типа данных. Однако при дальнейшем использовании переменной необходимо учитывать тип сохраняемого в ней значения.

Имя переменной задается непосредственно при присвоении ей значения. Существует несколько правил задания имен переменных:

* имя переменной может содержать не более 254 символов;
* имя переменной может содержать любые буквы и цифры;
* первый символ в имени переменной должен быть буквой;
* в имени переменной должны отсутствовать пробелы.

Например, допустимы следующие имена переменных:

CurrentNum, Total, Date\_of\_birth

Следующие имена недопустимы:

ITime, $total, Date of birth, append, delete

Прежде чем использовать переменную в программе, ей нужно присвоить значение. Самый простой способ присвоения состоит в использовании оператора присвоения «=», который имеет следующий синтаксис:

**переменная = выражение**

Аргумент **переменная**задает имя переменной, которой будет присвоено значение вы­ражения, стоящего справа от знака равенства.

Например:

nFirst = 10

cName = "Иван"

Справа от знака равенства может стоять не только константа, но и более сложное вы­ражение.

Например:

nResult = 255 + nFirst

cName = "Иван" + alltrim(LastName)

В этих примерах для вычисления значений переменных используются другие пере­менные и функции, значения которых в свою очередь также могут изменяться.

Например, если переменным nFiist и/или nSecond не были присвоены значения, то выполнение конструкции

nResult = nFirst + nSecond вызовет ошибку.

Кроме оператора присваивания, можно использовать команду STORE, которая используется для сохранения значения в переменной или элементе массива.

STORE **выражение** ТО **списокПеременных / списокЭлементовМассива**или

STORE **имяПеременной** / **имяМассива = выражение**

Например:

STORE «Иванов» ТО cName

**Область действия переменных**

При выполнении программы принципиальное значение имеет область действия ис­пользуемых переменных. Попытка использования переменных, которые не действуют в данном месте программы, приводит к ошибке программирования или же к неодно­значности результатов. В Visual FoxPro могут использоваться:

* глобальные (PUBLIC);
* локальные (PRIVATE или LOCAL) переменные.

*Глобальными переменными*называются переменные или массивы переменных, кото­рые могут быть доступны для чтения или изменения из любой программы, которая выполняется в текущем сеансе работы с Visual FoxPro.

Любые переменные или массивы, которые необходимо определить в качестве глобаль­ных, должны быть объявлены с помощью команды:

PUBLIC **имяПеременной**

Переменные или массивы, создаваемые с помощью команды PUBLIC, при инициа­лизации получают значение «ложно» (.F.), кроме глобальных переменных Visual FoxPro, которые автоматически принимают значение «истинно» (.Т.).

Команда PRIVATE позволяет создавать локальные переменные или массивы, дос­тупные в пределах программы, в которой объявлена переменная, а также во всех процедурах и функциях, вызываемых из нее. При завершении выполнении программы все пе­ременные и массивы, которые имели те же имена, что и переменные и массивы, пере­численные в команде PRIVATE, будут восстановлены.

Синтаксис команды:

* PRIVATE **списокПеременных**
* PRIVATE ALL [ LIKE | EXCEPT **шаблон** ]

Синтаксис команды PRIVATE ALL LIKE **шаблон**позволяет выбрать те имена пе­ременных и массивов, которые соответствуют указанному шаблону. Шаблон может со­стоять из операторов шаблона ?, \* и строки текста. Использование синтаксиса PRIVATE ALL EXCEPT **шаблон**приведет к тому, что будут выбраны все переменные, кроме соответствующих шаблону.

Переменные, объявленные командой LOCAL, доступны только в пределах програм­мы, в которой объявлена переменная. Невозможно использовать значения этих пере­менных в процедурах и функциях, которые вызываются из нее.

Синтаксис команды:

LOCAL **списокПеременных**

Пример объявления локальных переменных.

PRIVATE cName, nSum LOCAL nCount, yUnitPrice

**Определение типа данных**

В Visual FoxPro тип данных не объявляется явно, а определяется его значением. Тип данных переменной может изменяться в процессе выполнения программы. Поэтому при выполнении операций с данными может возникнуть необходимость определения типа данных переменной или выражения. Для этого используется функцию TYPE, кото­рая имеет следующий синтаксис:

TYPE ( выражение )

Функция TYPE возвращает символ, описывающий тип данных выражения. Список возвращаемых типов:

|  |  |
| --- | --- |
| **Значение** | **Тип данных** |
| С | Символьный |
| N | Числовой (Numeric, Float, Double или Integer) |
| Y | Денежный |
| D | Типа дата |
| Т | Типа дата/время |
| L | Логический |
| М | Memo |
| О | Объект |
| G | Типа General |
| U | Неопределенный тип |

Например:

eType=TYPE ("Иванов" ) && Возвращает "С"

сТуре=ТYРЕ ("(12\*3)+4") && Возвращает "N"

сТуре=ТYРЕ ("DATE ()") && Возвращает "D"

сТуре=ТYРЕ ("DATETIME()") && Возвращает "Т"

сТуре=ТYРЕ ("Customer.mNotes") && Возвращает "М"

сТуре=ТYРЕ ("Москва=42") && Возвращает "U"

В Visual Foxpro 7.0 для определения типа данных переменной или выражения можно также использовать функцию VARTYPE, которая имеет следующий синтаксис:

VARTYPE ( выражение [, флагАнализаЗначенияNull ])

Функция VARTYPE возвращает символ, описывающий тип данных выражения.

Аргумент флагАнализаЗначенняNull используется для анализа выражений, содержа­щих пустое значение. Если данный аргумент установлен равным (.Т.), функция возвра­щает тип данных заданного выражения. Если же установлено значение (.F.) или данный аргумент не указан при вызове функции, возвращается символ «X», указывающий, что заданное выражение содержит пустое значение.

**Преобразование типов данных**

Visual FoxPro позволяет преобразование значений одних типов данных в другие, если это преобразование возможно. Преобразование чисел, дат в символьный вид возможно всегда. Преобразование же символьных выражений в число или дату возможно только в том случае, если значение выражения может интерпретироваться в качестве преобразуемого типа данных.

**Преобразование значений в строку**

*Функция CHR* возвращает символ, чей численный код такой же, как и значение вы­ражения.

Синтаксис функции: CHR( выражение )

Например:

? СНR(100) - d

*Функция STR* преобразует численное значение в символьное представление. Синтаксис функции: STR (выражение [, длинаСтроки [, дробнаяЧасть] ] )

Функция преобразует выражение в строку длиной «длинаСтроки». Количество цифр, которые должны появиться справа от десятичной точки, указывается аргументом «дробнаяЧастъ».

Например:

? STR(123.456,10,4) - 123.4560

*Функция DTOC* преобразует значение типа дата и дата/время в символьное пред­ставление.

Синтаксис функции: DTOC( выражение [, 1 ] )

Формат представления даты определяется текущими установками команд SET CEN­TURY и SET DATE. Необязательный аргумент 1 вызывает возвращение даты в формате YYYYMMDD, пригодной для непосредственного использования при индексировании таблиц.

Например:

? DTOC (DATE()) - 26.09.2005

*Функция DTOS* возвращает восьмиразрядную символьную строку в формате YYYYMMDD для заданного выражения даты или даты/времени.

Синтаксис функции: DTOS ( выражение )

Например:

? DTOS (DATE () ) - 20050926

*Функция SYS(10)*- преобразует числовое выражение номера дня по юлианскому календа­рю в символьную строку.

Для определения номера текущего дня используется *функция SYS(l)*

Синтаксис функции: SYS ( 10, номерДня )

Например:

? SYS ( 10, VAL (SYS(1)) - 26.09.2005

*Функция VAL* преобразует символьную строку в численное значение.

Синтаксис функции: VAL ( символьноеВыражение )

При преобразовании в число учитываются все цифровые символы, расположенные в строке слева направо до первого нецифрового символа. Пробелы игнорируются. Если первый символ выражения не является цифрой, функция VAL возвратит значение ноль.

Например:

сА="12"

сВ="15"

? VAL (сА) + VAL (сВ) - 25.00

**Преобразование в дату**

Для преобразования значения даты, выраженного в символьной форме, в выражение типа даты используется функция CTOD.

Синтаксис функции: СТОD( символьноеВыражение )

Форматом выражения по умолчанию является «mm/dd/yy». Для изменения этого фор­мата по умолчанию могут быть использованы команды SET DATE и SET CENTURY. Если при вводе даты век не указывается (например, 12.12.98), то подразумевается XX век. Выражение должно содержать разрешенную дату, лежащую в пределах между 1.1.100 и 31.12.9999.

Например:

cDatePay="26.09.05"

? CTOD(cDatePay) - 26.09.1905

**Преобразование в дату/время**

Для преобразования значения даты/времени, выраженного в символьной форме, в выражение типа даты/времени используется функция СТОТ. Формат символьного вы­ражения должен соответствовать текущим установкам команд SET DATE, SET HOURS и SET MARK.

Синтаксис функции: СТОТ( символьноеВыражение )

Например:

cDatePay=”31.03.99 10:10:11”

? CTOT(cDatePay) - 31.03.99 10:10:11 AM

**Математические функции и команды**

Перечень математических функций и команд приведен в таблице. Если значения числа не являются допустимыми для функции, возникает ошибка.

|  |  |  |
| --- | --- | --- |
| **Функция** | **Допустимые значения** | **Результат** |
| ABS( ) | Любое число | Абсолютное значение числа |
| ACOS ( ) | от - 1 до 1 | Арккосинус числа |
| ASIN ( ) | от - 1 до 1 | Арксинус числа |
| ATAN ( ) | Любое число | Арктангенс числа |
| ATN2 ( ) | Любое число | 4-х квадрантный арктангенс числа |
| BETWEEN ( ) | Любое число | Определяет, лежит ли числовое выражение между двумя другими числовыми выражениями |
| CALCULATE ( ) |  | Производит финансовые и статистические операции |
| CEILING ( ) | Любое число | Целое число, округленное до ближайшего сверху целого числа |
| COS ( ) | Любое число | Косинус угла в радианах |
| DTOR ( ) |  | Преобразует градусы в радианы |
| EMPTY ( ) |  | Определяет, равно ли числовое выражение нулю |
| EXP ( ) | Любое число | Экспонента числа |
| FLOOR ( ) | Любое число | Целое число, округленное до ближайшего снизу целого числа |
| INT ( ) | Любое число | Преобразует число в его целое значение, отбрасывая его десятичную часть |
| LOG ( ) | Положительное число | Натуральный логарифм числа |
| LOG 10 ( ) | Положительное число | Десятичный логарифм числа |
| MOD ( ) | Любое число | Остаток от деления двух чисел |
| PI ( ) |  | Возвращает значение Пи |
| RAND ( ) |  | Случайное число в диапазоне от 0 до 1 |
| ROUND ( ) | Любое число | Округленное до указанного количества десятичных разрядов значение числа |
| RTOD ( ) |  | Преобразует радианы в градусы |
| SET DECIMALS |  | Задает число десятичных знаков, выводимых для числовых результатов |
| SIGN ( ) | Любое число | Возвращает числовое значение, зависящее от знака заданного числа |
| SIN ( ) | Любое число | Синус угла в радианах |
| SQRT ( ) | Любое число | Квадратный корень числа |
| TAN ( ) | Любое число | Тангенс угла в радианах |
| VAL ( ) | Любое число | Преобразует строку числовых символов в числовое выражение |

**Символьные функции**

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| $ | Определяет, содержится ли символьная строка внутри другой символь­ной строки или Memo-поля |
| ALLTRIM ( ) | Убирает начальные и конечные пробелы из символьной строки |
| ASC ( ) | Определяет код ASCII для левого старшего символа в символьной строке |
| АТ ( ) | Определяет положение (в виде целого числа) первого вхождения сим­вольной строки внутри другого символьного выражения или Memo-по­ля |
| АТС ( ) | Определяет положение (в виде целого числа) первого вхождения сим­вольной строки внутри другого символьного выражения или Memo-по­ля. В отличие от функции AT, АТС выполняется без учета регистра |
| ATCLINE () | Определяет номер строки (в виде целого числа) первого вхождения символьной строки внутри символьной строки или Memо- поля. Воз­вращаемое значение зависит от значения, определенного командой SET MEMOWIDTH. Аналогична функции ATLINE, но выполняется без учета регистра |
| ATLINE ( ) | Определяет номер строки (в виде целого числа) первого вхождения символьного выражения внутрь символьного выражения или Memo-поля. Выдаваемое значение зависит от значения, определенного командой SET MEMOWIDTH |
| BETWEEN () | Определяет, находится ли символьное выражение между двумя другими символьными выражениями |
| CHR ( ) | Определяет символ для определенного числового кода ASCII |
| CHRTRAN () | Переводит символьную строку, используя две другие строки в качестве таблиц перевода |
| CTOD ( ) | Определяет дату в обычном виде для данного символьного типа даты |
| DTOC ( ) | Преобразует дату в строку символов |
| EMPTY ( ) | Определяет, является ли выражение пустым |
| ISALPHA ( ) | Возвращает (.Т.), если определенное символьное выражение начинается с буквы |
| ISDIGIT ( ) | Определяет, является ли самый левый символ цифрой |
| ISLOWER ( ) | Возвращает (.Т.), если первый символ в символьном выражении — строчный |
| ISUPPER ( ) | Возвращает (.Т.), если первый символ в символьном выражении — прописной |
| LEFTO | Возвращает заданное количество символов, начиная с самого левого символа в символьной строке |
| LEN( ) | Определяет длину символьной строки |
| LIKE ( ) | Позволяет сравнить символьную строку, содержащую универсальные символы, с другой символьной строкой |
| LOWER ( ) | Преобразует все символы в строке в их строчные эквиваленты |
| LTRIM ( ) | Удаляет начальные пробелы из заданной символьной строки |
| OCCURS ( ) | Определяет число вхождений символьной строки внутри другой символьной строки |
| PADC ( ) | Дополняет символьную строку с двух сторон заданным символом |
| PADL ( ) | Дополняет символьную строку слева заданным символом |
| PADR ( ) | Дополняет символьную строку справа заданным символом |
| PROPER () | Определяет выражение, написанное с заглавной буквы, как соответствующее имени собственному |
| RIGHT () | Возвращает заданное количество символов строки, начиная с самого правого |
| RTRIM ( ) | Убирает все конечные пробелы из определенной символьной строки |
| SPACE ( ) | Создает символьную строку, состоящую из определенного числа про­белов |
| STR ( ) | Преобразует числовое выражение в строку |
| SUBSTR 0 | Определяет заданное число символов данного выражения или Memo-поля |
| SYS (10) | Преобразует число дня в числовом виде в символьную строку |
| TRIM 0 | Удаляет все конечные пробелы |
| TYPE ( ) | Определяет тип данных выражения |
| VARTYPE () | Определяет тип данных выражения |
| UPPER ( ) | Преобразует все строчные буквы в заглавные |

**Обработка данных типа дата и дата/время**

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| BETWEEN ( ) | Определяет, лежит ли выражение между двумя другими выражениями |
| CDOW ( ) | Определяет наименование дня для данной даты |
| CMONTH ( ) | Определяет наименование месяца для данной даты |
| CTOD ( ) | Определяет дату в обычном виде для данного символьного типа даты |
| CTOT ( ) | Определяет дату/время для данного символьного типа даты/времени |
| DATE () | Определяет текущую системную дату |
| DATETIME ( ) | Определяет текущую дату и время из переменной даты/времени |
| DAY( ) | Определяет день месяца (числовой) для данной даты |
| DMY( ) | Преобразует выражение даты в символьную строку европейского синтаксиса — День/Месяц/Год |
| DOW ( ) | Определяет день недели (числовой) для данной даты |
| DTOC ( ) | Определяет дату символьного типа для данной даты обычного типа |
| DTOS ( ) | Преобразует выражение даты в символьную строку |
| DTOT ( ) | Преобразует выражение даты/времени в символьную строку |
| EMPTY ( ) | Определяет, является ли выражение даты «пустым» |
| FDATE ( ) | Возвращает дату последней модификации файла |
| FTIME ( ) | Возвращает время последней модификации файла |
| GOMONTH ( ) | Определяет дату, которая на определенное число месяцев предшествует или следует после заданной даты |
| HOUR ( ) | Возвращает время из выражения типа дата/время |
| MDY( ) | Преобразует выражение даты в символьную строку синтаксиса Месяц/День/Год |
| MINUTE ( ) | Определяет значение минут из выражения даты/времени |
| MONTH ( ) | Определяет месяц (числовой) для заданной даты |
| SEC ( ) | Определяет секунды для выражения типа дата/время |
| SECONDS ( ) | Определяет системное время в формате секунды, тысячные с разрешением 1 миллисекунда |
| SET HOURS | Устанавливает системные часы на 12-часовой или 24-часовой синтаксис |
| SET MARK TO | Задает разделитель, используемый при отображении дат |
| SET SECONDS | Устанавливает режим отображения секунд для значений типа дата/время |
| SYS (1) | Определяет системную дату в виде юлианского числа |
| SYS (2) | Определяет число секунд, прошедших с полуночи |
| SYS (10) | Преобразует числовое выражение номера дня в символьную строку |
| SYS (11) | Определяет по номеру дня юлианский эквивалент для определенной даты |
| TIME ( ) | Определяет текущее системное время |
| TTOD ( ) | Преобразует выражение типа дата/время в выражение типа дата |
| WEEK ( ) | Определяет порядковый номер неделя с начала года |
| YEAR ( ) | Определяет год, соответствующий дате |

**Установка формата даты**

Для установки формата отображения даты в программе или из командного окна используется команду SET DATE, которая имеет следующий синтаксис:

SET DATE ТО форматДаты

Аргумент форматДаты может принимать одно из следующих значений:

|  |  |
| --- | --- |
| **Значение** | **Формат** |
| AMERICAN | mm/dd/yy |
| ANSI | yy.mm.dd |
| BRITISH | dd/mm/yy |
| ITALIAN | dd-mm-yy |
| FRENCH | dd/mm/yy |
| GERMAN | dd.mm.yy |
| JAPAN | yy/mm/dd |
| TAIWAN | yy/mm/dd |
| USA | mm-dd-yy |
| MDY | mm/dd/yy |
| DMY | dd/mm/yy |
| YMD | yy/mm/dd |

**Логические функции**

|  |  |
| --- | --- |
| **Функция** | **Описание** |
| IIF ( ) | Альтернативная форма IF-структуры. Определяет одно из двух значений, основанных на логическом выражении |
| BOF() | Возвращает (.Т.), когда указатель записи устанавливается в начале файла базы данных |
| DELETED ( ) | Возвращает (Т.), если текущая запись помечена для удаления |
| EOF() | Возвращает (.Т.), когда указатель записи устанавливается за последней записью файла базы данных |
| FILE ( ) | Возвращает (.Т.), если определенный файл присутствует в текущем каталоге |
| FOUND ( ) | Возвращает (.Т.), если размещенная запись была найдена |
| ISALPHA ( ) | Возвращает (.Т.), если первый символ — буква |
| ISLOWER ( ) | Возвращает (.Т.), если первая буква — строчная |
| ISUPPER ( ) | Возвращает (Т.), если первая буква — прописная |
| NETWORK ( ) | Возвращает (.Т.), если Visual FoxPro установлена в сети |
| SEEK ( ) | Ищет индекс для значения и возвращает (.Т.), если значение найдено |

**Массивы**

Для хранения величин кроме простых переменных могут быть использованы масси­вы. *Массив*представляет собой набор переменных с одним именем и разными индекса­ми. Каждая такая переменная называется *элементом массива.*Количество хранящихся в массиве элементов называется *размером массива.*Размер массива ограничен объемом системной памяти и типом данных элементов массива.

Индекс элемента указывается в квадратных скобках после имени массива. Напри­мер, aNames[l], aNames[2], aNames[l0] являются элементами массива с именем aNames. Можно использовать каждый из элементов массива точно также, как и простую пе­ременную.

Объявление массива

Команда DECLARE позволяет создавать одно- или двумерные массивы переменных. Синтаксис команды:

**DECLARE массив1 (количествоЭлементов1[,количествоЭлементов2]) [,массив2(количествоЭлементовЗ[,количествоЭлементов4])...]**

Элементы массива переменных могут содержать любой тип данных. Элементы мас­сива, создаваемые с помощью команды DECLARE, при объявлении получают значение «ложно» (.F.).

Нумерация индексов массива начинается с 1. Если массив объявлен DECLARE с двумя индексами (двумерный массив), доступ к нему возможен по одному из индек­сов. В этом случае при определении индекса следует учесть, что элементы в массиве хра­нятся построчно.

При объявлении массива могут быть использованы как круглые, так и квадратные скобки.

Массивы, создаваемые в командном окне, автоматически получают атрибут PUB­LIC, а создаваемые программой — PRIVATE. Для объявления глобального массива сле­дует использовать команду PUBLIC.

Команды APPEND FROM ARRAY, COPY TO ARRAY, SCATTER и GATHER позво­ляют осуществлять обмен данных между записями базы данных и массивами.

Пример использования команды DECLARE.

DECLARE uCustomer(4,3)

uCustomer(1,1)="Петров"

uCustomer(1,2)=(21.03.02)

uCustomer(1,3)=20000

*Команда DIMENSION*

Команда DIMENSION предназначена для создания одно- или двумерных массивов переменных. По действию и синтаксису команда DIMENSION аналогична команде DECLARE.

Синтаксис команды:

**DIMENSION массив1 (количествоЭлементов1(,количествоЭлементов2) [, массив2(количествоЭлементов3[,количествоЭлементов4) ...]**

*Команда PUBLIC*

Команда **PUBLIC,**используемая для объявления глобальных переменных, может создавать глобальные одно- или двумерные массивы. Синтаксис команды:

**PUBLIC массив1 (количествоСтрок1 [,количествоСтолбцов1])**

**[,массив2(количествоСтрок2 [,количествоСтолбцов2])...]**

VFP позволяет изменить размер ранее объявленного массива и даже преобразовать его из двумерного в одномерный и наоборот. Для этого используется команда DIMENSION или DECLARE для существующего мас­сива.

**Обмен данными между массивом и таблицей**

Команде *APPEND FROM ARRAY*

Эта команда добавляет к выбранной на текущий момент базе данных записи из пере­менной памяти типа массив. Каждая строка массива становится при этом записью базы данных.

Синтаксис команды:

APPEND FROM ARRAY массив [FOR условие]

В случае одномерного массива, содержимое его первого элемента станет первым по­лем новой добавляемой записи, содержимое второго элемента — вторым полем и т.д. В том случае, если одномерный массив имеет больше элементов, чем полей в базе дан­ных, то все лишние элементы массива игнорируются. Если же база данных имеет боль­ше полей, чем массив элементов, то все лишние поля заполняются значениями по умол­чанию. Оставшиеся символьные поля заполняются пробелами, числовые поля — нуля­ми, поля типа даты — пустой датой, а логические поля — значением (.F.).

При добавлении записей из двумерного массива, содержимое его первого столбца становится первым полем вновь добавляемой записи, содержимое второго столбца — вторым полем и т.д. Если двумерный массив имеет больше элементов, чем полей в базе данных, оставшиеся столбцы массива игнорируются. Если же база данных имеет больше полей, чем массив столбцов, оставшиеся поля заполняются аналогично одномерному массиву.

При включении в команду конструкции **FOR условие**возможно выполнение добавления записей из массива с ограничением. Условие должно содержать имена полей таблицы, в которую добавляется запись. Прежде чем произойдет добавление строки массива в базу данных, проверяется выполнение условия**.**Если элемент массива удовлетворяет условию, то происходит добавление записи. Если же условие не выполняется, то добавление записи не происходит и выполняется проверка следующей строки массива. Новые записи добавляются столько раз, сколько строк в массиве.

Пример использования данной команды:

\* определяем атрибуты покупателей

DECLARE uCustomer(4,3)

uCustomer(1,1)="Петров"

uCustomer(1,2)={21.09.81}

uCustomer(1,3)=20000

…

uCustomer(4,1)="Иванов"

uCustomer(1,2)={22.10.80}

uCustomer(1,3)=10000

USE Customer && открываем таблицу

\* добавляем данные из массива

APPEND FROM uCustomer FOR CreditLimit>15000

Команда *COPY TO ARRAY -*предназначена для копирования данных из записи таблицы в массив. С помощью этой команды можно использовать массивы для редактирования таб­лиц, копирования записей из одной таблицы в другую, выполнения вычислений над за­писями таблиц и т.п.

Синтаксис команды:

COPY TO ARRAY **массив** [FIELDS списокПолей] [областьОпределения] [FOR **условие1]** [WHILE **условие2]**

Команда **COPY TO ARRAY** позволяет переслать в массив несколько записей. Для ко­пирования записей в массив необходимо предварительно объявить массив командами **DIMENSION, DECLARE** или **PUBLIC.**

В случае одномерного массива, первое поле записи помещается в первый элемент массива, второе поле — во второй элемент массива и т.д. Если число элементов одно­мерного массива превышает число полей в базе данных, оставшиеся элементы массива остаются неизменными. Если же массив имеет меньше элементов, чем полей в базе дан­ных, оставшиеся поля игнорируются.

Двумерные массивы создаются в строчно- столбцовом формате. Запись помещается в строку массива, при этом каждое поле записи занимает отдельный столбец. Для каж­дой записи первое поле хранится в первом столбце массива, второе поле — во втором столбце массива и т.д. Если массив имеет больше столбцов, чем полей в базе данных, ос­тавшиеся элементы массива остаются неизменными. Если же массив имеет меньше столбцов, чем полей в базе данных, оставшиеся поля в массив записаны не будут.

Каждая последующая строка массива заполняется содержимым следующей записи в описанном выше строчно-столбцовом формате. Выполнение команды заканчивается тогда когда, либо в массиве не осталось свободных строк, либо в базе данных кончились записи.

В команде **COPY TO ARRAY** можно указать список копируемых полей. При за­дании списка полей в массив будут записаны только те поля, которые указаны в этом списке.

Если не задан аргумент **область Определения,**Visual FoxPro выполняет копирование записей с начала таблицы. В противном случае, в массив копируются лишь записи, по­падающие в заданную область определения.

При включении в команду конструкции FOR условие1 записи будут заноситься в массив только в том случае, если результат выполнения условия 1 равен логическому значению (.Т.). При включении конструкции WHILE условие2, записи будут помещать­ся в массив только до тех пор, пока результат выполнения условня2 равен (.Т.).

Пример использования команды:

DECLARE uCustomer(4,3)

USE Customer && открываем таблицу

\* копируем данные в массив

COPY TO ARRAY uCustomer FOR CreditLimit>150000

**Операторы**

В Visual FoxРrо существуют три группы операторов:

* Арифметические;
* логические;
* сравнения.

**Арифметические операторы** позволяют выполнять действия над числами, строками или переменными типа дата и время, В таблице приведены арифметические операторы и выполняемые ими функции в зависимости от типа данных.

|  |  |  |  |
| --- | --- | --- | --- |
| **Оператор** | **Числовые** | **Дата**, **время** | **Текстовые** |
| + | Сложение | Увеличение | Сцепление строк |
| — | Вычитание | Уменьшение, разность между двумя значениями | Сцепление строк |
| \* | Умножение | Не совместим | Не совместим |
| / | Деление | Не совместим | Не совместим |

При операциях с типом данных дата приняты следующие соглашения:

|  |  |
| --- | --- |
| **Выражение** | **Действие** |
| Date + Number(или Number + Date) | Прибавляет количество дней. Результат имеет тип Date |
| Date - Number | Уменьшает количество дней |
| Date - Date | Определяет количество днеймежду двумя датами |

Выражения, в которых элементы связаны только арифметическими операторами, называют *формулами.*Формула может содержать более одного оператора. Если в форму­ле отсутствуют скобки, то вначале выполняются операции умножения и деления, а за­тем — сложения и вычитания.

**Операторы сравнения**

В Visual FoxPro допустимы семь операторов сравнения:

|  |  |
| --- | --- |
| **Оператор** | **Значение** |
| = | Равно |
| <>, != | Не равно |
| < | Меньше |
| > | Больше |
| *<=* | Меньше или равно (не больше) |
| >= | Больше или равно (не меньше) |
| = = | Точное совпадение двух значений |

Эти операторы используются для сравнения значений, которые могут быть заданы в виде значении или сложных выражений. Однако сравнивать можно только однотип­ные значения.

Выражение, в котором участвует оператор сравнения, называют *условием.*Результа­том условия может быть только логическое значение: (.Т.), если условие выполняется и (.F.) — в противном случае.:

Пустое значение любого типа считается меньше любого значения соответствующего типа.

**Логические операторы**

В Visual FoxPro существуют три логических оператора AND, OR и NOT (или !), пред­назначенные для записи более сложных условий. Использование этих операторов по­зволяет объединить в одном выражении два и более условий.

Если несколько условий объединены с помощью оператора AND, значение совокуп­ного логического выражения будет истинно только в том случае, если выполняется каж­дое из условий, входящих в его состав. В случае невыполнения хотя бы одного из условий, логическое выражение будет равно (.F.). Совокупное логическое выражение назы­вают *объединенным тестом.*

При использовании оператора OR для объединения простых условий, для того чтобы, объединенный тест был равен (.Т.), достаточно, чтобы хотя бы одно из простых условий было равно (.Т.). Объединенному тексту присваивается значение (.F.) только в том слу­чае, если все простые условия имеют значения (.F.).

Оператор NOT означает логическое отрицание. Например, условие NOT x = 3 будет иметь значение (.Т.) для значений переменной х, не равных 3, и (.F.) для х = 3.

**Комментарии**

Помимо команд и выражений, в программы возможно включение лю­бого произвольного текста или комментария.

Для включения в текст программы комментария необходимо ввести символ «\*» , ко­торый должен быть первым символом в строке. Этот символ означает начало коммента­рия. Любой текст, расположенный в строке следом за этим символом, будет восприни­маться как комментарий, то есть Visual FoxPro не будет транслировать этот текст.

Если требуется расположить комментарий в од­ной строке с командами, используются два символа «&».

Например;

DATE ( )

\* Текущая системная дата

TIME ( ) && Текущее системное время

**Управляющие конструкции Visual FoxPro**

Различают три вида управляющих конструкций:

* ветвление или условные переходы (IF и CASE);
* организация циклов (FOR, WHILE и SCAN);
* завершения или безусловные переходы (EXIT, DO, RETURN и QUIT).

**Конструкция IF ... ENDIF -**выполняет одну из двух последовательностей кодов про­граммы в зависимости от результата проверки условия. Эта конструкция имеет следую­щий синтаксис:

IF условие

**Конструкции 1**

ELSE

**Конструкции 2**

ENDIF

После имени конструкции IF должно следовать логическое выражение, содержащее условие. Ключевое слово ENDIF обозначает конец конструкции и его присутствие в ко­манде обязательно. Если указанное условие выполняется (результат проверки равен (.Т.)), то Visual FoxPro выполнит конструкции, следующие за условием, после чего пе­рейдет к следующей после ENDIF конструкции.

Наличие ключевого слова ELSE не является обязательным. Если указанное условие не выполняется (результат проверки равен (.F.)) и конструкция содержит ключевое сло­во ELSE, Visual FoxPro выполнит последовательность конструкций, расположенных следом за ELSE. После чего управление перейдет к конструкции, следующей после ENDIF. В том случае, если ELSE и следующая за ним последовательность конструкций отсутствует, сразу осуществляется переход к конструкции, следующей за ENDIF.

Например:

IF *х >=*10

IF х > 10

@ 10,10 SAY "Больше 10"

ELSE

@ 10,10 SAY "Равно 10"

ENDIF

ELSE

@ 10,10 SAY "Меньше 10"

ENDIF

Команда IF может проверить только одно условие. Если потребуется осущест­вить переход управления в зависимости от результатов проверки нескольких условий, можно применить вложенные конструкции IF или воспользоваться конструкцией DO CASE/ENDCASE.

**Конструкция IIF**

Конструкция IIF (immediate IF — непосредственно выполняемый IF) аналогична конструкции IF, но в отличие от нее принимает форму функции. Синтаксис конструкции:

IIF(Условие, значениеДляТгuе, значениеДляFalse)

При выполнении условия возвращается значениеДляТтue, в противном случае — значеннеДляFalse.

**Конструкция DO CASE/ENDCASE -**последовательно одно за другим проверяет усло­вия, расположенные после ключевых слов CASE. Если очередное условие выполняется, Visual FoxPro выполняет последовательность конструкций, определенных для данного условия. После завершения выполнения конструкций управление будет передано кон­струкции, следующей за ключевым словом ENDCASE.

Если не выполняется ни одно из условий и конструкция содержит ключевое слово **OTHERWISE**, Visual FoxPro выполнит последовательность конструкций, расположен­ных следом за OTHERWISE. После чего управление перейдет к конструкции, следую­щей после ENDIF. В том случае, если OTHERWISE и следующая за ним последователь­ность конструкций отсутствует, сразу осуществляется переход к конструкции, следую­щей за ENDCASE.

Синтаксис конструкции:

DO CASE

CASE **условие1**

**Конструкция 1**

CASE **условие2**

**Конструкция 2**

**…**

CASE **условиеN**

**Конcтрукция N**

[OTHERWISE

**Конструкции N+1 ]**

ENDCASE

Пример:

DO CASE

CASE x<10

@ 10,10 SAY "Меньше 10"

CASE x=10

@ 10,10 SAY "Равно 10"

CASE x>10

@ 10,10 SAY "Больше 10"

ENDCASE

DO CASE может выполнить не более одной из содержащихся в ней по­следовательностей конструкций. После того, как одно из условий оказалось равно (.Т.) и была выполнена соответствующая последовательность конструкций, DO CASE завер­шит свою работу. Остальные условия проверяться не будут.

**Цикл FOR/ENDFOR -**выполняет последовательность команд определенное число раз. Такую конструкцию называют *циклом,*а выполняемые ею программные коды — *телом цикла.*

Синтаксис цикла FOR/ENDFOR.

FOR **счетчик =**начЗначение **ТО к**онЗначение [STEP **шаг]**

**Конструкции**

ENDFOR

Первый аргумент **счетчик**определяет имя переменной, которая будет «считать» ко­личество выполнений цикла. Цикл выполняется до тех пор, пока значение счетчика не превысит конеч­ное значение, указанное после ключевого слова ТО. После каждого прохода значение счетчика изменяется на величину **шаг,**указанную после ключевого слова STEP. Ключе­вое слово ENDFOR обозначает конец тела цикла и является обязательным.

Перед каждым проходом цикла Visual FoxPro сравнивает значения счетчика и аргу­мента **конЗначение**. Если значение счетчика не превышает установленного значения **конЗначение,**выполняются конструкции тела цикла. В противном случае, управление переходит к следующей за ENDFOR конструкции.

Например:

FOR nCountVar = 1 ТО 10 STEP 2

nNextWeek[nCountVar] = date( ) + nCountVar \* 7

ENDFOR

Здесь цикл выполняется пять раз при значениях счетчика nCountVar 1, 3, 5, 7 и 9. Переменная-счетчик используется в теле цикла в качестве обыч­ной переменной.

Шаг изменения счетчика может быть отрицательным.

Например:

FOR nCounter = 100 ТО 1 STEP -10

nDecades [nCounter] = nCounter

ENDFOR

В этом случае цикл будет выполняться до тех пор, пока nCountVar больше 1. Для от­рицательного значения шага цикла начальное значение счетчика должно быть больше конечного.

Ключевое слово STEP можно опустить, в этом случае значение шага по умолчанию принимается равным 1.

Например:

\* Невыполняемый цикл: начальное значение счетчика больше конечного

\* при положительном шаге цикла

FOR nCounter = 100 ТО 1

nDecades[nCounter] = nCounter

ENDFOR

\* Бесконечный цикл: значение счетчика изменяется в теле цикла и никогда

\* не превысит 10
FOR nCounter FROM 1 ТО 10

nCounter = 1

ENDFOR

**Цикл DO WHILE/ENDDO**

Конструкция FOR/ENDFOR удобна в тех случаях, когда точно известно, сколько раз должна быть выполнена определенная последовательность команд. Однако на прак­тике не всегда заранее можете определить количество выполнений цикла. В таких случаях для организации цикла может быть использована конструкция DO WHILE/ENDDO, которая позволяет выполнять последовательность команд до тех пор, пока указанное условие равно (.Т.).

Синтаксис:

DO WHILE **условие**

**Конструкции**

[EXIT]

[LOOP]

ENDDO

Аргумент конструкции условие является логическим выражением, значение которо­го проверяется перед каждым проходом цикла. Если это значение равно (.Т.), выполня­ется последовательность команд, которые расположены между DO WHILE и ключевым словом ENDDO. Эти конструкции образуют тело цикла. Если при очередном проходе цикла условие равно (.F.), происходит выход из цикла и управление передается конст­рукции, следующей за ENDDO,

**Цикл SCAN/ENDSCAN**

В отличие от конструкций FOR/ENDFOR и DO WHILE/ENDDO, цикл SCAN/ENDSCAN предназначен для работы непосредственно с записями таблицы.

Синтаксис конструкции:

SCAN **Переменная**[FOR **логическоеВыражение1**] [WHILE **логическоеВыражение2]**

**Конструкции**

[LOOP]

[EXIT]

ENDSCAN

Этот цикл просматривает текущую таблицу запись за записью и каждый раз, когда находит запись, удовлетворяющую указанному после ключевого слова FOR условию, выполняет конструкции, предшествующие ключевому слову ENDSCAN. Цикл SCAN начинает работу с первой записи таблицы и заканчивает после просмотра последней за­писи. Для индексированных таблиц сканирование записей осуществляется в порядке, определяемом активным индексом таблицы. Ключевое слово FOR может отсутствовать. В этом случае конструкции цикла будут выполнены для всех записей таблицы.

**Команда EXIT**

В некоторых случаях необходимо прервать выполнение цикла до его завершения. Это можно сделать с помощью команды безусловного перехода EXIT.

Команда EXIT завершает выполнение цикла и передает управление следующей за циклом конструкции.

**Процедуры**

Синтаксис процедуры в Visual FoxPro имеет следующий вид:

PROCEDURE имяПроцедуры

Тело процедуры

RETURN

Возможно хранение каждой процедуры в отдельном файле или объединять несколько процедур в одном файле. Для выполнения процедуры используется команда

DO **имяПроцедуры**

При этом Visual FoxPro осуществляет поиск процедуры в следующем порядке:

1. В файле, из которого осуществляется вызов процедуры.
2. В процедурном файле, который указан в команде SET PROCEDURE.
3. Просматриваются программы в соответствии с иерархией вызовов, начиная с по­следней вызвавшей программы и заканчивая основной программой.
4. Ищется отдельный файл, который имеет такое же имя, как и имя вызываемой проце­дуры.

При вызове процедуры можно указать имя файла, в котором содержится процедура: DO IN имяФайла имяПроцедуры

В этом случае поиск осуществляется только в указанном файле. Для открытия процедурного файла используется команда:

SET PROCEDURE ТО имяФайла

Можно одновременно открыть только один процедурный файл, который можно назвать библиотечным файлом. В этом файле обычно хранятся наиболее часто исполь­зуемые процедуры и функции. Для закрытия процедурного файла используется команда SET PROCEDURE TO без указания имени файла.

**Функции**

Синтаксис функции в Visual FoxPro имеет следующий вид:

FUNCTION имяФункции

Тело функции

RETURN результат

В отличие от процедуры, функция может использоваться в выражениях аналогично стандартным функциям Visual FoxPro. Поиск функции осуществляется в таком же по­рядке, как и поиск процедуры.

**Передача аргументов**

Процедуры и функции могут иметь параметры, которые объявляются в них командой:

PARAMETERS списокПараметров

Эта команда должна быть первой командой процедуры или функции, если они ис­пользуют параметры. Каждый элемент списка параметров разделяется запятой.

При вызове процедуры, содержащей параметры, необходимо использовать следую­щий вариант команды DO:

DO **имяПроцедуры** WITH **списокПараметров**

По умолчанию аргументы (переменные и массивы) передаются в процедуру по ссыл­ке. В этом случае в процедуру передается указатель аргумента и поэтому изменение зна­чений параметров в вызываемой процедуре приводит к изменению значений аргументов в вызывающей программе.

При передаче по значению в процедуру передается копия исходного значения аргу­мента. В вызывающей программе значение аргумента не изменится. При передачи аргу­ментов в процедуру по значению они заключаются в круглые скобки в списке параметров.

По умолчанию в функцию параметры передаются по значению. Если требуется изменить значения передаваемых в функцию аргументов, то есть передавать аргументы по ссылке, используется команда SET UDFPARMS.

Синтаксис команды:

SET UDFPARMS TO VALUE / REFERENCE

Ключевое слово VALUE этой команды устанавливает режим передачи аргументов по значению, а ключевое слово REFERENCE — режим передачи по ссылке.

В приведенных ниже примерах демонстрируются оба метода передачи аргумента.

Пример передачи аргумента по ссылке.

nAutoCod=0

do autoNumber with nAutoCod

@ 10,10 SAY nAutoCod && выводит значение 1

PROCEDURE autoNumber

PARAMETERS nAutoNumber

nAutoNumber=nAutoNumber+1

@ 10,40 SAY nAutoNumber && выводит значение 1

RETURN

Пример передачи аргумента по значению.

nAucoCod=0

do autoNumber with (nAutoCod)

@ 10,10 SAY nAutoCod && выводит значение 0

PROCEDURE autoNumber

PARAMETERS nAutoNumber

nAutoNumber=nAutoNumber+l

@ 10,40 SAX nAutoNumber && выводит значение 1

RETURN