«Азы программирования»

Методический материал по программированию

Встроенный язык программирования. Программные модули

Любой программный модуль в 1С не является самостоятельным, а представляет собой составляющую конфигурации. В модулях происходит выполнение каких-то определенных действий, связанных с тем объектом метаданных, которому принадлежит этот модуль (справочник, документ, отчет и т.д.). Эти действия реализуются с помощью процедур (наборов команд), чаще всего вызываемых при возникновении одного из системных событий: открылась форма, пользователь нажал кнопку на форме и пр. Таким образом, модуль как бы «реагирует» на возникающие ситуации; в нем нет четкой последовательности команд, которые должны быть выполнены по принципу «сверху вниз».

Контекст выполнения модуля

Контекстом выполнения модуля называется связь между данным программным модулем и всей конфигурацией. Существует два типа контекстов выполнения модуля:

глобальный контекст задачи;

локальный контекст выполнения определенного модуля.

Глобальный контекст задачи доступен всем программным модулям и определяет общую языковую среду конфигурации.

В образовании глобального контекста задачи участвуют значения системных атрибутов; системные процедуры и функции; значения, которые заданы в Конфигураторе в виде констант, перечислений, регистров; переменные, процедуры и функции глобального модуля, объявленные с ключевым словом Экспорт.

В свою очередь, назначение локального контекста модуля состоит в том, чтобы предоставить возможность разработчику управлять «деталями» общей задачи.

Формирование локального контекста модуля осуществляется в конкретном объекте метаданных, в котором содержится этот программный модуль. Локальный контекст виден только конкретному программному модулю и определяет тот набор методов, которые доступны именно в этом контексте (т.е. для модуля формы справочника предусмотрены одни методы, для модуля документа - другие, для модуля формы отчета - третьи и т.д.).

Разновидности модулей

У каждой разновидности программного модуля имеются определенные свойства, отличающие его от модулей других типов. Рассмотрим основные типы программных модулей с описанием их свойств.

1. *Глобальный модуль - расположен в корневом разделе конфигурации; запускается в начале выполнения всей задачи (в режиме 1 С:Предприятие); определяет глобальный контекст всей конфигурации.*
2. *Модуль формы элемента справочника - размещается в объекте метаданных Справочник; запускается при открытии формы элемента справочника. В контексте модуля этой формы доступны реквизиты (поля) выбранного элемента справочника и реквизиты (элементы) формы.*
3. *Модуль формы документа - содержится в форме объекта метаданных Документ; запускается при открытии формы документа. В контексте ее модуля доступны реквизиты (поля) активного документа и реквизиты (элементы) его формы.*
4. *Модуль документа - размещается непосредственно в объекте метаданных Документ; запускается при выполнении определенных действий с документом (проведение, удаление проведенного, отмена проведения). В контексте модуля документа доступны реквизиты (поля) активного документа.*
5. *Модуль формы журнала документов - располагается в объекте метаданных Журнал; запускается при открытии формы журнала документов. В контексте ее модуля доступен выбранный в журнале документ и реквизиты формы журнала.*
6. *Модуль формы отчета - размещается в объекте метаданных Отчет; запускается при открытии экранной формы отчета. В контексте модуля отчета доступны реквизиты (элементы) формы отчета.*
7. *Модуль формы обработки - содержится в объекте метаданных Обработка; запускается при открытии экранной формы этого объекта. В контексте модуля объекта Обработка доступны реквизиты (элементы) его экранной формы.*

Структура модуля

Программный модуль состоит из следующих частей:

- *раздел описания переменных;*
- *раздел процедур и функций;*
- *раздел программы.*

Раздел описания переменных - располагается в начале модуля (перед первым оператором Процедура или Функция либо, в случае их отсутствия, - перед первым исполняемым оператором). Здесь указываются операторы ***Перем*** *(объявление переменных).*

Раздел процедур и функций - размещается от первого оператора Процедура или Функция до первого исполняемого оператора, находящегося после процедур (и функций).

Раздел программы - начинается с первого исполняемого оператора после последней процедуры (функции) до конца модуля. Здесь располагаются только исполняемые операторы.

Операторы раздела описания переменных и раздела программы выполняются в тот момент, когда активизируется данный модуль, например при открытии формы документа или отчета.

Как правило, в разделе программы выполняют присваивание переменным модуля каких-то определенных значений, если эти переменные должны получить данные значения перед выполнением процедур (функций) модуля, например перед выполнением процедуры Сформировать для экранной формы отчета.

Операторы языка программирования 1С

В 1С каждый оператор пишется в отдельной строке и оканчивается точкой с запятой (за исключением некоторых случаев, о которых будет сказано особо). Количество пробелов и знаков табуляции между частями оператора не имеет значения.

Для повышения «читабельности» оператор может размещаться нескольких строках, то есть в пределах оператора можно также нажимать Enter. Кроме того, в одной строке можно записывать, несколько операторов, разделяя их точкой с запятой.

Примечание. Для каждого русскоязычного оператора, функции, процедуры и пр. в языке программирования 1С существует англоязычный аналог.

Объявление переменных

Переменные в модуле создаются для того, чтобы хранить какие-либо данные. Перед тем как использовать переменную, ее нужно описать оператором ***Перем*** *(Var):*

Перем <переменная>; В этом операторе:

Перем - ключевое слово, свидетельствующее о том, что осуществляется объявление переменной; <переменная> - имя переменной, которая объявляется.

В одном операторе можно одновременно описать несколько переменных, указывая каждую последующую через запятую.

Если переменную в программе заранее не объявлять, то это действие будет происходить автоматически при первом же присвоении ей какого-либо значения

Ключевые слова

В языках программирования ключевыми (служебными) словами называются такие слова, которые используются только для описания каких-либо операторов и не могут быть использованы программистом ни в каких других целях. В частности, ключевые слова нельзя использовать в качестве имен для переменных в программе.

В 1С каждое ключевое слово имеет два варианта написания -на русском и английском языках. Они могут смешиваться в программе в каком угодно порядке, без ограничений. Перечень ключевых слов приведен в табл. 1.1.

Таблица 1.1 ***Зарезервированные слова языка программирования 1С***

Рус.	Англ.	Рус.	Англ.	Рус.	Англ.
Возврат	*Return*	*Конец Цикла*	*Context*	*Процедура*	*Procedure*
Вопрос	*DoQueryBox*	*Контекст*	*Context*	*Разм*	*Dim*
Дата	*Date*	*Лев*	*Left*	*СокрЛ*	*TrimL*
Для	*For*	*Не*	*Not*	*СокрП*	*TrimR*
Если	*If*	*Окр*	*Round*	*Сред*	*Mid*
Знач	*Val*	*Перейти*	*Goto*	*СтрДлина*	*StrLen*
И	*And*	*Перем*	*Var*	*Строка*	*String*
Или	*Or*	*По*	*To*	*Тогда*	*Then*
Иначе	*Else*	*Пока*	*While*	*Формат*	*Format*
Иначе Если	*ElseIf*	*Прав*	*Right*	*Функция*	*Function*
Конец Если	*EndProcedure*	*Предупреждение*	*DoMessage-Box*	*Цел*	*Int*
Конец-Процедуры	*EndFunction*	*Прервать*	*Break*	*Цикл*	*Do*
Конец-Функции	*EndDo*	*Продолжить*	*Continue*	*Число*	*Number*

Среда программирования 1С настроена таким образом, что при написании текста программы в окне кода все используемые ключевые слова по умолчанию будут обозначаться красным цветом, в то время как имена переменных изображаются синим. В этом случае программист легко может определить, являются ли те слова, которые он планирует использовать, например, в качестве имен для переменных, ключевыми словами 1С или нет.

Комментарии

Справа от любого оператора можно записывать комментарии для него, поясняющие идею той или иной выполняемой операции. Перед началом комментария ставятся две косые черты //, а сам текст комментариев выделяется зеленым цветом:

```
// Это отдельная строка комментария
Перем Список; // Это тоже комментарий
```

В записи оператора строчные и прописные буквы равноценны, поэтому для повышения читаемости текста программы служебные слова обычно начинаются с прописной буквы. Например:

```
Перем СпрСотр;
```

Выбор имени переменной

Когда программист выбирает имя для той или иной переменной в своей программе, он имеет довольно широкие возможности. В частности, переменную, которая будет использоваться в качестве счетчика циклов, можно назвать, например, именем i. Но это название не несет практически никакой смысловой нагрузки. С другой стороны, можно давать переменным длинные составные имена, например КоличествоВыделенныхЭлементов, *которые будут в достаточной степени информативными, но в этом случае на написание подобного названия во всех соответствующих местах программы будет уходить много времени. Поэтому не следует давать переменным очень длинные имена, лучше попытаться найти золотую середину, используя сокращения.*

Существует несколько ограничений на имена используемых в программе переменных:

- *в имени не должно быть точек и пробелов;*
- *необходимо соблюдать уникальность имен переменных в рамках одной рассматриваемой процедуры;*
- *количество символов в имени не должно быть более 255.*

Зачастую для повышения информативности используемых переменных программисты используют в именах так называемые

префиксы, которые определяют принадлежность сохраняемых в них данных к определенному типу (см. табл. 1.2).

Таблица 1.2 Префиксы в именах переменных

Префикс	*Тип переменных*	*Пример*
Спр	*Справочник*	*СпрСотр*
Док	*Документ*	*ДокПрием*
Рег	*Регистр*	*РегТовары*
ТабЗнач	*ТаблицаЗначений*	*ТабЗначСотр*
СписокЗнач	*СписокЗначений*	*СписокЗначСотр*

Область видимости переменных

Когда переменная объявляется в процедуре (или в функции) модуля, область ее видимости ограничивается только этим модулем; за его пределы значение переменной не передается. Такая переменная называется локальной переменной; время ее жизни -пока не закончит выполняться процедура (или функция). Таким образом, в разных процедурах одного и того же модуля можно объявлять переменные с одинаковыми именами - это будут абсолютно разные переменные; после выполнения процедуры модуля их значения будут потеряны. Эти переменные удобно использовать, к примеру, в качестве счетчиков циклов. Если в нескольких процедурах необходимо выполнять циклы, то счетчикам для них можно давать одно и то же имя, например Номер (или Счетчик).

Если объявление переменной выполняется непосредственно в модуле (в начале модуля), а не внутри процедуры, то переменная доступна во всех его процедурах, и ее значение, вычисленное в одной процедуре модуля, можно тут же передать в другую процедуру. Эта переменная называется переменной модуля, время ее существования - пока не будет закрыта форма, которой принадлежит модуль.

Когда переменная объявляется в глобальном модуле с ключевым словом **Экспорт***:* **(Перем ИмяОрг Экспорт;)** *то это 03начает, чт0 переменная будет доступна во всех модулях фигурации. Такая переменная называется глобальной; ее значение доступно во всех процедурах всех модулей конфигурации, а время ее жизни - пока пользователь не окончит работу с данными в режиме 1 С:Предприятие.*

Например, если указать в глобальном модуле такие строки:

Перем Стаж Экспорт; Стаж =20;

а в любом другом модуле (например, в модуле документа) - следующие операторы:

Перем а; а = Стаж;

то переменной а в этом модуле на данном этапе будет присвоено значение 20.

Использование массивов

Массив - это упорядоченная последовательность элементов, обращение к которым осуществляется при помощи его имени и индекса (т.е. порядкового номера элемента). Обычно обработка элементов массива выполняется в циклах, где в качестве индекса выступает счетчик цикла (подробнее о циклах см. раздел «Разновидности циклов»).

Массив объявляется оператором Перем следующим образом:

Перем Масс[10];

В квадратных скобках задается количество элементов массива, а их нумерация в 1С начинается с 1.

Обращение к элементу массива выполняется так:

Масс[1] = 5; Масс[2] = "Элемент";

Элементами массива могут быть значения разного типа: числа, строки и т.д.

Для того чтобы определить размерность массива, т.е. количество его элементов, используется системная функция Разм (Dim), ***например:***

Перем Масс[10]; Размер = Разм(Масс); // Размер =10

Управление средой программирования

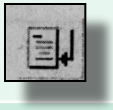
Панель Текстовый редактор

В окне Конфигуратора, когда открыт модуль, становится доступной панель Текстовый редактор, предназначенная для управления комментариями и закладками (закладки в окне модуля предназначены для быстрого перехода к нужным командам), а также для вызова дополнительных средств (проверка синтаксиса, отладка программного кода в модуле и т.д.). Описание кнопок панели Текстовый редактор приведено в табл. 2.1.

Таблица 2.1 Кнопки панели инструментов Текстовый редактор

Кнопка	Название	Описание
	*Установить/снять закладку*	*Позволяет добавить закладку в текущую строку (или снять, если она установлена)*
	*Следующая закладка*	*Выполняет переход к следующей строке с закладкой*
	*Предыдущая закладка*	*Переходит к предыдущей строке с закладкой*
	*Убрать все закладки*	*Удаляет все созданные закладки*
	*Сдвинуть блок вправо*	*Сдвигает выбранный блок операторов вправо (применяется для операторов внутри конструкций)*
	*Сдвинуть блок влево*	*Позволяет сдвинуть выбранный блок операторов влево (применяется для операторов внутри конструкций)*
	*Добавить комментарий*	*Оформляет выбранную строку (или строки) в виде комментария, добавляя перед ней две косые черты*
	*Удалить комментарий*	*Удаляет для выбранной строки (или строк) оформление в виде комментария*

Таблица 2.1 ***Кнопки панели инструментов Текстовый редактор***

Кнопка	Название	Описание
	*Форматировать блок*	*Выполняет автоматическое форматирование выделенного блока операторов: сдвигает их в соответствии с имеющимися в этом блоке вложенными конструкциями языка*
	*Открыть модуль в отладчике*	*Активизирует окно Отладчика и сразу открывает в нем данный модуль*
	*Синтаксический контроль*	*Выполняет проверку программного кода в данном модуле на наличие синтаксических ошибок. Если ошибки найдены, отображает информацию о них в нижней части окна*
	*Переход к строке*	*Позволяет быстро перейти в модуле к строке с указанным порядковым номером*
	*Процедуры и функции модуля*	*Отображает список процедур и функций данного модуля с возможностью быстрого перехода к первой строке выбранной процедуры (или функции)*

<u>Окно сообщений</u>

В том случае, если разработчик допустил ошибку при написании программы, при ее запуске появятся соответствующие сообщения в специальной панели в нижней части экрана, - эта панель носит название Окно сообщений.

Данная панель может быть переведена в оконный режим. Для этого следует щелкнуть правой кнопкой мыши в текстовой области панели и выбрать в контекстном меню команду Перемещать в окне (для возврата в режим панели нужно еще раз выбрать в контекстном меню данный пункт).

Закрытие Окна сообщений в оконном режиме выполняется стандартным образом, а для закрытия Окна сообщений в режиме панели нужно нажать крестик в левом верхнем углу.

Настройка шаблонов

Для повышения продуктивности собственной работы разработчик может создать для себя так называемые шаблоны, применение которых значительно повышает скорость набора программного кода.

Например, если часто используются ключевое слово, завершающее процедуру (КонецПроцедуры), то для него можно создать шаблон кп. Для создания и редактирования шаблонов необходимо выполнить команду меню Сервис \ Настройка шаблонов, после чего отобразится одноименное окно (см. рис. 2.1).

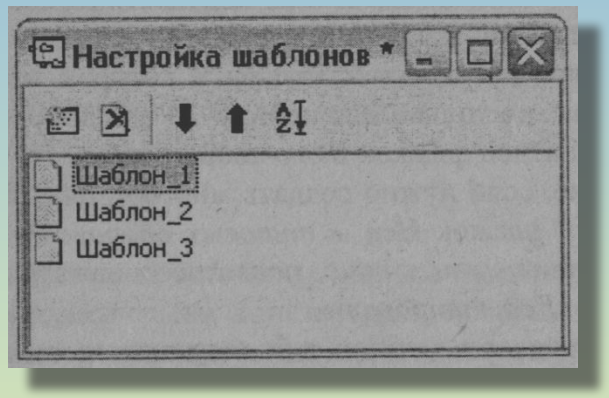


Рис. 2.1 Окно шаблонов

Здесь в верхней части окна имеются стандартные кнопки для управления шаблонами, позволяющие создавать и удалять шаблоны, менять их местами и сортировать по алфавиту.

Настройка параметров системы

Наряду с созданием шаблонов. Перед написанием программного кода следует настроить параметры редактора модулей. Для этого нужно выполнить команду Сервис/Параметры, после чего откроется диалоговое окно Настройка параметров системы. На вкладке модули (Рис. 2.2) можно настроить цвет для различных синтаксических конструкций

(выделяем тип синтаксической конструкции из списка и выбираем нужный цвет). Флажок ***Запретить выделение*** *цветом отменяет все цветовые схемы для синтаксических конструкций, которые в этом случае будут отображаться черным цветом. Но цветовые схемы сохраняются и могут быть восстановлены, если снять этот флажок.*

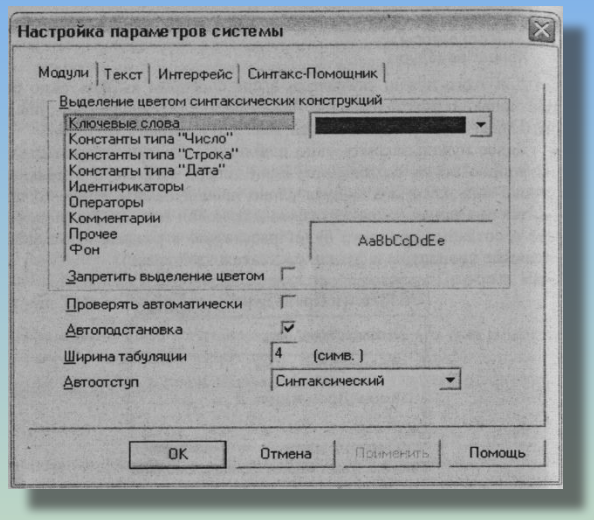


Рис. 2.2 Настройка параметров системы

Флажок ***Проверять автоматически*** *активизирует автоматический режим проверки синтаксиса для программного кода. Если разработчик допустит какую-либо ошибку, то при попытке закрыть окно текст модуля не будет сохранен автоматически, а появится предупреждение о наличии ошибок.*

Флажок ***Автоподстановка*** *задает режим, при котором шаблоны автоматически подставляются вместо введенных сокращений (установлен по умолчанию).*

В поле ***Ширина*** *табуляции задается (в символах) величина (в диапазоне от 1 до 16) отступа при нажатии клавиши Tab.*

В раскрывающемся списке ***Автоотступ****, устанавливается стиль отступа операторов в управляющих конструкциях, когда в конце очередной строки нажимается клавиша Enter*

Диалог с пользователем

В 1С реализовано множество специальных функций, организующих диалог с пользователем:

- *отображение сообщений для пользователя;*
- *реагирование пользователем на вопрос в окне сообщения;*
- *ввод пользователем значений определенного типа.*

Рассмотрим более подробно, какие функции доступны в каждой из указанных категорий. Но перед этим познакомимся с функцией Сигнал (веер), которая используется для подачи короткого звукового сигнала (функция без параметров):

Сигнал();

Отображение сообщений

Для того чтобы вывести сообщение в отдельном окне, применяется функция Предупреждение (DoMessageBox), которая имеет следующий синтаксис:

Предупреждение(<Текст>, <Задержка>)

Эта функция выводит в окне сообщение, заданное параметром <Текст>, с указанием задержки в секундах, устанавливаемой параметром <Задержка> (этот параметр необязателен). Если параметр <Задержка>, который определяет, как долго окно предупреждения будет отображаться на экране, не задан либо установлен равным 0 (значение по умолчанию), то интервал задержки считается бесконечным. ***Например:***

Предупреждение(«операция завершена», 7);

Кроме того , информация для пользователя может помещаться в Окне сообщений. Для этого используется функция Сообщить (Message), которая имеет такой синтаксис:

Сообщить(<Текст>,<Маркер>)

Функция отображает <Текст> в Окне сообщений и имеет необязательный параметр <Маркер>, задающий тип пиктограммы слева от строки сообщения. ***Например:***

Сообщить («Маркер I», «I»);

Сообщить («Обычный маркер», «.»);

Сообщить («Без маркера», « »);

Для очистки Окна сообщений предназначена функция ОчиститьОкноСообщений (ClearMessageWindow). Не имеющая параметров:

ОчиститьОкноСообщений();

Сообщения для пользователя можно выводить также в строке состояния (в левом нижнем углу экрана). Для этого применяется функция состояние (Status). ***Например:***

Состояние(«Операция завершена»);

Отображение вопросов

Чтобы отобразить вопрос, в ответ на который пользователь должен нажать в окне какую-либо кнопку, применяется функция Вопрос(DoQueryBox). Она имеет такие параметры:

Вопрос(<Текст>, <Режим>, <Задержка>)

Здесь, как и в функции Предупреждение, в окне отображается <текст> на время, заданное параметром <Задержка >. Но есть еще дополнительный параметр <Режим>, которым определяется режим отображения вопроса (т.е. набор кнопок, которые присутствуют в окне).

Параметр <Режим> может быть задан либо в виде числа, либо в виде строки. В табл. 3.1 приведены все возможные числовые значения параметра <Режим> и соответствующие им русскоязычные и англоязычные варианты.

Таблица 3.1 Значения параметра <Режим>

Число	Рус	Англ
0	ОК	OK
1	ОК+Отмена	OK+Cancel
2	Стоп+Повтор+Пропустить	Abort+ Retry+Ignore
3	Да+Нет+Отмена	Yes+No+Cancel
4	Да+Нет	Yes+No
5	Повтор+Отмена	Retry+Cancel

Например:

```
Текст = «подтвердите завершение операции»;
Если Вопрос(Текст. «Ок+Отмена») = «ОК» Тогда
    Предупреждение(«Операция успешно завершена»)
Иначе
    Предупреждение(«Операция не завершена»);
КонецЕсли;
```

В этом случае появится окно с текстом Подтвердите завершение операции и двумя кнопками «ОК» и «Отмена». Если пользователь нажмет на кнопку «ОК», то появится сообщение «Операция успешно завершена», в обратном случае появится сообщение «Операция не завершена»

Как видно из этого примера, результат функции Вопрос используется в конструкции Если...Тогда, чтобы выяснить, какую кнопку нажал пользователь, и, исходя из этого, в программе реализуются определенные действия.

Отображение запросов на ввод значений

Для ввода пользователем данных различного типа разработчик может отображать специальные диалоговые окна. Функции, выполняющие эти действия.

Функция ВвестиЗначение (InputValue) позволяет отобразить диалог для ввода пользователем значения заданного типа и имеет такой синтаксис:

ВвестиЗначение(<Переменная>, <Заголовок>, <Тип>, <Длина>, <Точность>)

где < Переменная> - имя переменной, в которую будет занесено введенное пользователем значение (должна быть предварительно объявлена);

<Заголовок> - строковое значение, указываемое в качестве подсказки для пользователя в заголовке диалогового окна; <Тип> - тип данных для вводимого значения (например, "Число", "Строка", "Дата", "Справочник.Сотрудники" и т.д.). <Длина> - длина вводимого значения (в символах); используется для типов данных "Число" и "Строка" (необязательный параметр);

<Точность> - количество десятичных знаков для значения типа "Число" (необязательный параметр).

Функция ВвестиЗначение возвращает 1, если пользователь нажал в диалоге кнопку ОК, или возвращает 0, если была нажата кнопка Отмена.

Пример:
Перем Зарплата;
Принято = 0;
Пока Принято = 0 Цикл
Ввод = ВвестиЗначение(Зарплата,
"Введите размер зарплаты", "Число", 6, 2); Если Ввод = 1 Тогда
Если Зарплата < 0 Тогда
Предупреждение("Некорректное значение. Повторите ввод") Иначе
Предупреждение("Зарплата установлена"); Принято = 1 КонецЕсли Иначе
Предупреждение("Зарплата не установлена"); Принято = 1 КонецЕсли КонецЦикла;

*Функция **ВвестиЧисло (InputNumeric)** служит для отображения запроса на ввод пользователем числового значения и имеет следующий синтаксис:*

ВвестиЧисло(<Переменная>, <Заголовок>, <Длина>, <Точность>, <Задержка>)

Здесь, как и в функции ВвестиЗначение, задается имя предварительно объявленной <Переменной>, в которой будет сохранено введенное пользователем число заданной <Длины> и <Точности>. Диалог с <Заголовком> будет отображаться на экране определенный промежуток времени в секундах, равный <Задержке> (необязательный параметр).

Что касается ввода пользователем числовых значений, в качестве вспомогательного средства можно воспользоваться Калькулятором, вызываемым щелчком мыши на кнопке справа от поля ввода.

*Функция **ВвестиСтроку (InputString)** применяется для отображения запроса на ввод текста пользователем и имеет такой синтаксис:*

ВвестиСтроку(<Переменная>, <Заголовок>, <Длина>, <Признак>, <Задержка>);

Назначение параметров аналогично рассмотренным выше функциям. Отличие состоит в необязательном параметре <Признак>, который определяет, будет ли вводиться однострочный текст (значение 0, принято по умолчанию) или многострочный текст (значение 1).

Например, предложим пользователю ввести фамилию сотрудника (максимальная длина строки - 20 символов):

Перем Фамилия;
Ввод = ВвестиСтроку(Фамилия,
"Введите фамилию сотрудника", 20); Если Ввод = 1 Тогда
Предупреждение("Фамилия введена") Иначе
Предупреждение("Фамилия не введена") КонецЕсли;

Далее рассмотрим пример ввода пользователем многострочного текста. Будет отображен запрос на ввод ФИО, в котором нужно ввести в отдельных строках фамилию, имя и отчество (Рис. 3.1)

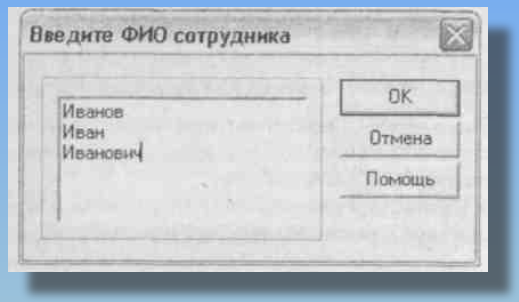


Рис. 3.1

Перем ФИО;
Ввод = ВвестиСтроку(ФИО, "Введите ФИО сотрудника", 50, 1); Если Ввод = 1 Тогда
Фамилия = СтрПолучитьСтроку(ФИО, 1);
Имя = СтрПолучитьСтроку(ФИО, 2); Отчество = СтрПолучитьСтроку(ФИО, 3); . Предупреждение("ФИО введено:" + Симв(13) +
Фамилия + " " + Имя + " " + Отчество)
Иначе
Предупреждение("ФИО не введено")
КонецЕсли;

*Функция **ВвестиДату (InputDate)** предназначена для запроса пользователю на ввод даты и имеет такой синтаксис:*

ВвестиДату(<Переменная>, <Заголовок>, <Задержка>)

*Функция **ВвестиПериод (InputPeriod)** позволяет установить период (т.е. временной интервал) и часто используется в отчетах и обработках. Она имеет такой синтаксис:*

ВвестиПериод(<Начало, <Конец>, <Заголовок>)

Переменная <Начало> сохраняет дату начала периода, а переменная <Конец> - дату окончанию периода, которые заданы пользователем в диалоговом окне с названием <Заголовок>.

Кроме того, переменным <Начало> и <Конец> могут быть предварительно присвоены определенные значения. В этом случае они отобразятся в диалоге как исходные.

Типы данных

Тип данных задается переменной для того, чтобы определить способ ее хранения и обработки (базовый тип) или указать, с какими объектами она будет работать (агрегатный тип).

Базовые типы данных

К базовым типам данных относятся числовой, строковый и тип даты.

Числовой тип служит для хранения любых десятичных чисел в переменных и выполнения основных арифметических операций. При этом в качестве разделителя целой и дробной частей числа выступает точка. ***Например:***

Стаж = 30;

Стаж1 = Стаж + 0.5;

Строковый тип предназначен для хранения последовательностей символов, а также «пустой строки». Строковые значения указываются в двойных кавычках.

Например:

Стаж = "30 лет";

Тип даты предусмотрен для хранения дат в любом из стандартных форматов (в частности, ДД.ММ.ГГГГ или ДД.ММ.ГГ). Значения дат указываются в одиночных кавычках:

Стаж = '30.07.1977';

Тип данных переменной может изменяться по ходу выполнения программы.

Агрегатные типы данных

Агрегатными типами данных называются специальные типы, предназначенные для работы с объектами метаданных и другими вспомогательными объектами (таблицами значений и пр.). В отличие от базовых типов, которые определяются по конкретным значениям, агрегатные типы либо задаются с помощью специальной функции СоздатьОбъект, либо значение с агрегатным типом данных уже известно из глобального контекста задачи или из локального контекста модуля.

Другое отличие от базовых типов состоит в том, что каждый агрегатный тип данных содержит набор определенных свойств (атрибутов) и методов (действий). Атрибуты управляют значениями (в этом смысле они аналогичны переменным), а методы выполняют определенные действия.

Рассмотрим стандартную последовательность действий при работе с агрегатным типом данных, описывающим объект метаданных.

1. *Создаем объект агрегатного типа с помощью функции СоздатьОбъект.*
2. *Ассоциируем этот объект с определенным объектом метаданных конфигурации.*
3. *Выполняем манипуляции с этим объектом с помощью изменения атрибутов и вызова методов.*

Например, выполним в справочнике Должность поиск указанной пользователем должности и выведем соответствующее сообщение (найдена такая должность или нет).

Ввод пользователем названия должности будет реализован в виде стандартного диалогового окна, вызываемого функцией ВвестиСтроку, а результат будет занесен в предварительно объявленную переменную Поиск:

Перем Поиск;

ВвестиСтроку(Поиск, "Поиск должности", 50);

Второй параметр функции ВвестиСтроку - это заголовок окна, а третий параметр - длина вводимой строки.

Далее функцией СоздатьОбъект выполним создание объекта агрегатного типа Справочник.Должности и присвоим его переменной Должн:

Должн = СоздатьОбъект("Справочник.Должности");

Сейчас переменной Должн не соответствует никакой объект, она содержит «пустое» значение, своеобразную «заготовку» для объекта с заданным агрегатным типом. Чтобы установить ассоциацию этой переменной с конкретным объектом (в данном случае с записью в справочнике), нужно использовать специальные методы. Для записи в справочнике такими методами являются НайтиЭлемент, НайтиПоКоду, НайтиПоНаименованию, Получить Элемент. Если же необходимо ассоциировать переменную с объектом Документ, предусмотрены следующие методы: НайтиДокумент, НайтиПоНомеру, ПолучитьДокумент.

Очередной этап - с помощью метода НайтиПоНаименованию (используемого только в объектах-справочниках) осуществляем поиск указанной пользователем должности:

Должн.НайтиПоНаименованию(Поиск, , 1);

Здесь первый параметр - это искомая строка, второй параметр (необязательный) пропущен, а значение третьего параметра установлено равным 1 (поиск строгого соответствия). Затем выполним проверку (с использованием конструкции Если...Тогда) функцией Выбран, которая возвращает 1, если поиск завершен удачно.

Если Должн.Выбран() = 1 Тогда

Предупреждение("Должность " + Поиск + " найдена"); Иначе

Предупреждение("Должность " + Поиск + " не найдена"); КонецЕсли;

Арифметические и логические выражения

Арифметические выражения

При использовании в программе каких-либо арифметических вычислений их представление на языке 1С схоже с математическим. Выражения могут содержать числа, переменные, функции, которые соединены между собой знаками арифметических действий. Кроме обычных арифметических действий (сложения (+), вычитания (-), умножения (*) и деления (/)), в языке 1С можно также определять остаток от деления (%). Например:

Перем m, n, x m = 5 n = 2

x = m/n //x = 2.5

x = m % n // x = 1

При наличии в выражении нескольких арифметических операций порядок их выполнения определяется правилами приоритета.

1. *Умножение и деление (*, /).*
2. *Остаток от деления (%).*
3. *Сложение и вычитание (+,-).*

Операции с одинаковым приоритетом выполняются в соответствии с порядком их записи в операторе слева направо. Если в выражении какие-либо операции заключены в скобки, то независимо от приоритета они выполняются в первую очередь.

Математические функции

В языке 1С для решения различных математических задач существуют встроенные функции, зависящие от одного аргумента, которые можно использовать непосредственно при вычислении каких-либо выражений.

Функция Окр (Round) позволяет округлять заданное число с указанной точностью и имеет такой синтаксис:

Окр(<Число1>, [<Число2>], [<Способ>])

<число1> - округляемое числовое значение; <Число2> - значение, определяющее формат округления, т.е. количество десятичных знаков (если Число2 > 0) или количество знаков целой части (если Число2 < 0), до которых оно проводится. Значение по умолчанию = 0; < Способ >-устанавливает способ округления:

0 (по умолчанию) - если при округлении 1.5 = 1;

1- если при округлении 1.5 = 2.

Функция Цел (Int) позволяет получить целую часть от указанного числового значения (отсекая его дробную часть) и имеет такой синтаксис:

***Цел(<Число>) где <Число>** - заданное числовое значение (или выражение).*

Например:

Зарплата = 546.455;

БезКопеек = Цел(Зарплата); // БезКопеек = 546

Функция Мин (Min) возвращает минимальное значение из набора указанных значений и имеет следующий синтаксис:

Мин (<Значение1>, ... , <ЗначениеN>) *где <Значение1>, ... , <ЗначениеN > - сравниваемые значения.*

Аналогичный синтаксис имеет функция Макс (Max), которая возвращает максимальное значение среди указанных. Например:

Стаж1 = 20;'

Стаж2 = 25;

Стаж3 = 15;

МинСтаж = Мин(Стаж1, Стаж2, Стаж3); // МинСтаж = 15

МаксСтаж = Макс(Стаж1, Стаж2, Стаж3); // МаксСтаж = 25

Функция Лог (Ln) вычисляет натуральный логарифм указанного значения и имеет следующий синтаксис:

Лог(<Число>)

<Число> - заданное числовое значение (или выражение).

Логические операции

Помимо математических, в языке 1С можно также вычислять значения логических выражений, которые могут принимать одно из двух значений: истина или ложь.

Прежде всего рассмотрим виды операций сравнения, определенных в 1С (см. Таблицу 4.1)

Операция	Выражение	Описание
больше	Знач1 > Знач2	(для чисел, строк или дат)
больше или равно	Знач1 >= Знач2	(для чисел, строк или дат)
меньше	Знач1 < Знач2	(для чисел, строк или дат)
меньше или равно	Знач1 <= Знач2	(для чисел, строк или дат)
равно	Знач1 = Знач2	(для чисел, строк, дат или агрегатных типов)
не равно	Знач1 <> Знач2	(для чисел, строк, дат или агрегатных типов)

Таблица 4.1

Теперь рассмотрим логические функции, которые реализованы в том языке.

Конъюнкция (логическое И). Это действие выполняется функцией И (AND), которая возвращает истина, если оба ее операнда имеют значение истина; в противном случае возвращает ложь. ***Пример***

```
цена1 = 3000; цена2 = 2000; Если (Цена1 > 1000) И (Цена2 > 1000) Тогда
Вывод = "Все дорого" Иначе
Вывод = "Почти все дорого" КонецЕсли; Предупреждение(Вывод);
```

Дизъюнкция (логическое ИЛИ). Данное действие реализуется функцией ИЛИ (OR), которая возвращает истина, если хотя бы один из двух ее операндов имеет значение истина, в противном случае возвращает ложь. ***Например:***

```
Цена1 = 2000;
Цена2 = 500;
Если (Цена1 < 1000) ИЛИ (Цена2 < 1000) Тогда
Вывод = "Хоть что-то дешево" Иначе
Вывод = "Все дорого" КонецЕсли; Предупреждение(Вывод);
```

Отрицание (логическое НЕ). Это действие реализует функция НЕ (NOT), которая имеет только один операнд и возвращает противоположное ему значение, ***например:***

```
Цена1 = 2000;
Если НЕ (Цена1 < 1000) Тогда
Вывод = "Дорого" Иначе
Вывод = "Дешево" КонецЕсли; Предупреждение(Вывод);
```

У перечисленных логических функций предусмотрена определенная очередность их выполнения, если они одновременно присутствуют в одной команде. Приоритетность выполнения операций следующая:

1. *выражение в круглых скобках;*
2. *отрицание НЕ;*
3. *Дизъюнкция ИЛИ;*
4. *конъюнкция И.*

Конструкция принятия решений

Часто в определенном месте программы необходимо выполнять те или иные операторы в зависимости от некоторых условий. Эта возможность в 1С реализуется при помощи так называемых управляющих конструкций (или структур), которые, в свою очередь, состоят из структур принятия решений и циклов. Ниже приводится подробное описание конструкции принятия решений Если...Тогда.

Существует несколько разновидностей данной структуры. Если проверяется только одно условие, то конструкция выглядит таким образом (слева приведен русскоязычный вариант, справа -англоязычный):

Если <условие> Тогда If <условие> Then

<операторы> <операторы>

КонецЕсли Endlf

Если <условие> истинно, то будут выполнены <операторы>, которые расположены после ключевого слова Тогда. Если же <условие> ложно, то выполнится следующий после данной конструкции оператор.

Вторая разновидность конструкции применяется, если нужно выполнять тот или иной оператор (или блок операторов) в зависимости от результата проверки условия:

Если<условие> Тогда If <условие> Then

<операторы1> <операторы1>

Иначе Else

<операторы2> <операторы2>

КонецЕслиё Endlf

Если проверяемое <условие> истинно, выполняется блок нераторы1> после ключевого слова Тогда. Если же <условие> ложно то выполнится блок <операторы2> после служебного слова Иначе.

В том случае, когда определенное действие (или набор действий) требуется выполнять после проверки не одного, а нескольких условий, на языке 1С следует использовать такую управляющую структуру:

Если <условие1> Тогда If <условие1> Then

<операторы1> <операторы1>

ИначеЕсли <условие2> Тогда Elself <условие2> Then

<операторы2> <операторы2>

ИначеЕсли <условие3> Тогда Elself <условие3> Then

<операторы3> <операторы3>

Иначе <операторы14> КонецЕсли Else <операторыM> Endlf

Разновидности циклов

Кроме структуры принятия решений Если...Тогда существует еще одна разновидность управляющих конструкций, называема циклом. Цикл - это алгоритмическая структура, при помощи которой реализуется многократное повторение блоков операторов

В языке 1С существует два основных вида циклов, которые реализуются при помощи описываемых ниже конструкций ***Для...КонецЦикла и Пока...КонецЦикла****.*

Цикл ***Для.. .КонецЦикла***

Эта разновидность применяется в том случае, когда количество повторов заданного блока операторов известно заранее. Синтаксис конструкции Для...КонецЦикла таков:

Для <счетчик> = <нач_знач> По <кон_знач> Цикл

<операторы> КонецЦикла

For <счетчик> = <нач_знач> To <кон_знач> Do

<операторы> EndDo

При первом проходе цикла переменной <счетчик> присваивается исходное значение <нач_знач>, после чего возможны два варианта действий. Если проверка условия счетчик > кон_знач вернула значение истина, то цикл завершится, при этом блок <операторы> ни разу не выполнится. Если же проверка этого условия вернет ложь, то блок <операторы> выполняется первый раз, после чего осуществляется переход на начало цикла.

Затем переменная <счетчик> увеличивается на 1, после чего снова выполняется проверка истинности условия счетчик > кон_знач и т.д. Цикл заканчивается в тот момент, когда результатом данной проверки станет истина.

Цикл ***Пока...КонецЦикла***

Этот цикл используется в ситуациях, когда число повторений операторов тела цикла заранее неизвестно, и имеет следующий синтаксис:

Пока <условие> Цикл While <условие> Do

<операторы> <операторы>

КонецЦикла EndDo

Если <условие> истинно, то происходит циклическое выполнение блока <операторы>; когда <условие> становится ложным, выполнение цикла прекращается.

Управление выполнением циклов

Зачастую возникает необходимость в аварийном завершении работы цикла при истинности какого-либо дополнительного условия. В этом случае внутри цикла применяется ключевое слово Прервать (Break), которое обычно указывается в управляющей конструкции Если...Тогда:

<операторы1> <операторы1>

Если <условие> Тогда If <условие> Then

Прервать Break

КонецЕсли Endlf

<операторы2> <операторы2>

Если <условие> будет истинным, то выполнение цикла прекратится, причем блок <операторы1> выполнится очередной раз, в отличие от блока <операторы2>.

Например, будем циклически выводить сообщение об отчете за определенный год, начиная с 1997 и заканчивая 2004 (год будет храниться в переменной ГодОтчета). Когда последовательность лет дойдет до 2000, будет выполнен аварийный выход из цикла, и сообщение об отчете за 2000 год не отобразится. Вместо него будет выведено сообщение «Ошибка 2000»:

```
ГодОтчета = 1997;
Пока ГодОтчета < 2005 Цикл
    Если ГодОтчета = 2000 Тогда
    Предупреждение("Ошибка " + ГодОтчета);
    Прервать
КонецЕсли;
   Предупреждение-("Отчет за " + ГодОтчета + " год");
   ГодОтчета = ГодОтчета + 1
КонецЦикла;
```

Теперь рассмотрим другую ситуацию, когда нужно пропустить какое-то значение и продолжить выполнение цикла дальше.

Чтобы при истинности заданного условия пропустить блок операторов в конце цикла и перейти к его следующему шагу, нужно воспользоваться ключевым словом Продолжить (Continue), указываемого в управляющей конструкции:

<операторы1> *<операторы1>*

If <условие> Then *Если <условие> Тогда*

Continue Endlf *Продолжить*

КонецЕсли EndIf

<операторы2> *<операторы2>*

Если проверка <условия> вернет значение истина, то на данном шаге цикла блок <операторы2> будет пропущен и выполнится переход на начало цикла.

Вложенные циклы

Важной особенностью использования циклов является то, что с их помощью можно создавать так называемые вложенные циклы, когда один цикл располагается внутри другого. То есть для каждого значения счетчика внешнего цикла «пробегаются» все значения счетчика внутреннего цикла.

В качестве примера рассмотрим программу, которая будет выводить сообщения об отчетах за каждый квартал каждого года за период 1997-2004 гг. За основу возьмем предыдущий пример, дополнительно указав внутренний цикл по кварталам (переменная Квартал будет принимать значения от 1 до 4).

Использование вложенных циклов

```
ГодОтчета = 1997; Пока ГодОтчета < 2005 Цикл Для Квартал = 1 По 4 Цикл
Предупреждение ("Отчет за " + Квартал +"-й квартал " + ГодОтчета + " года");
КонецЦикла;
ГодОтчета = ГодОтчета + 1
КонецЦикла;
```

Таким образом, строка для сообщения будет складываться из номера квартала (переменная Квартал) и года (переменная ГодОтчета). Первое сообщение будет об отчете за 1-й квартал 1997 года, а последнее – об отчете за 4-й квартал 2004 года.

Работа со строками

Строковыми называются такие переменные, которые предназначены для работы с текстом или с какими-либо строковыми константами, обозначаемыми в тексте программы в двойных кавычках. Например:

Отчетность = "Отчет за 2004 год";

Строковые константы могут быть как однострочными, так и многострочными. В последнем случае возможны два варианта действий. Во-первых, можно указать составляющие этой строковой константы в отдельных строках, но при этом между ними не должно быть никаких символов, кроме пробелов, переводов строки и комментариев. Кавычки для каждой строки должны закрываться. Например:

ФИО = "Иванов " // первая строка константы

"Иван " // вторая строка константы

"Иванович"; //третья строка константы

// ФИО = "Иванов Иван Иванович"

Второй способ состоит в использовании символа «|» для разделения строк. В этом случае никаких лишних символов, кроме перевода строки, быть не должно, так как они автоматически войдут в итоговое значение. Кавычки здесь закрываются только в последней строке. Например:

ФИО = "Иванов

| Иван

| Иванович"; // только здесь можно задать комментарий

// ФИО = "Иванов Иван Иванович"

Удаление лишних пробелов

Для удаления лишних пробелов в начале и в конце строки используются следующие функции: СокрЛ (TrimL), СокрП (TrimR) и СокрЛП (TrimAll):

СокрЛ - удаляет все пробелы в начале строки;

СокрП - удаляет все пробелы в конце строки;

СокрЛП - удаляет все пробелы в начале и в конце строки.

Например:

Должность = " Бухгалтер ";

Результат = СокрЛ(Должность); // Результат = "Бухгалтер " Результат = СокрП(Должность); // Результат = " Бухгалтер" Результат = СокрЛП(Должность);// Результат = "Бухгалтер"

Объединение строк

При объединении двух и более строк используется операция, называемая конкатенацией, которая реализуется с помощью оператора «+».

Например:

```
Имя = "Иван";
Фамилия = "Иванов";
Отчество = "Иванович";
ФИО = Фамилия + " " + Имя + " " + Отчество;
//ФИО = "Иванов Иван Иванович"
```

Пустые строки

Чтобы проверить, содержатся ли в заданной строке какие-либо значащие символы (т.е. любые символы, кроме пробелов), применяется функция ПустаяСтрока (IsBlankString), которая возвращает значение 0, если строка непустая (т. е. если есть значащие симвоы)> и значение 1, если строка пустая (нет ни одного символа либо только пробелы).

Например:

```
Фамилия = "    ";
Если ПустаяСтрока(Фамилия) = 1 Тогда
Предупреждение("Фамилия не введена!");
КонецЕсли;
```

Длина строки

Чтобы определить количество символов в строке, используется функция СтрДлина (StrLen). ***Например:***

```
ФИО = "Иванов Иван Иванович"; ДлинаФИО = СтрДлина(ФИО); // ДлинаФИО = 20
```

Строчные и прописные буквы

Для преобразования строки таким образом, чтобы все ее буквы были в верхнем регистре (ВСЕ ПРОПИСНЫЕ), используется функция Врег (Upper). Соответственно, для преобразования в нижний регистр применяется функция Нрег (Lower).

Например:

```
ФИО = "Иванов Иван Иванович";
ФИО = Врег (ФИО);       // ФИО = "ИВАНОВ ИВАН ИВАНОВИЧ"
ФИО = Нрег (ФИО);       // ФИО = "иванов иван иванович"
```

Поиск и замена подстроки

Чтобы определить, входит ли заданная подстрока в указанную строку, и если да, то с какого символа, используется функция Найти (Find), имеющая такой синтаксис:

Найти(<СтрокаОсн>, <СтрокаИск>)

где <СтрокаОсн> - строка, в которой выполняется поиск; <СтрокаИск> - искомая подстрока.

Если подстрока <СтрокаИск> найдена в строке <СтрокаОсн>, то результатом функции Найти является порядковый номер первого символа искомой подстроки в указанной строке. Если же подстрока не найдена, то функция Найти возвращает 0.

Например:

ФИО = "Петров Иван Семенович"; НайтиИмя = Найти(ФИО, "Иван"); // НайтиИмя = 8

Если искомая подстрока может встретиться в основной строке несколько раз, то для подсчета количества этих совпадений применяется функция СтрЧиспоВхождений (StrCountOccur), имеющая следующий синтаксис:

СтрЧислоВхождений(<СтрокаОсн>, <СтрокаИск>)

<СтрокаОсн> - строка, в которой выполняется поиск; < СтрокаИск> - искомая подстрока.

Например:

```
ФИО = "Иванов Иван Иванович";
НайтиПодстроку = СтрЧислоВхождений(ФИО, "Иван");
// НайтиПодстроку = 3
```

Для замены определенной подстроки в заданной строке служит функция СтрЗаменить (StrReplace), которая имеет следующий синтаксис:

Заменить(<СтрокаОсн>, <СтрокаИск>, <СтрокаЗам>)

где <СтрокаОсн> - строка, в которой выполняется замена; <СтрокаИск> - искомая подстрока; <СтрокаЗам> - подстрока замены.

Результатом функции СтрЗаменить является либо итоговая строка с заменой, либо исходная строка <СтрокаОсн>, если заданная подстрока <СтрокаИск> не была найдена.

Примечание. Если в исходной строке имеется несколько совпадений с искомой подстрокой, то все они заменяются.

Например:

```
ФИО = "Иванов Иван Иванович";
ЗаменитьФИО = СтрЗаменить(ФИО, "Иван", "Петр");
// ЗаменитьФИО = "Петров Петр Петрович"
```

Преобразование кодов и символов

Для преобразования кода символа в соответствующий ему символ применяется функция Симв (Chr), возвращающая полученный символ. ***Пример:***

Отображение текста на нескольких строках

Фамилия = “Иванов";

Имя = “Иван”;

Отчество = “Иванович”;

Результат = Фамилия + Симв(13) + Имя + Симв(13) + Отчество;

Предупреждение(Результат);

В результате получим сообщение с ФИО в несколько строк, т.к. код 13 – символ Enter.

Выделение части строки

Нередко в программе возникает необходимость выделения из строковой переменной той или иной ее части. В этом случае применяются следующие функции: Лев (Left), Прав (Right) или Сред (Mid).

Функция Лев применяется для выделения подстроки в начале исходной строки и имеет следующий синтаксис:

Лев(<строка>, <количество>)

Результатом функции Лев является левая часть <строки>, имеющая заданное <количество> символов.

Функция Прав используется для выделения подстроки в конце исходной строки и имеет такой синтаксис:

Прав(<строка>, <количество>)

Результатом функции Прав является правая часть <строки>, имеющая заданное <количество> символов.

Функция Сред применяется для выделения подстроки в середине исходной строки и имеет следующий синтаксис:

Сред(<строка>, <номер>, [количество])

Результатом функции Сред является строка с указанным <количеством> символов, находящаяся в исходной <строке> начиная с определенного <номера>. Если <количество> символов не указывается (это необязательный параметр), то строка выделяется до конца; таким образом, в этом случае Сред будет аналогом функции Прав, но с другим параметром:

Прав: <количество> означает, сколько символов выделять; Сред: <номер> означает, с какого символа начинать.

Ниже приводится пример использования рассмотренных функций с определением отдельно фамилии, имени и отчества указанного сотрудника.

Работа с датой и временем

Как уже было сказано выше, значения типа Дата должны указываться в тексте программы в одиночных кавычках, в формате ДД:ММ:ТТ. При этом они могут обрабатываться как обычные строковые значения. В частности, можно извлекать часть из значения даты (число, месяц или год) по отдельности.

С помощью функции Текущая дата (CurDate), указываемой без параметров, можно получить системную дату и вычислить, какой сегодня год, месяц и день. ***Пример:***

```
Обработка даты как строкового значения
ТекДата = ТекущаяДата();
День = Лев(ТекДата, 2);
Месяц = Сред(ТекДата, 4, 2);
Год = Прав(ТекДата, 2);
Сегодня = "Текущая дата: " + ТекДата + Симв(13) + Симв(13) + "Год: " + Год + Симв(13)+ "Месяц: " + Месяц + Симв(13) + "Число: " + День;
```

Рабочая дата

Для того чтобы установить или вернуть рабочую дату, служит функция РабочаяДата (WorkingDate), которая имеет следующий синтаксис:

РабочаяДата(<Дата>, <Режим>)

Все параметры этой функции необязательны. Если они не указываются, то функция РабочаяДата возвращает значение рабочей даты для текущего сеанса программы. Это значение пользователь может изменить самостоятельно, выбрав в меню Сервис пункт Параметры.

Параметры функции РабочаяДата. Параметр <Дата> позволяет установить новую рабочую дату, а параметр <Режим> задает режим изменения рабочей даты при наступлении полуночи и может принимать такие значения (соответствующие описанным выше настройкам параметров системы):

0- Не изменять в полночь;

1 - Предлагать изменять в полночь;

2- Изменять в полночь автоматич.

Изменение даты

Для изменения даты в 1С предусмотрена функция ДобавитьМесяц (AddMonth), которая добавляет к указанной дате заданное количество месяцев и имеет такой синтаксис:

ДобавитьМесяц(<Дата>, <КоличествоМесяцев>)

Определение начальных и конечных дат указанного типа

В 1С зачастую требуется определить дату начала или конца месяца (квартала, года, недели) относительно заданной даты. Для этих целей предусмотрен ряд специальных функций, имеющих один параметр <Дата> и возвращающих соответствующую дату:

Функция (рус.)	Функция (англ.)	Описание
НачМесяца	BegOfMonth	Возвращает дату начала месяца для указанной даты
КонМесяца	EndOfMonth	Возвращает дату конца месяца для заданной даты
НачКвартала	BegOfQuart	Возвращает дату начала квартала для указанной даты
Конквартала	EndOfQuart	Возвращает дату конца квартала для заданной даты
НачГода	BegOfYear	Возвращает дату начала года для указанной даты
КонГода	EndOfYear	Возвращает дату конца года для заданной даты
НачНедели	BegOfWeek	Возвращает дату начала недели для указанной даты
КонНедели	EndOfWeek	Возвращает дату конца недели для заданной даты

Определение года, месяца и числа

Чтобы определить для заданной даты год, месяц или число, следует воспользоваться соответственно такими функциями:

***ДатаГод (GetYear)** – возвращает числовое значение года для указанной даты;*

***ДатаМесяц (GetMonth)** – возвращает числовое значение месяца для заданной даты;*

***ДатаЧисло (GetDay)** - возвращает числовое значение дня для указанной даты;*

Номер дня и номер недели

Для определения номера недели и дня предусмотрены следующие функции, возвращающие числовые значения:

***НомерНеделиГода (GetWeekOfYear)** - определяет для заданной даты порядковый номер недели в году;*

***НомерДняГода (GetDayOfYear)** - возвращает для заданной даты порядковый номер дня в году;*

***НомерДняНедели (GetDayOfWeek)** - определяет для указанной даты порядковый номер дня недели (1 - понедельник, 2 - вторник,..., 7 - воскресенье).*

Работа со временем

Значения времени можно присваивать так же, как и значения дат, но разделителем здесь будет выступать не точка, а двоеточие, т.е. формат таков: ЧЧ:ММ:СС. Например:

Время = '13:50';

Время1 = '22:17:05';

Для работы со временем в 1С предусмотрена единственная функция ТекущееВремя (CurrentTime), обычно указываемая без параметров и возвращающая текущее время:

Время = ТекущееВремя();

У этой функции есть три необязательных параметра: <Час>, <Мин> и <Сек>. Если на их месте указать имена переменных, то после вызова функции ТекущееВремя в эти переменные будут занесены соответственно количество часов, минут и секунд для текущего времени.

Что касается обычных значений времени, задаваемых вручную, - для них нет специальных функций для извлечения отдельно часов, минут и секунд, но это легко можно сделать с помощью строковых функций Лев, Прав и Сред, которые были рассмотрены выше. ***Пример:***

```
Время = ТекущееВремя();
Часов = Лев(Время, 2);
Минут = Сред(Время, 4, 2);
Секунд = Прав(Время, 2);
Сейчас = "Текущее время: " + Время + Симв(13) +
         Симв(13) + "Часов: " + Часов + Симв(13) +
         "Минут: " + Минут + Симв(13) + "Секунд: " + Секунд;
Предупреждение(Сейчас);
```

Пользовательские процедуры и функции

Под процедурой или функцией понимается последовательность операций, которую нужно многократно выполнять в различных местах приложения. При этом требуемый блок команд записывается в коде только один раз, после чего к нему можно обращаться из любой части программы.

Существует несколько разновидностей описанных элементов программирования.

Системные процедуры и функции - определенные наборы команд, имеющиеся в языке 1С и предназначенные для вычисления тех или иных значений на основании исходных данных. Это, в частности, математические, строковые функции и т.д.

Системные предопределенные процедуры - вызываются в том случае, если произошло какое-либо событие (ввод нового объекта, изменение, удаление). Существует также ряд предопределенных процедур модуля документа (ОбработкаПроведения, 0бработкаУдаленияПроведения, АрхивироватьДокумент), которые могут вызываться как при интерактивных действиях пользователя, так и программным способом.

***Примечание**. Предопределенные процедуры имеют изначально установленными только название и набор параметров, а команды, реализующие те или иные действия, должен написать сам разработчик.*

Пользовательские процедуры и функции - наборы команд, создаваемые разработчиком для выполнения определенных задач и не зависящие от текущего состояния приложения или произошедших в тот или иной момент событий.

Между процедурами и функциями, в том числе пользовательскими, существует ряд различий.

Пользовательская процедура в модуле объявляется следующим образом:

Процедура <Имя>(<Параметры>)	***Procedure <Имя>(<Параметры>)***
<Операторы>	***<Операторы>***
КонецПроцедуры	***EndProcedure***

Описанная таким образом процедура может быть вызвана из любого места программного модуля (из любой другой процедуры или функции). Для этого нужно просто указать <Имя> со списком параметров через запятую в круглых скобках. Даже если передача параметров не происходит, скобки все равно необходимо указывать.

Ключевые слова Процедура и КонецПроцедуры не должны заканчиваться точкой с запятой, как все остальные операторы, так как эти ключевые слова являются операторными скобками.

Аналогичным образом объявляются пользовательские функции, но с обязательным оператором Возврат (обычно в конце тела функции), присваивающим функции значение, которое она возвратит в то место программы, откуда она была вызвана:

Функция <Имя>(<Параметры>)	***Function <Имя>(<Параметры>)***
<Операторы1>	***<Операторы1>***
Возврат <Значение>	***Return <Значение>***
<Операторы2>	***<Операторы2>***
КонецФункции	***EndFunction***

Когда в программе вызывается процедура или функция, в качестве параметров нужно указать имена тех переменных, значения которых требуется в данный момент обработать.

Например, создадим функцию, вычисляющую значение факториала для заданного числа N (N! = 1 x 2 x ... x N). Функция будет называться Факториал; число будет вводить пользователь в Диалоговом окне системной функции ВвестиЧисло; результат будет отображаться с помощью системной функции Предупреждение:

```
Функция вычисления факториала
Перем ВводЧисла;
Функция Факториал(N) Перем Счетчик, Итог;
Итог = 1;
Для Счетчик = 1 По N Цикл Итог = Итог * Счетчик
КонецЦикла;
Возврат Итог
КонецФункции;
Ввод = ВвестиЧисло(ВводЧисла, "Вычисление факториала", 3, 0);
Подсчет = Факториал(ВводЧисла);
Предупреждение("Результат: " + Подсчет);
```

Для того чтобы проверить работоспособность программы, проще всего разместить ее в глобальном модуле. В программе вначале объявляется переменная ВводЧисла, которая будет использована для хранения числа, введенного пользователем. После этого описывается функция Факториал, имеющая один параметр - N. Кроме того, в теле функции объявляются еще две локальные переменные - Счетчик (счетчик цикла) и Итог (наращиваемое значение, которое будет по окончании цикла содержать значение факториала числа N). Оператор Возврат присваивает функции значение переменной Итог в качестве возвращаемого в основную программу значения.

Далее идет основная программа. Вначале функцией ВвестиЧисло вызывается диалог, где пользователь задает число, для которого нужно найти факториал. Это значение заносится в переменную ВводЧисла.

В следующей команде в левой части оператора присваивания указывается переменная Подсчет, а в правой части - вызывается функция Факториал, в качестве параметра которой задается переменная ВводЧисла

Таким образом, значение переменной ВводЧисла подставляется в теле функции Факториал вместо переменной N, после чего производятся вычисления по описанному выше алгоритму.

Для пользовательских процедур и функций, как и для переменных, при помощи ключевого слова Экспорт можно устанавливать областью видимости все модули программы, однако это имеет смысл только в том случае, если процедура (функция) объявляется в глобальном модуле.

Отличия между процедурой и функцией.

1. Функция возвращает в программу значение, которое присваивается ей оператором Возврат: Возврат <Значение>;

2. Вызов функции обычно осуществляется путем указания ее имени и параметров в правой части какой-либо команды, а процедура вызывается при помощи отдельной команды: <ИмяПроцедуры>(< Параметры;»);

Вызываемая процедура может не иметь параметров. В этом случае после имени процедуры следует ставить пустые скобки.

Пользовательские процедуры обычно используются при необходимости выполнения одной и той же последовательности операций. Например, пусть из рассмотренной выше программы по определению факториала требуется неоднократно вызывать команды ввода пользователем числа, вызова функции Факториал и отображения результата. В этом случае данные команды можно оформить процедурой, которую назовем Главная.

Таким образом, в основной программе останется только одна команда (кроме объявления переменной ВводЧисла) - это вызов процедуры Главная:

```
Перем ВводЧисла;
Функция Факториал(N) Перем i, Итог; Итог = 1;
Для i = 1 По N Цикл Итог = Итог * i КонецЦикла; Возврат Итог КонецФункции
Процедура Главная() Перем Ввод, Подсчет;
Ввод = ВвестиЧисло(ВводЧисла,
"Вычисление факториала", 3, 0);
Подсчет = Факториал(ВводЧисла); Предупреждение("Результат: " + Подсчет); КонецПроцедуры
Главная();
```

Передача параметров по ссылке и по значению Параметры могут передаваться в процедуры или функции двумя способами.

Передача параметров по ссылке. Такие параметры еще называют параметрами-переменными. Данный режим в языке 1С используется по умолчанию. При этом вызываемая процедура (функция) изменяет значения тех переменных, которые ей передаются в качестве параметров.

Передача параметров по значению. Эти параметры носят также название параметров-значений. При вызове процедуры в таком режиме значения переменных, которые используются в качестве параметров, не изменяются. То есть в этом случае изменениям будут подвержены только «копии» переменных, передаваемых в процедуру. Перед такими переменными необходимо указывать ключевое слово Знач (Val).

Пример программы с процедурами, поясняющий возможные способы передачи параметров:

```
Перем x, y;
// Передача параметров по ссылке Процедура ПоСсылке(a, b)
a = a + 1;
b = b + 1 КонецПроцедуры
// Передача параметров по значению Процедура ПоЗначению(a, Знач b)
a = a + 1;
b = Ь + 1
КонецПроцедуры
x = 3;
y =5;
ПоСсылке(x, y);
// Результат: x = 4, y = 6
x = 3;
y = 5;
ПоЗначению(x, y);
// Результат: x = 4, y = 5
```

Вызов процедуры ПоСсылке приводит к тому, что значения обеих переменных x и y, используемых в качестве параметров, увеличиваются на 1.

После вызова процедуры ПоЗначению изменяется значение только переменной x, в противоположность которой переменная y остается без изменений. Использование при описании параметра b служебного слова Знач привело к тому, что в переменную y при выполнении процедуры ПоЗначению не будет передано никакого значения.

Специальные процедуры и функции

Функция ***СоздатьОбъект (CreateObject)*** *создает в программе объект указанного агрегатного типа и возвращает в заданную переменную ссылку на него. Без этой функции невозможно реализовать полноценную программную обработку в 1С содержимого справочников, документов и т.д.*

Обычно эта функция вызывается в правой части оператора присваивания, а в левой части указывается переменная, через которую в дальнейшем будет происходить обращение к данному объекту. Например:

Спр = СоздатьОбъект("Справочник"); СпрДолжн = СоздатьОбъект("Справочник.Должности"); ДокПрием = СоздатьОбъект("Документ.Прием"); ТабЗнач = СоздатьОбъект("ТаблицаЗначений"); СписокЗнач = СоздатьОбъект("СписокЗначений");

При создании объектов агрегатного типа можно указывать как неопределенный тип (Справочник, Документ и пр.), так и конкретный объект метаданных, с которым необходимо работать, - в последнем случае название требуемого объекта указывается через точку (Справочник.Должности, Документ.Прием и т.д.).

Объекты агрегатного типа, созданные при помощи функции СоздатьОбъект, изначально не готовы к работе, так как в них не выполнено позиционирование, т.е. нет текущего элемента.

Функция ***СтатусВозврата (RetumStatus)*** *используется только в предопределенных процедурах (ПриОткрытии, ПриЗакрытии, ПриЗаписи и т.д.) и позволяет установить значение так называемого статуса возврата той предопределенной процедуры, из которой она вызывается. Статус возврата, как правило, используется для отмены выполнения действий, реализуемых этой предопределенной процедурой, если не были выполнены какие-либо условия (эти условия задает сам разработчик). При этом в качестве параметра функции СтатусВозврата указывается значение 0.*

Например, для процедуры ПриЗаписи был написан следующий программный код, добавляющий в справочник Сотрудники нового сотрудника с заданной должностью:

```
Процедура ПриЗаписи()
СпрСотр = СоздатьОбъект("Справочник. Сотрудники");
СпрСотр.Новый();
СпрСотр. Наименование = ФИО;
Если Должность.Выбран() = 0 Тогда
Предупреждение("Должность не выбрана!")
СтатусВозврата (0); КонецЕсли;
СпрСотр.Должность = Должность; СпрСотр. Записать(); КонецПроцедуры
```

Использование текстовых объектов

В 1С предусмотрен специальный агрегатный тип данных Текст (Text) для работы с текстом. С его помощью обычно организовывается работа с текстовыми файлами (добавление строк в текстовые файлы, считывание строк из текстовых файлов). В частности, можно сформировать отчет в текстовом виде.

Чтобы получить доступ к методам агрегатного типа Текст, необходимо вначале объявить переменную-объект этого типа с помощью системной функции СоздатьОбъект (CreateObject). ***Например:***

ТекстОтчет = СоздатьОбъект("Текст");.

Операции с файлами

Поскольку объекты типа Текст обычно используются для взаимодействия с TXT-файлами, вначале рассмотрим методы, позволяющие наладить это взаимодействие.

Метод Открыть (Open) открывает указанный текстовый файл и заносит его содержимое в объект типа Текст. ***Например:***

ТекстОтчет = СоздатьОбъект("Текст"); ТекстОтчет.Открыть(“d:\Отчет.txt");

Заметим, что если указанный файл не существует, никакой ошибки не возникнет - просто объект (в данном случае ТекстОтчет) останется пустым, без текста.

Метод Записать (Write) позволяет записать текст из объекта типа Текст в заданный текстовый файл. Например:

ТекстОтчет = СоздатьОбъект("Текст");
... // Формирование текста в объекте ТекстОтчет
ТекстОтчет.Записать (“d:\Отчет.txt");

Если файл с указанным именем не существует, он будет создан. Если же такой файл существует, то при наличии в нем каких-либо данных они будут удалены.

Метод Очистить (Clear) позволяет удалить содержимое указанного текстового файла (указывается без параметров). Перед вызовом этого метода файл должен быть открыт.

Например: ТекстОтчет = СоздатьОбъект("Текст"); ТекстОтчет.Открыть(" d:\Отчет.txt "); ТекстОтчет. Очистить();

Отображение текста

Метод Показать (Show) показывает текст в специальном окне, в котором можно выполнять редактирование, и имеет следующий синтаксис: Показать(<Заголовок>, <ИмяФайла>)

При вызове этого метода содержимое объекта типа Текст отображается в специальном окне с заданным <Заголовком> (см. рис. 10.1). Параметр <ИмяФайла> определяет, в какой файл будет заноситься введенный пользователем текст. Если файл не указан, то текст не будет сохраняться ни в каком файле и пропадет по окончании работы программы.

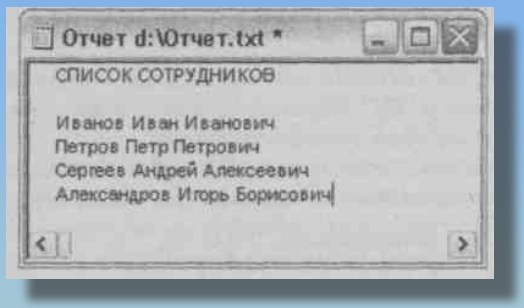


Рис. 10.1

Обычно с помощью метода Показать выполняется редактирование заданного текстового файла, предварительно открытого методом Открыть. ***Например:***

ТекстОтчет = СоздатьОбъект("Текст"); ТекстОтчет.Открыть(" d:\Отчет.txt "); ТекстОтчет.Показать ("Отчет", " d:\Отчет.txt ");

Метод ТолькоПросмотр (Readonly) предназначен для переключения между режимами просмотра и редактирования в окне редактирования, вызываемом методом Показать. Если в качестве параметра метода ТолькоПросмотр указано значение 1, то редактирование запрещено, а если 0, то редактирование разрешено.

Пример. В программе отображается вопрос, хочет ли пользователь редактировать текст. Если будет нажата кнопка Да, вызовется метод ТолькоПросмотр(0). Если же пользователь нажмет кнопку Нет, то вызовется метод ТолькоПросмотр(1).

Режимы просмотра и редактирования текста

```
ТекстОтчет = СоздатьОбъект("Текст");
ТекстОтчет. Открыть (" d:\Отчет.txt ");
ТекстВопроса = "Открыть текст для редактирования?";
Если Вопрос(ТекстВопроса, "Да+Нет") = "Да" Тогда
ТекстОтчет.ТолькоПросмотр(0)
Иначе
ТекстОтчет.ТолькоПросмотр(1)
КонецЕсли; ТекстОтчет.Показать();
```

Операции со строками

Рассмотрим методы, позволяющие выполнять различные действия со строками текста.

Метод КоличествоСтрок (LinesCnt) возвращает количество строк в тексте (указывается без параметров). Например, создадим объект ТекстОтчет и проверим, есть ли какой-либо текст в файле ***d:\Отчет.txt*** *, и если нет, выведем соответствующее сообщение.*

ТекстОтчет = СоздатьОбъект("Текст"); ТекстОтчет.Открыть ("d:\Отчет.txt "); Если ТекстОтчет. КоличествоСтрок() = 0 Тогда Предупреждение("Отчет не сформирован") КонецЕсли;

Метод ВставитьСтроку (InsertLine) позволяет вставить в текст строку с заданным номером. Например:

```
ТекстОтчет = СоздатьОбъект("Текст");
ТекстОтчет.Открыть(" d:\Отчет.txt ");
ТекстОтчет.ВставитьСтроку(1, "СПИСОК СОТРУДНИКОВ");
ТекстОтчет.ВставитьСтроку(2, "");
ТекстОтчет.ВставитьСтроку(3, "Иванов Иван Иванович");
ТекстОтчет.Записать(" d:\Отчет.txt");
```

Если в уже были какие-то строки, то новые строки добавятся без удаления имеющихся.

Метод ДобавитьСтроку (AddLine) служит для добавления строки в конец текста. Например:

```
ТекстОтчет = СоздатьОбъект("Текст");
ТекстОтчет.открыть(" d:\Отчет.txt"); "); Т
ТекстОтчет.ДобавитьСтроку("СПИСОК СОТРУДНИКОВ");
ТекстОтчет.ДобавитьСтроку ("");
ТекстОтчет.ДобавитьСтроку("Иванов Иван Иванович");
ТекстОтчет.Записать(" d:\Отчет.txt"); ");
```

Метод ПолучитьСтроку (GetLine) предназначен для извлечения из текста строки с заданным номером. В следующем примере

Метод ЗаменитьСтроку (ReplaceLine) служит для замены в тексте строки с заданным номером на другую строку.

Метод УдалитьСтроку (DeleteLine) используется для удаления строки с заданным номером из текста.

Добавление строк по шаблону

При работе с текстом имеется возможность добавлять в него не только обычные строковые значения, но и значения из заданных полей (например, из реквизитов справочников). В этом случае имена полей должны указываться в квадратных скобках.

Например:

ТекстОтчет.ДобавитьСтроку("[СпрСотр. Наименование]")

Добавляемое значение может быть смешанным, т.е. может включать в себя как поля, так и их пояснение:

Стр = "Должность: [СпрСотр.Должность]"; ТекстОтчет.ДобавитьСтроку(Стр);

Описанная выше возможность становится доступной только в том случае, если установлен специальный флаг добавления строк по шаблону. Этот флаг устанавливается одним из двух методов:

Шаблон (Template) или ФиксШаблон (FixTemplate), когда в качестве параметра задается значение 1. Например:

ТекстОтчет.Шаблон(1);

Если же нужно отменить добавление строк по шаблону, следует снова вызвать метод, но уже с параметром 0.

Метод ФиксШаблон отличается от метода Шаблон тем, что значения полей выводятся в фиксированном виде, а именно: если длина поля (то есть количество символов в этом поле, с учетом квадратных скобок) меньше, чем возвращаемое значение поля, то это значение обрезается. Если же длина поля превышает возвращаемое значение, то оно дополняется пробелами.

Если необходимо, чтобы значение поля не обрезалось, нужно дополнять его пробелами между квадратными скобками.

Строковые системные константы

В 1С существует несколько строковых системных констант, которые можно использовать в любом месте программы. Эти константы относятся к тексту и позволяют вставлять в него специальные символы.

Константа РазделительСтраниц (PageBreak) вставляет специальный символ перевода страницы.

Константа РазделительСтрок (LineBreak) добавляет специальный символ перевода строки.

Константа СимволТабуляции (TabSymbol) вставляет специальный символ табуляции.

Работа с файловой системой

Для работы с файлами и каталогами в 1С предусмотрен специальный агрегатный тип ФС. По умолчанию в системе всегда существует один объект этого типа, и называется он точно так же - ФС (FS). Помимо этого, в случае необходимости разработчик может создать в программе сколько угодно объектов этого типа функцией СоздатьОбъект.

Сразу необходимо заметить, что использование объектов агрегатного типа ФС не предусматривает действий по изменению содержимого файлов (в частности, текстовых) или извлечению из них информации - для этих целей предназначен другой тип, Текст, описание которого приведено выше, в разделе «Использование текстовых объектов».

Метод СвободноеМестоНаДиске (GetDiskFreeSpace), который возвращает объем свободного места на заданном диске в байтах.

Например, подсчитаем, сколько свободно Мбайт на диске С: (см. рис.11.1). Для этого вначале воспользуемся методом СвободноеМестоНаДиске, затем пересчитаем результат в Мбайты и, наконец, округлим полученное значение:

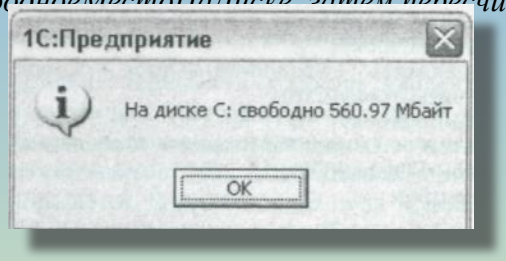


Рис. 11.1

СвобМесто = ФС.СвободноеМестоНаДиске("С:"); СвобМбайт = СвобМесто / (1024 * 1024); СвобМбайт = Окр (СвобМбайт, 2); Предупреждение("На диске С: свободно " + СвобМбайт + " Мбайт");

Действия с каталогами

Метод ВыбратьКаталог (SelectDirectory) отображает диалоговое окно, позволяющее пользователю выбрать требуемый для работы каталог, имеет следующий синтаксис:

ВыбратьКаталог(<ИмяКаталога>, <Заголовок>, <Задержка>)

В диалоговом окне с установленным <Заголовком> пользователь должен выбрать нужный каталог, имя которого будет сохранено в переменной <ИмяКаталога>. При этом возвращается полное имя каталога, т.е. с указанием пути.

Параметр <Задержка> устанавливает временной интервал (в секундах), в течение которого диалоговое окно выбора каталога будет видно на экране.

Метод СоздатьКаталог (Createdirectory) создает на диске новый каталог с заданным именем. Единственный параметр метода – название каталога в виде строкового значения. Если задано только название каталога, он создается в текущем рабочем каталоге системы. Если же разработчику требуется самому установить место создания каталога, нужно полное имя каталога, с указанием пути.

Например:

ФС.СоздатьКаталог("Отчеты"); // Каталог "Отчеты" создается в текущем рабочем каталоге

ФС.СоздатьКата лог(" D: \Отчеты"); // Каталог "Отчеты" создается в корневом каталоге диска D:

Метод УдалитьКаталог (RemoveDirectory) удаляет заданный каталог. Аналогично работе с методом СоздатьКаталог, если в методе УдалитьКаталог задается только имя каталога, он будет удаляться из текущего рабочего каталога.

Например:

ФС.УдалитьКаталог("D:\Отчеты");

В том случае, если указанный каталог не будет найден, в панели сообщений автоматически отобразится сообщение «Ошибка при выполнении команды».

Метод УстТекКаталог (GetCurrentDirectory) позволяет установить заданный каталог в качестве текущего. Если каталог не будет найден, то в панели сообщений отобразится сообщение об ошибке при выполнении команды.

Метод ТекКаталог (CurrentDirectory) возвращает текущий рабочий каталог (указывается без параметров).

Работа с файлами

Метод ВыбратьФайл (SelectFile), аналогично методу ВыбратьКаталог, отображает диалоговое окно для выбора или сохранения требуемого файла и имеет следующий синтаксис:

ВыбратьФайл(<ТипДиалога>, <ИмяФайла>, <ИмяКаталога>, <Заголовок>, <Фильтр>, <Расширение>, <Задержка>)

<ТипДиалога> - числовое значение, определяющее тип диало-. о - Открыть, 1 - Сохранить;

<ИмяФайла> - имя переменной, в которую будет возвращено имя выбранного файла;

<ИмяКаталога> - имя переменной, в которую будет возвращено имя каталога, где находится выбранный файл;

<Заголовок> - строковое значение, задающее текст заголовка для диалогового окна;

<Фильтр> - строковое значение, определяющее один или несколько фильтров, по которым будет производиться отбор из списка файлов. Каждый фильтр состоит из двух частей: описания фильтра и маски (или шаблона) для отбора файлов. Маска задает ограничения на имена и на расширения отбираемых файлов. Специальные символы, используемые в маске:

** - обозначает цепочку любых символов, любой длины;*

? - обозначает один любой символ.

Параметр <Расширение> позволяет задать расширение, которое будет задаваться по умолчанию при сохранении файла.

Параметр <Задержка> определяет, как долго диалог выбора или сохранения файла будет отображаться на экране.

Метод ВыбратьФайлКартинки (SelectPictFile) похож по своему синтаксису на метод ВыбратьФайл, однако здесь нет фильтров, а просто отображается список всех графических файлов с возможностью выбрать один из них.

Метод СуществуетФайл (ExistFile) позволяет выяснить, существует ли указанный файл, и возвращает 1 (если файл существует) или 0 (если файл не существует).

Метод КопироватьФайл (FileCopy) выполняет копирование указанного файла и имеет такой синтаксис:

КопироватьФайл(<Источник>, <Приемник>, <Перезапись>)

Таким образом, выполняется копирование файла с именем <Источник> в новый файл с именем <Приемник>. Если файл с именем <Приемник> уже существует, то параметр <Перезапись> определяет дельнейшие действия: если он равен 0, то выполняется перезапись, а если 1, то перезапись не производится.

Метод УдалитьФайл (DeleteFile) служит для удаления указанного файла. При этом, если файл не обнаружен, сообщение об ошибке не появится.

Метод ПереименоватьФайл (MoveFile) предназначен для переименования (или перемещения) указанного файла и имеет синтаксис, аналогичный методу КопироватьФайл:

ПереименоватьФайл(<Источник>, <Приемник>, <Перезапись>)

Таким образом, файл <Источник> будет переименован в файл <Приемник>. Что касается параметра <Перезапись>, если он равен 0, то перемещение с одного диска на другой будет запрещено, а при обнаружении файла с устанавливаемым именем операция переименования будет отменена. Если же параметр <Перезапись> равен 1, то описанные выше действия разрешены.

Метод АтрибутыФайла (GetFileAttr) возвращает атрибуты указанного файла и имеет такой синтаксис:

АтрибутыФайла(<Имя>, <Размер>, <Атрибуты>, <ВремяСоздания>, < Время ПоследнДоступа >, <ВремяПоследнЗаписи>, <ПолноеИмя>)

Первый параметр этого метода, <Имя>, задает имя требуемого файла, а в качестве остальных параметров указываются переменные, в которые возвращаются следующие значения:

<Размер> -размер файла в байтах (числовое значение);

<Атрибуты> - закодированные в виде строки значения атрибутов файла; каждый атрибут кодируется либо символом 0 (не установлен), либо 1 (установлен):

Первый символ - файл только для чтения;
Второй символ - скрытый файл;
Третий символ - системный файл;
Четвертый символ - каталог;
Пятый символ - архивный файл;
*Шестой символ - обычный файл; при этом никакой другой * атрибут не установлен;*
Седьмой символ - временный файл;
Восьмой символ - архивный файл;
Девятый символ - недоступный файл;

<ВремяСоздания> - дата и время создания файла (строковое значение);

<ВремяПоследнДоступа> - дата и время последнего доступа к файлу (строковое значение);

<ВремяПоследнЗаписи> - дата и время последнего сохранения файла (строковое значение);

<ПолноеИмя> - полное имя файла с указанием расширения (строковое значение).

Перебор файлов в текущем каталоге

Для поиска файлов по заданной маске (шаблону) предназначен метод НайтиПервыйФайл (FindFirstFile), который имеет следующий синтаксис:

НайтиПервыйФайл(< Маска >)

*В качестве маски указывается строковое значение, задающее фильтр для поиска требуемых файлов. Например, маска для текстовых файлов будет выглядеть так: *.txt*

Метод возвращает имя первого найденного файла (строковое значение), удовлетворяющего заданной маске.

Как правило, этот метод используется в паре с методом Най-тиСледующийФайл (FindNextFile) (указывается без параметров), который работает в цикле и отыскивает все последующие файлы, удовлетворяющие маске, заданной методом НайтиПервыйФайл.

*Эти методы выполняют поиск в текущем каталоге и имеют одну особенность (при переборе всех файлов по маске *.*), первоисточником которой является команда Dir операционной системы MS-DOS. Особенность заключается в следующем: если текущий файл находится не в корневом каталоге, то первым найденным Результатом будет значение "." (точка) - ссылка на текущий каталог, следующим - ".." (две точки)- ссылка на каталог уровнем выше, и только после этого придет очередь для первого по счету файла в текущем каталоге.*

*Вторая особенность поиска по маске *.* состоит в том, что текущем каталоге будут перебраны не только все файлы, но и все подкаталоги (проверка в подкаталогах не выполняется).*

Отладка программных модулей. Панель инструментов Отладчик

При разработке программных модулей зачастую требуется отслеживать поведение программы в тот или иной момент времени. Для этого в 1С предусмотрен специальный режим - Отладчик в котором отслеживается возникновение ошибок при работе в пользовательском режиме, состояние различных переменных в определенном месте программы и т.д. То есть с помощью режима Отладчик разработчик убеждается в том, что его программа работает именно так, как и планировалось.

Вызов режима Отладчик осуществляется из Конфигуратора нажатием на панели инструментов Конфигурация кнопки «Отладчик» или клавиши F12. Кроме того, зачастую требуется запустить отладчик для конкретного модуля - в этом случае нужно открыть требуемый модуль, после чего нажать кнопку «Открыть модуль в отладчике».

Панель инструментов Отладчик

В режиме Отладчик разработчику доступна одноименная панель инструментов (см. рис. 12.1), в которой имеется широкий набор средств для отладки программы.

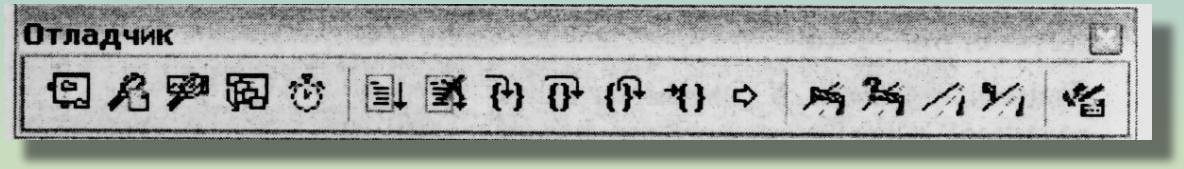


Рис. 12.1

В таблице 12.1 приведено краткое описание каждого из имеющихся здесь инструментов.

Кнопка	Название	Описание
	Список модулей	Отображает в иерархическом виде список модулей конфигурации
	Вычислить выражение	Позволяет в процессе отладки вычислить любое выражение (Shift+F9). Обычно в выражении используются переменные из отлаживаемого модуля
	Открыть табло	Открывает специальное окно Табло, в котором отображаются результаты вычисленных выражений
	Стек вызовов	Открывает специальное окно Стек вызовов. В котором отображается название от
	Замер производительности	Активизирует режим замера производительности, в котором для каждой строки модуля запоминается длительность ее выполнения, а также частота, с которой строка выполнялась в процессе выполнения модуля
	Продолжить	Возобновляет выполнение программы после прерывания (F5)
	Прекратить	Прекращает процесс отладки модуля (Alt+F5)
	Шагнуть в	Выполняется один оператор (очередной), причем вызываемые процедуры и функции в этом случае будут выполняться пошагово (F8)
	Шагнуть через	Выполняется очередной оператор, причем выполнение любой процедуры или функции будет осуществляться за один шаг (F10)
	Шагнуть из	Завершает выполняемую процедуру и переходит на оператор модуля, следующий за оператором вызова этой процедуры (Shift+F7)
	Идти до курсора	Выполняет все операторы модуля до той строки, в которой расположен курсор (F7)

Кнопка	Название	Описание
	*Текущая строка*	*Открывает окно выполняемого в данный момент модуля и показывает текущую строку*
	*Точка останова*	*Устанавливает в текущей строке модуля точку останова (F9)*
	*Точка останова с условием*	*Устанавливает в текущей строке модуля точку останова и предлагает задать условие, при выполнении которого программа будет прерываться в этой строке*
	*Убрать все точки останова*	*Удаляет из модуля все заданные точки останова*
	*Включить/ Отключить точку останова*	*Включает или отключает точку останова в текущей строке модуля (Ctrl+Shift+F9)*
	*1 С: Предприятие*	*Запускает конфигурацию в пользовательском режиме, тем самым начиная процесс отладки*

Отладка при помощи точек останова

Выполнение программы в 1С можно приостановить в любом месте с использованием точек останова. Так называются особые метки, помещенные в программе на определенных операторах, на которых работа программы должна быть остановлена. Эти точки еще называют «точками прерываний» от их английского названия «breakpoint», или «контрольными точками».

Поместить или удалить уже имеющуюся контрольную точку в коде программы можно одним из следующих способов:

- ✔ *расположить курсор в строке модуля, где нужно остановить * его выполнение, и нажать клавишу F9;*
- ✔ *нажать кнопку ^ Точка останова, которая находится на панели инструментов Отладка;*
- ✔ *выбрать пункт меню Отладка \ Точка останова; *
- ✔ *щелкнуть мышью на поле индикаторов в левой части окна редактирования.*

Точка останова изображается в виде «шлагбаума» слева от строки программы, на которой она установлена, а текущая строка в процессе отладки обозначается символом гоночной машины.

Убрать все точки прерывания можно с помощью пункта меню Отладка | Убрать все точки останова или кнопкой на панели инструментов Отладчик.

Большинство ошибок в программах являются не синтаксическими, а вызваны неверной логикой программы или определенными неверными предпосылками при разработке. Ошибки такого Рода обнаруживаются зачастую через продолжительное время, и Для их поиска приходится применять разнообразные приемы. Контрольные точки позволяют сузить область программы, где предполагается наличие ошибки. Определив участок программного кода с ошибкой, можно изучить его подробнее, просматривая значения переменных.

Наблюдение за значениями переменных

Отслеживаемые переменные и их текущие значения отображаются в специальном окне Табло, которое открывается с помощью кнопки ***Открыть табло*** *панели инструментов Отладчик. Обычно табло представлено в виде встроенной панели.*

Чтобы отобразить табло в виде окна, нужно щелкнуть внутри него правой кнопкой мыши и выбрать в контекстном меню пункт Перемещать в главном окне. Тогда табло примет следующий вид (см. рис 14.1)

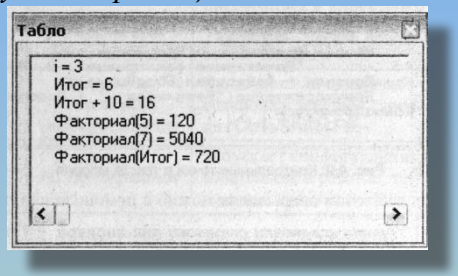


Рис. 14.1

Для того чтобы добавить в это окно одну или несколько переменных, значения которых требуется отслеживать, можно воспользоваться кнопкой ***Вычислить выражение*** *панели инструментов Отладка. Но более простой и удобный способ - непосредственный ввод имен переменных и выражений в окно Табло. Когда введено очередное выражение (или имя переменной), необходимо нажать клавишу Enter, после чего рядом через знак равенства (=) будет отображено текущее значение переменной (или выражения).*

На рис. 14.1 можно увидеть, какие возможности по просмотру текущих значений переменных и выражений имеет разработчик. В частности, можно узнать значение функции для указанного параметра, причем параметром может быть как конкретное значение (в данном случае - числа 5 и 7), так и используемая в программе переменная (здесь - переменная Итог).

Если разработчик задаст некорректное выражение (например, если указанной переменной не существует или выполняется некорректная операция типа деления на 0), то рядом с этой переменной (или с выражением) будет отображаться сообщение «Ошибка в выражении!» (см. рис.. 14.2).

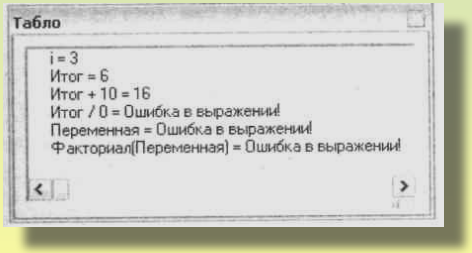


Рис. 14.2

Пошаговое выполнение программы

После того как участок программного кода, где находится ошибка, локализован, часто приходится прибегать к поочередному выполнению операторов, чтобы обнаружить, на каком именно шаге появляется нежелательный результат.

Существует два режима пошагового выполнения: Шагнуть в и Шагнуть через. Первый из них служит для прохода всего модуля строка за строкой. Если в одной из строк происходит вызов процедуры, то пошаговое выполнение распространяется и на код этой процедуры: каждый ее оператор выполняется пошагово. Для выполнения очередного шага нужно нажать клавишу F8 или выбрать пункт меню Отладка/ Шагнуть в.

Для того чтобы выполнить программу в режиме Шагнуть в, полезно установить точку останова сразу в том месте, откуда требуется начать пошаговое выполнение, и запустить программу нажатием клавиши ***F11*** *или кнопки 1С: Предприятие. После остановки программы на точке останова можно перейти к пошаговому выполнению.*

На рис. 15.1 приведено окно с фрагментом программного кода, где точка останова была установлена в теле процедуры Главная в строке где вызывается функция Факториал. Это позволило зайти в функцию Факториал для отладки в пошаговом режиме. Текущий оператор отмечен символом гоночной машины в области индикаторов.

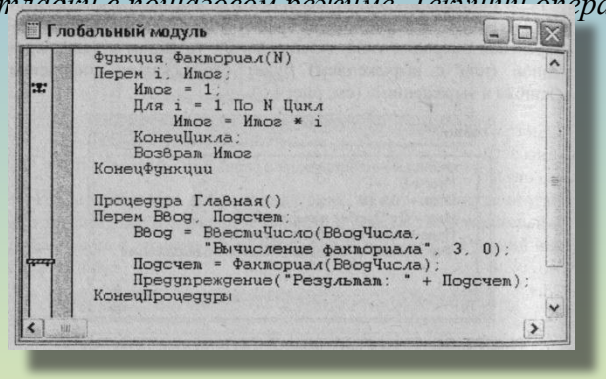


Рис. 15.1 Пошаговое выполнение программы с заходом в функцию

Если заход в процедуру (или функцию) осуществлен, но дальнейшее ее выполнение по одному оператору не представляет интереса, то можно немедленно пройти ее до конца и продолжить работу в пошаговом режиме в основной программе. Для этого служат комбинация клавиш ***Shift+F7*** *и команда меню Отладка/ Шагнуть из.*

Режим Шагнуть через аналогичен режиму Шагнуть из за исключением того, что если текущий оператор содержит вызов процедуры (или функции), то заход в нее не выполняется. Таким образом, вся процедура (или функция) рассматривается как одно действие.

Выполнение произвольных фрагментов кода

Контрольные точки, помещенные в тексте программы, действуют до тех пор, пока они не будут сняты. Когда в программе их слишком много, процесс отладки становится медленным и утомительным. Вместе с тем, часто возникает необходимость выполнить фрагмент кода только один раз и остановиться в определенном месте. Так как устанавливать точку прерывания для однократного выполнения фрагмента нерационально, можно воспользоваться командой меню Отладка/ Идти до курсора. Для этого следует установить курсор в той строке, где нужно остановиться, и выбрать указанную команду из меню (или нажать комбинацию клавиш F7).

Средства поиска и замены текста

Поиск и замену текста приходится очень часто применять при работе с любым документом в текстовом редакторе. В то же время эта функция может оказаться полезным средством при отладке программы. Она позволяет достаточно быстро внести стереотипные изменения в большой фрагмент программы (например, заменить СпрСотр на СпрСотрудник).

Для активизации поиска нужно выбрать команду меню Действия/ Поиск/ Искать или воспользоваться сочетанием клавиш ***Ctrl+F3.*** *В появившемся диалоговом окне Поиск (см. рис. 17.1) следует ввести текст для поиска в поле Искать. Этому полю соответствует раскрывающийся список, где хранятся последние варианты поиска. Для осуществления поиска очередного вхождения строки в тексте нужно нажать кнопку Искать.*

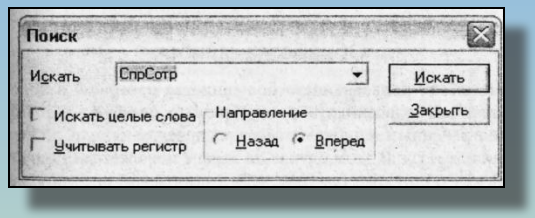


Рис.17.1*. Окно поиска текста*

В окне поиска имеется ряд параметров:

- переключатели Направление - служат для выбора направления поиска: Назад - только вверх по тексту от текущей строки, Вперед - только вниз по тексту от текущей строки;

- флажок Искать целые слова устанавливает режим поиска введенного текста только в качестве отдельного слова, а все вхождения этого текста в состав других слов будут пропущены;

- флажок Учитывать регистр позволяет различать при поиске заглавные и строчные буквы.

Окно замены текста аналогично по своим возможностям окну поиска, за исключением того, что в нем следует ввести текст, на который нужно заменять найденный. Для поиска очередного вхождения в тексте нужно нажимать кнопку Искать а для его замены - кнопку Заменить. Чтобы заменить все вхождения текста в программе, не останавливаясь по очереди на каждом из них, можно воспользоваться кнопкой Заменить все.

В разделе Изменить в разработчик может задать область для замены - либо в выделенном блоке (переключатель Блоке становится доступен, если в модуле выделен блок), либо во всем модуле (переключатель Файле).

Обработка ошибок

При работе подавляющего большинства программ в ходе выполнения могут возникать разнообразные ошибки. К наиболее распространенным ошибкам относятся деление на нуль, неверный тип данных и т.д. В этом случае на экране появляется стандартное сообщение об ошибке (см. рис. 18.1) и программа аварийно завершает работу.

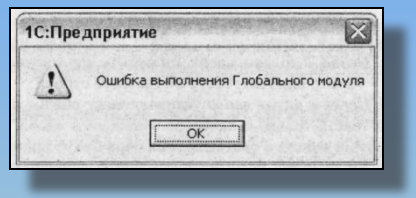


Рис. 18.1

При возникновении ошибки в процессе выполнения программы в окне сообщений отображается строка, в которой возникла ошибка, ее порядковый номер в модуле, а также указывается тип этой ошибки.

Несмотря на то, что это чрезвычайно сложно, при разработке программы следует пытаться предугадать все возможные ситуации, в которых могут произойти ошибки, и предусмотреть для их обработки определенный программный код.

Если есть подозрение, что в определенном блоке операторов может возникнуть ошибка, необходимо поместить этот блок в инструкцию Попытка...Исключение, имеющую следующий синтаксис:

Попытка Try
//Операторы1 //Операторы1
Исключение Except
//Операторы2 //Операторы2
КонецПопытки EndTry

После ключевого слова Попытка располагается блок операторов Операторы1, в которых может возникнуть ошибка. Если эти операторы выполняются без ошибок, то управление передается первому оператору, следующему за конструкцией Попытка...Исключение. Если же при выполнении блока Операторы1 возникла ошибка, то выполнение этого блока прерывается и управление передается на блок операторов Операторы2, размещенный после ключевого слова Исключение. Здесь происходит обработка ошибки (обычно это выдача предупреждающих сообщений для пользователя, а также предложение повторить ввод, если ошибка возникла из-за некорректного значения, введенного пользователем). На этом выполнение операторов конструкции Попытка...Исключение заканчивается.