**Введение в язык Cache ObjectScript. Строки с разделителями. Списки.**

В незапамятные, по меркам области информационных технологий, времена, в 1967 г. был создан язык MUMPS. Эта аббревиатура полного названия "Massachusetts General Hospital's Utility Multiprogramming System", была неудачна уже по тому, что слово mumps означает болезнь, именуемую в просторечии свинкой. В 1977 г. язык был стандартизирован ANSI/ISO сразу после Фортрана и Кобола и в середине 90-х переименован в язык M. Катастрофически не везло языку с именами. Не согласны — попробуйте найти сведения о нём, задав поисковику ключевое слово "M". Современную версию языка М, называемую вполне приличным именем Cache ObjectScript (сокращённо, COS), поддерживает фирма InterSystems International.

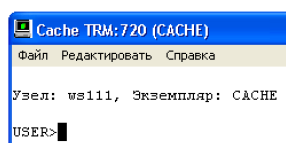
Система управления базами данных Cache (это слово в переводе с французского означает то, что приносит известность или престиж) и правда обладает многими достоинствами. Почему для изложения материала выбрана именно Cache? А потому, что это единственная СУБД, в которой можно не только поработать с тремя моделями данных — иерархической, реляционной и объектной — и изучить их отображения, но и самим создавать другие модели данных. Мы освоим основные изобразительные средства языка и, особенно, работу с древесными структурами. В первых разделах будем изучать Cache ObjectScript так, как изучают какой-нибудь QBasic в школе. Это позволит приобрести необходимые начальные навыки и не отвлекаться на мелочи при работе с деревьями.

Для нас важно, что ObjectScript — это персистентный язык. Это означает, что он работает со структурами данных, хранящимися на диске. И эти структуры представляются деревьями, может быть связанными между собой. Их мы будем изучать подробно. Позже вы узнаете, что и программа в ObjectScript—это дерево специального вида.

Не считайте, что приобретённые в процессе скитаний по деревьям знания и навыки, нужны только в Cache. В любой современной СУБД используют XML-модели, имеющие древесную структуру. Так что вы всегда и везде будете на должном уровне.

**Введение в Cache ObjectScript (переменные, выражения, основные команды)**

Первое время нас будет интересовать единственный пункт меню — "Терминал". Позже мы будем использовать студию и часть функций портала управления системой. Итак, выбираем пункт меню "Терминал". Окно терминала выглядит примерно так, как показано на рисунке 3.11.



**Рис. 3.11.** Начало работы с терминалом

Подсказка USER> приглашает начать работу в пространстве имён (или в базе) USER. Всегда существуют две обязательных области имен %SYS и %CACHELIB и сколько угодно необязательных, в том числе USER. Пространство имён — это множество уникальных имён, обычно как-то связанных по смыслу. Одно имя может повторяться только в разных пространствах имён.

Область имён всегда можно сменить командой ZN:

USER>ZN "SAMPLES"

SAMPLES>

Кстати, имена команд не чувствительны к регистру.

**Основные команды COS**

В одной строке можно размещать несколько команд. Определение командной строки для работы в терминале:

командная\_строка ::= команда[\sqcupкоманда...] \hookleftarrow

Командная строка состоит из одной или нескольких команд. Каждая команда состоит из командного слова, следующего за ним одного пробела-разделителя и аргументов, разделенных запятыми. Команды в одной строке разделяются не менее чем одним пробелом. За командой без аргументов должно следовать не менее двух пробелов. Количество команд в строке ограничивается лишь заданием максимальной длины строки.

Иногда можно задать несколько аргументов команды, разделенных запятыми. Если хотите, можно после запятых помещать один или несколько пробелов. Можно пробелов после запятых не делать вообще.

В число аргументов команд ввода (READ) и вывода (WRITE) можно включать символы форматирования:

1. переход к новой строке задается символом восклицательный знак (!);
2. табуляция отмечается знаком вопроса (?), за которым следует номер позиции в строке, с которой продолжается вывод.

Команда WRITE обеспечивает вывод информации в терминале. Допускается использование как полного имени команды WRITE, так и сокращенного до одной буквы W.

Синтаксис команды WRITE:

W[RITE][:постусловие]\sqcup[форматирование | аргумент\_WRITE], \dots,

где форматирование — один из символов !, ?, \#;

за символом ? следует число;

аргумент\_ WRITE — переменная, константа или выражение.

Постусловия — это условия, при выполнении которых команда будет исполнена, иначе команда пропускается. Необходимость понятия аргумент станет понятна при изучении косвенности.

Заметим, что все имена команд не чувствительны к регистру.

В конструкции "?число", аргумент не должен превышать 32767. Да и зачем нам такое число пробелов? Числа меньше нуля или 0 не сдвигают маркер. Дробная часть числа просто отбрасывается и аргументом считается оставшаяся целая часть. Иначе говоря, минимальный шаг — это знакоместо.

WRITE без аргументов выводит на терминал имена и значения всех существующих на данный момент в памяти переменных, называемых локалами. Выполните командную строку W 0, ?20, "a", ?40, "b", !, 0, попытавшись предсказать результат до выполнения команды на машине.

Применение перехода к новой странице — знак диез (#) — в общем случае имеет смысл, только если печатается твердая копия (в COS можно перенаправлять вывод на любое устройство). Команда W # очищает экран.

Присваивание выполняется командой SET. Её формат:

S[ET][:постусловие]\sqcup [присваивание][,присваивание \dots]

Присваивания являются аргументами команды.

Присвоим переменной x значение 1 командой SET\ x=1\rfloor. Проверить, была ли создана переменная x, можно, набрав команду WRITE\ x\rfloor. Результатом выполнения этой команды является вывод на экран значения переменной x. Действительно оно равно 1.

USER>SET x=1

USER>WRITE x

1

USER>

Пример 3.1. Присваивание и выдача значения переменой

Вы, конечно, заметили, что редактор терминала — строчный. Например, набрав WRITTE x, мы можем клавишей — подвести знак маркера к лишней букве "T" и удалить её клавишей Delete.

Выполнявшиеся ранее команды запоминаются. Последняя выполненная команда вызывается клавишей \uparrow. Каждое нажатие на неё вызывает предыдущую команду, пока все сохранённые записи не исчерпаются. Клавиша \downarrow используется для обратного прохода по этим же командам.

Вспомним, что командные слова не чувствительны к регистру, так что в дальнейшем позволим себе вольности и будем писать их и заглавными и строчными буквами.

Командные слова сокращаются, как правило, до одной буквы. Так, вместо строки SET x=1 можно написать S x=1. Если же нужно создать несколько переменных, то это можно сделать следующим образом:

|USER>S z=1, Ы="", h=1.2

Для проверки значений всех существующих на текущий момент переменных можно распечатать их значения при помощи команды WRITE без параметров.

*Замечание*. Cache одинаково хорошо работает с обеими половинами таблицы ASCII. Так что кириллица используется без ограничений.

В дальнейшем изложении знак \hookleftarrow проставляться не будет, так как его применение достаточно очевидно — обозначение конца командной строки и конца ввода.

Синтаксис команды READ ввода данных с терминала:

R[EAD][:постусловие]\sqcup[[подсказка,]аргумент] 

В качестве аргументов используются имена переменных.

Форматирование включается в подсказку.

При исполнении команды система ждёт ввода данных. Завершается ввод клавишей Enter. Система позволяет ограничить время ввода. Если набрать  R x:3 и ничего не вводить, то через 3 секунды ожидание ввода прекратится.

USER>R x:3

USER>

Пример 3.2. Установка времени ожидания

Точно также можно ограничить количество вводимых символов. Так, команда R x#4 позволяет ввести не более четырёх символов. Можно сочетать оба ограничения # и :. Например, R x#4:8.

Команда READ позволяет вводить значения с любого внешнего устройства, но мы этот вариант рассматривать не будем.

USER>R ?10,"Введите ваше имя: ",nn

Введите ваше имя: Николай

USER>R ?5,"Введите ваше имя: ",nn,!,?5,"Фамилию: ",sn

Введите ваше имя: Николай

Фамилию: Бессарабов

Пример 3.3. Команда READ

Каждое введенное значение завершается нажатием на клавишу Enter. Проверьте командой WRITE введенные значения фамилии и имени.

Стирать созданные переменные можно командой KILL, сокращённо K. При этом KILL без параметров удаляет все локальные переменные. В варианте "KILL список\_имён" удаляются все указанные переменные, а "KILL (список\_имён)" удаляет все переменные кроме указанных.

USER>K s a=1,b=2,c=3,d=4,bc=7 7 W

a=1

b=2

bc=77

c=3

d=4

USER>K a,b W

bc=77

c=3

d=4

USER>K (c,d) W

c=3 d=4 USER>

Пример 3.4. Команда Kill

В первой командной строке командой K мы удалили все имевшиеся локальные переменные и создали локалы a, b, c, d, bc. Обратите внимание, что здесь после K стоит ровно два пробела. Во второй командной строке "K a, b W" удаляются переменные a и b и выводятся все оставшиеся локалы. В третьей строке "K (c, d) W" удаляются все переменные, кроме c и d, то есть удаляется bc.

**Выражения**

В командах SET и WRITE можно вычислять сложные выражения. При этом следует помнить, что в Cache для ускорения разбора принято расшифровывать выражения слева направо, не учитывая старшинство операций. Поэтому 2 + 3 \* 8 равно 40, а не 26, как учили в школе. Пользуйтесь скобками для задания порядка действий.

USER>W 2+3\*8

40

USER>W "12qwerty"+4

16

USER>W "qwerty12"+4

4

USER>

Пример 3.5. Чтение слева направо

Все переменные не типизированы. По сути, хранится единственный тип данных — строка переменной длины. Последовательность символов в зависимости от контекста расшифровывается слева направо, пока это возможно. Поэтому "12qwerty" в арифметическом выражении равно 12, а "qwerty12" нулю.

Выражения могут быть трех типов:

* Арифметические выражения. Содержат арифметические операторы, интерпретируют операнды как числа и возвращают числовой результат.
* Строковые выражения. Содержат строковые операторы, интерпретируют операнды как строки и возвращают результат в виде строки.
* Логические выражения. Содержат логические операторы и отношения, интерпретируют операнды как логические значения и возвращают булев результат, в котором истина передаётся как TRUE или 1, а ложь как FALSE или 0. В интерпретациях логических выражений "не 0" - это TRUE, а 0 — это FALSE. В таблице 3.1 перечислены только те операторы ObjectScript, смысл которых не вполне очевиден.

|  |  |
| --- | --- |
| Таблица 3.1. Операторы ObjectScript, толкование которых нуждается в уточнении | |
| Оператор | Выполняемое действие |
| \ | Целочисленное деление |
| \*\* | Степень |
| # | Остаток от деления по модулю |
| \_ | Конкатенация (соединение строк) |
| ' | Логическое отрицание (NOT) |
| = | Присваивание или сравнение на равенство |
| ‘= | Сравнение на неравенство |
| [ | Содержит |
| ] | Следует за |
| ]] | Сортируется за |
| &&,& | Логическое AND (&& это "короткое" AND) |
| ||, ! | Логическое OR (|| это "короткое" OR) |
| @ | Косвенность |
| ? | Сравнение с шаблоном |

Допускаются комбинации '> не больше чем (меньше или равно) и '< не меньше чем.

**Логические выражения**

Основу логических выражений образуют операторы сравнения (=, >, <, '=, [, ], ?). Заметим, что обозначение для неравенства <> не употребляется. А вот условия <= и >= можно использовать наряду с '< и '>.

Например, R x IF x <=2 W x,"<=2"

Обратите внимание, что в качестве одиночной кавычки выбирается та, которая расположена на клавише с буквой "э".

Операторы сравнения и логические константы связываются логическими операторами AND (обозначается & или &&), OR (обозначается | или ||), NOT (обозначается ').

Предскажите и проверьте результат выполнения трёх командных строк с постусловием:

W:"1ab" "Условие истинности" W:-1 "Условие истинности" W:0 "Условие истинности"

Ответ: Первое постусловие "^^'представляет константу, интерпретируемую как логическое значение 1. Второе условие в виде константы —1 также интерпретируется как "не 0", то есть 1. И только третье соответствует ложному значению.

Краткие варианты && логического оператора "И" и || оператора "ИЛИ" отличаются от обычных & и | тем, что выполнение цепочки связанных условий прекращается, как только результат определится. Для "И" достаточно получить первый ложный результат, а для "ИЛИ" первый истинный. Условия в IF, записанные через запятую, также выполняются по краткому варианту. Поэтому для повышения скорости, необходимо первыми помещать те условия, у которых больше шансов получить ложное значение для "И" и истинное значение для "ИЛИ".

Логические и арифметические выражения могут сочетаться. Логическое выражение может использоваться для выбора арифметического значения, например, (11*(x>7))+(27*(x<=7)).

Для тренировки объясните работу следующих командных строк:

 S\ x=3\ W\ 11*(x>7)+27*(x<=7) 

и

 S\ x=8\ W\ 11*(x>7)+27*(x<=7) 

**Переменные. Зарезервированные слова**

Мы уже обратили внимание на то, что Cache очень лояльна к именам из второй половины ASCII таблицы. В других СУБД с ними могут возникать большие проблемы. Скажем, такие имена допускаются при создании объектов, которые затем работают с ошибками. Имена переменных должны начинаться либо с буквы, либо со знака процента (%), за ними могут следовать буквы или цифры. Системные переменные могут начинаться с символа $.

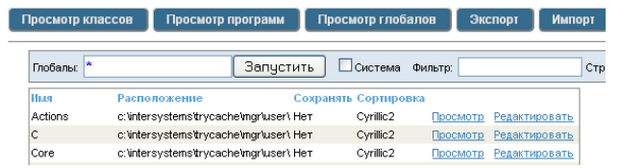
Длина имени не ограничена, но в идентификации имён участвуют только первые 31 символов. Пользовательские имена чувствительны к регистру, а предопределённые имена команд нет.

В COS нет проверки на совпадение пользовательских имён с зарезервированными именами. Можно создать переменную с именем set, например, SET set=11, но так поступать не рекомендуется. Вы можете запутать и себя и других разработчиков.

**Глобалы**

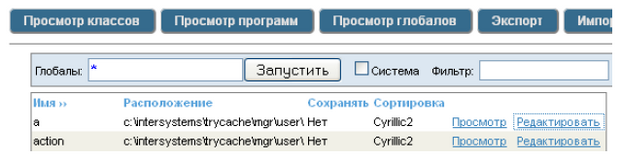
Язык COS способен создавать переменные, значения которых сохраняются на диске. В противовес временным переменным, именуемым локалами, хранимые переменные называются глобалами. Их имена обязательно начинаются со знака "^". Как остроумно сказано в одной книге, это галочка, которая в простонародье именуется циркумплекс. Действия над глобалами осуществляются так же, как и с обычными переменными — локалами. Но WRITE без параметра их не обнаруживает.

Даже если вы ещё не создавали глобалов, то в выбранном вами пространстве имён, например USER, они уже существуют. Для проверки войдем в портал управления системой и в разделе "Управление данными (System Explorer)" выберем "Глобалы", а затем пространство USER (рисунок 3.12).



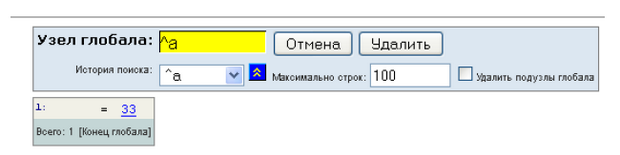
**Рис. 3.12.** Глобалы, созданные при инсталляции

Командой S ^а=33 в терминале создаём глобал и проверяем его наличие нажав на "Просмотр глобалов"(рисунок 3.13).



**Рис. 3.13.** Глобал ^a создан

Выбираем "Просмотр" или "Редактировать". Теперь можно просмотреть глобал и удалить его, а в глобале с древесной структурой удалить его весь или же отдельные узлы ([рисунок 3.14](https://www.intuit.ru/studies/courses/3687/929/lecture/19320?page=3#image.3.14)).



**Рис. 3.14.** Редактирование глобала

Глобалы пространства имён XX хранятся в папке с именем XX в файле с одним и тем же именем Cache.dat. На моей машине созданный глобал ^а находится в файле Cache.dat в папке C:\InterSystems\Cache\mgr\user. У вас вместо имени Cache может использоваться TryCache.

Заметим, что процессом сохранения созданных или изменённых глобалов пользователь управлять не может.

**Ошибки**

Предполагалось, что, выполняя предыдущие примеры, вы не делали ошибок. Вообще говорят, что люди делятся на умных, которые учатся на чужих ошибках и других, которые учатся на своих. Так вы этому не верьте. По крайней мере, в программировании это не так. Хорошо бы, конечно, никогда не делать ошибок. Но в реальности остаётся стараться не делать их слишком много, особо избегая непоправимых ошибок, и, самое главное, быстро исправлять сделанные ошибки. Научиться этому можно, только на своих ошибках.

Итак, вы решили определить переменную с неправильным именем 2x задав команду S 2x=1. Получаем сообщение об ошибке (пример 3.6, первая командная строка). Ошибка (<SYNTAX>) означает синтаксически неправильно сформированную строку языка Cache ObjectScript.

Сделаем ещё одну ошибку, которая позволит выяснить, как исполняется командная строка (пример 3.6, вторая командная строка). Ошибка <UNDEFINED> сигнализирует о том, что некоторая переменная не имеет значения. Имя переменной пишется после знака \*.

USER>S 2x=1

S 2x=1

^

<SYNTAX>

USER>K S x=1,y=2,z=a,v=3

K S x=1,y=2,z=a,v=3 л

^

<UNDEFINED> \*a

USER>W

x=1

У=2 USER>

Пример 3.6. Ошибки

Вторая ошибка произошла из-за того, что переменной z мы пытались присвоить значение неопределенной переменной а. Результат выполненной позднее в третье строке команды W показывает, что командная строка читается слева направо и те её части, которые могут быть выполнены до появления ошибки (S x=1,y=2) выполняются. Появление ошибки прекращает исполнение командной строки и присваивание S v=3 не производится, хотя эта часть команды ошибки не содержит.

**Строки**

Мы уже встречались со строковыми константами, представляющими последовательность символов, заключенных в двойные кавычки. Набор используемых символов зависит от сделанного при установке Cache выбора кодировки (Unicode или UTF8). Длина строки не превышает 32 К символов.

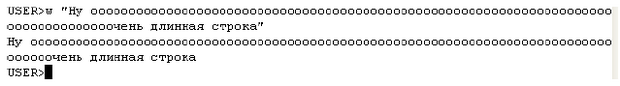
Поскольку двойная кавычка ограничивает строковую константу возникает вопрос, что делать, если символ " встречается внутри текста? В этом случае её просто удваивают:

USER>W "a""b"

a"b

Длина пустой строки — ноль. В команде READ вводимой переменной присваивается пустая строка каждый раз, когда ввод состоит из единственного нажатия клавиши Enter.

В стандартном ASCII-терминале длина строки 80 символов. При выводе длинная строка разрезается. При вводе необходимо продолжать набор символов, не обращая внимание на самопроизвольный переход маркера на следующую строку ([рисунок 3.17](https://www.intuit.ru/studies/courses/3687/929/lecture/19320?page=5#image.3.17)).



**Рис. 3.17.** Очень длинная строка

При выборе UTF8 допускается использование символов кириллицы и псевдографики, занимающих вторую половину кодовой таблицы ASCI ОТ 128 до 255).

Над строками определены пять бинарных операций:

* равенство (=),
* соединение строк (конкатенация), обозначаемое знаком подчеркивания (\_),
* проверка соответствия шаблону, обозначаемая знаком вопроса (?),
* операция "содержит", обозначаемая [, и определяющая, содержится ли второй операнд в первом,
* операция "следует за", обозначаемая ], и определяющая, следует ли первый операнд за вторым в лексикографической последовательности,
* операция "сортируется за" ]].

Конкатенация соединяет оба операнда в одну строку, записывая операнды последовательно слева направо (пример 3.14).

USER>W "conca"\_"te"\_"nation"

concatenation

Пример 3.14. Конкатенация

Рассмотрим операторы строкового сравнения =, [ , ] и ]] . Оператор = проверяет строки на идентичность.

Оператор "содержит" обозначается символом "[ " и возвращает логическое значение. Запись s[t означает проверку того, что строка t входит в строку s хотя бы один раз (пример 3.15):

USER>S x= "abcd"["bc" W x

1

USER>S x= "abcd"["aa" W x

0

Пример 3.15. Вхождение строки

Операторы следования и сортировки могут показаться мелочно сложными, не интересными, а может быть и не нужными. Тем не менее, рассмотрите их внимательно и во всех деталях, чтобы понять как Cache упорядочивает узлы дерева одного уровня.

Оператор "следует за" обозначается как закрывающая прямая скобка "] ". Строка s следует за t, то есть выражение t]s истинно, если выполняется одно из трех условий:

* строка t непустая, а s пустая, то есть пустая строка следует за любой непустой (пример 3.16):

USER>S x="abc"]"" W x

1

Пример 3.16.

* строки s и t непустые и код ASCII первого символа t превышает код ASCII первого символа строки s (пример 3.17):

USER>S x="cd"]"axz" W x

1

USER>S x="cd"]"exz" W x

0

Пример 3.17.

* начала строк совпадают, но для оставшихся частей выполняется одно из двух предыдущих условий (пример 3.18):

USER>S x="aacd"]"aabxz" W x

1

USER>S x="aacd"]"aa" W x

1

Пример 3.18.

Оператор "сортируется за", обозначается ]] . В предыдущем операторе "следует за" оказывается, что вопреки здравому смыслу, но в соответствии с правилами лексикографического упорядочения число 10 предшествует 9 потому, что код первого символа "1" меньше кода "9". Так командная строка S x="10"]"9"W x отрицает, что 10 следует за 9. Оператор ]] устраняет этот недостаток. Проверим:

USER>S x="10"]]"9" W x

1

В самом деле, число 10 больше 9, то есть следует за 9.

Поскольку все типы данных, за исключением больших, записываются строками, необходимо интерпретировать строку в соответствие с ее типом. Важно помнить правило числовой интерпретации строки: "Строка в арифметическом выражении читается слева направо и понимается как число до тех пор, пока встречаются символы цифр (0,. . . ,9), буквы "E" или "e" между двумя цифрами, понимаемые как разделитель мантиссы и порядка в экспоненциальной форме записи числа". На первом символе не входящем в указанный набор числовая интерпретация прекращается. Если строку не удается интерпретировать как число, ей присваивается значение ноль (пример 3.19).

USER>S x="5 apples"+"7 books" W x

12

USER>S x=+"abc" W x

0

USER>S x=-".7e2cccw" W x

-70

USER>S x=".7e2cccw" W x

.7e2cccw

Пример 3.19. Интерпретация строки

Работая со строками, нужно уметь определить длину строки (это делает функция $LENGTH), выделить фрагмент строки (функция $EXTRACT), проверить вхождение подстроки в строку (функция $FIND или оператор "[" ). Может потребоваться замена одних символов другими (функция $TRANSLATE).

Функция $L[ENGTH] возвращает длину строки символов. При работе с простыми строками без разделителей используется одноаргументая форма функции (пример 3.20):

USER>W $L("12345678")

8

USER>W $L("")

0

Пример 3.20. Одноаргументая форма функции

Функция $E[XTRACT] выделяет из указанной строки подстроку. Второй аргумент указывает начальную позицию, с которой вырезается подстрока, а третий аргумент — конечную позицию (пример 3.21):

USER>W $E("12345678",3,5)

345

USER>W $E("Those evening bell",7,13)

evening

Пример 3.21. Выделение подстроки

Второй и третий аргументы, если они имеются, интерпретируются как целые числа.

USER>W $E("Those evening bell",7.9,12.9)

evenin

Если значение второго аргумента, определяющее первую позицию меньше нуля, то выделение начинается с первой позиции строки:

Если третий аргумент превышает длину исходной строки, строка выделяется до конца. Этим пользуются для задания условия "до конца строки". Достаточно в третьем аргументе указать число, превышающее максимально допустимую длину строки.

USER>W $E("Those evening bell",15,9999)}

bell

Если позиция начала подстроки больше позиции ее конца, скажем $E("12345678",10,5), то функция возвращает пустую строку. Пустая строка выделяется и тогда, когда начальная позиция подстроки больше длины исходной строки.

Функция $F[IND] определяет, встречается ли строка заданная ее вторым аргументом в строке заданной первым аргументом. Если подстрока не найдена, то возвращается ноль:

USER>W $F("12345678","x")

0

Если подстрока найдена, возвращается номер позиции следующей за концом найденной подстроки:

USER>W $F("12345678","45")

6

Полезно понять, зачем выдаётся именно эта позиция.

В трехаргументной форме функции $FIND третий аргумент задает номер символа исходной строки, с которого начинается поиск. Поскольку время поиска, пропорционально числу проанализированных символов, задание третьего аргумента может существенно уменьшить время поиска:

USER>W $F("12345678","8",7)

9

Если значение третьего аргумента превышает общую длину исходной строки и второй аргумент не пустая строка, то возвращается ноль.

В соответствии с классической теорией множеств в Cache ObjectScript пустая строка содержится в любой строке, в том числе и в пустой:

USER>W $F("123","")

1

USER>W $F("","")

1

Для поиска всех вхождений подстроки в строку необходимо использовать циклы (пример 3.22).

USER>S p=1 F {S p=$F("1111", "1",p) Q:p=0 W !,p}

2

3

4

5

Пример 3.22.

Функция $TR[ANSLATE] используется с двумя аргументами

![$TR[ANSLATE](выражение1, выражение2)](data:image/png;base64,)

или тремя

![$TR[ANSLATE]( выражение!,  выражение2, выражение3)](data:image/png;base64,)

В двухаргументной форме из строки, определённой первым выражением удалятся все символы, имеющиеся в строке, определённой вторым выражением. Например:

USER>W $TR("стpокa","оa")

"стрк"

В трёхаргументной форме строки s2 и s3, определённые вторым и третьим выражениями, определяют таблицу трансляции. Символу в n-ой позиции строки s2 соответствует символ в n-ой позиции строки s3. Если длина s2 больше длины s3, то символы s2, у которых нет соответствия, будут удалены:

USER>W $TR(&QUOT;3TO e-mail", "maile-", "мыло")

Это мыло

**Строки с разделителями**

Элементы строк с разделителями называют полями.

Символ-разделитель нельзя употреблять в самих полях. Одной строкой можно записать, например, список фамилий работников отдела "Иванов^Петров^Сидоров". В качестве разделителя можно использовать любой печатаемый символ, в том числе и пробел. Естественно, в этом случае поле не должно содержать пробел. Поля также могут быть строками с разделителями, но разделитель внутри поля должен быть другим. Можно и эти поля представить в виде переменных с разделителями и т.д.

Похоже, что многоуровневое вложение полей не очень удобно. Тем не менее, строки с разделителями могут существенно расширить возможности базы данных. В качестве мысленного упражнения полезно представить этот раздел без рисунков в виде многоуровневой строки с разделителями. Какие разделители выберете?

Очевидно, для строк с разделителями должны поддерживаться следующие операции:

* определение количества полей в строке;
* выделение поля;
* вставка значения в нужное поле, вставка дополнительных полей.

В Cache ObjectScript для решения этих задач используются функции $LENGTH, $FIND, $EXTRACT, $PIECE.

Итак, выбираем в качестве разделителя печатаемый символ, который не может использоваться в тексте полей, скажем: """, "+" и т. д. Строка с разделителями создаётся обычным присваиванием (команда "SET"). Создадим две строки str1="A~77~BD" и str2="A+B~A/B~A B" (пример 3.23), определим их длины (не в символах, а в полях) и выделим поля со второго по третье в строке str1.

USER>S str1="A^77^BD"

USER>S str2="A+B^A/B^A B"

USER>W "Длина str1=",$L(str1,"^")

Длина str1=3

USER>W "Длина str2=",$L(str2,"^")

Длина str2=3

USER>W $Piece(str1,"^",2,3)

77^BD

Пример 3.23. Строки с разделителями

Количество полей определяется уже известной функцией $L[ENGTH] , в которой во втором аргументе записывается знак разделителя. Выделяется нужное поле функцией $PIECE, имеющей формат:

![$P[IECE](строка, разделитель, первое_поле,последнее\_поле)](data:image/png;base64,)

Например, функция $P(str,"^",1) выделяет первое поле, а $P(str, "^",1,3) — поля с первого по третье. Функция $PIECE относится к так называемым левым функциям. Это означает, что она может стоять в левой части команды присваивания. Вставим текст "QQ" на место второго поля, добавим четвёртое поле и седьмое поля (пример 3.24).

USER>S str1="A^77^BD"

USER>Set $P(str1,"^",2)="QQ" write str1

A^QQ^BD

USER>s $P(str1,"^",4)="PP" w str1

A^QQ^BD^PP

USER>s $P(str1,"^",7)="PP" w str1

A^QQ^BD^PP^^^PP

Пример 3.24. Левая функция $PIECE

Заметим, что вставка в седьмое поле привела к созданию пустых пятого и шестого полей "A^QQ^BD^PP^^^PP", что и вызвало появление в записи строки двух дополнительных циркумплексов.

**Списки**

Одна из основных структур данных — список — создаётся специальной функцией $LISTBUILD (сокращенно $LB). Её формат:

$LISTBUILD(элeмeнт\_спискa [,элемент_списка ...])

Создадим список из двух элементов и с помощью команды ZZDUMP разберёмся с его структурой (пример 3.25).

USER>S x=$LB("Red","Blue")

USER>ZZDUMP x

0000: 05 01 52 65 64 06 01 42 6C 75 65

Пример 3.25. Структура списка

В левой части строки коды символов, а в правой отпечатанная строка. Точками обозначены непечатаемые символы. Оказывается, элементы списка разделяются парами непечатаемых символов. Первый символ такой пары — это число в шестнадцатиричной системе счисления, равное количеству символов в следующем за разделителем элементе списка плюс 2. Вторым идёт символ с кодом 01. Разделители в конце списка не ставятся. Зададим список из одного пустого элемента и список из единственного элемента, представляющего пустую строку (пример 3.26).

USER>S y=$LB(), z=$LB("") ZZDUMP y,z

0000: 01

0000: 02 01

Пример 3.26. Пустой элемент и пустая строка — не одно и то же

При вставке в список значения не определённой переменной ошибка не появляется, а вставляется NULL (пример 3.27).

USER>Kill a Set list1=$LB(1,a,3)

USER>For i=1:1:3 Write "i=" ,i , ?10, $LISTGET(list1,i),!

i=1 1

i=2

i=3 3

Пример 3.27. Вставка значения не определённой переменной

Двухаргументная функция $LISTGET была использована для извлечения из списка указанного первым аргументом элемента с номером, указанным вторым аргументом.

Результат конкатенации двух списков есть список. Для работы со списками кроме их создания необходимо определять правильность списков (функция $LISTVALID) и их длину (функция $LISTLENGTH), сравнивать списки ($LISTSAME). Навигация по спискам производится с использованием функций $LISTDATA, $LISTFIND, $LISTGET, $LISTNEXT. Предусмотрены преобразования строк с разделителями в списки (функция $LISTFROMSTRING) и обратное преобразование ($LISTTOSTRING).

###### Функция $LISTVALID

Булева функция $LISTVALID проверяет правильность списка. Для правильных списков возвращает 1, а для неправильных 0. Формат:

\$LISTVALID(вырaжeниe)

или

\$LV(вырaжeниe)

Проверяемый список должен быть создан с помощью функций $LIST-BUILD и $LISTFROMSTRING, либо извлечён из существующего списка с помощью $LIST. В примере 3.28 приведено несколько примеров, достаточных, чтобы разобраться с деталями.

USER>Set r="A", s=33, t="", u=$LB("A","B"), v=$LB(1)

USER>Write $LV(r),” ”,$LV(s),” ”,$LV(t)

0 0 1

USER>W $LV(u),?40,$LV(v)

1 1

USER>S y=$LB(NULL) w $LV(y)

1

Пример 3.28. Функция $LISTVALID

Вы видите, что пустой элемент t есть правильный список.

###### Функция $LISTLENGTH

Возвращает длину списка. Неопределённые элементы засчитываются. Вложенные списки учитываются как один элемент.

\$LISTLENGTH(список)

или

\$LL(список1)

###### Функция $LISTSAME

Проверяет совпадение списков. Формат:

\$LISTSAME(список1, список2)

или

\$LS(список1, список2)

Примеры употребления приведены в примере 3.29.

USER>s x=$LB("A","B"), y=$LB("B","A"), z=$LB("A","B"), v=$LB(), w=$LB("")

USER>w "$LS(x,y)="\_$LS(x,y),!,"$LS(x,z)="\_$LS(x,z),!,"$LS(v,w)="\_$LS(v,w),!

$LS(x,y)=0

$LS(x,z)=1

$LS(v,w)=0

Пример 3.29. Функция $LISTSAME

Обратите внимание на то, что пустая строка (в примере в примере 3.28) и список из одного пустого элемента (x) оба допустимые списки, но они не равны между собой.

###### Функция $LISTDATA

Формат:

\$LISTDATA(список, позиция)

или

\$LD(список, позиция)

Функция $LISTDATA проверяет указанный элемент списка и возвращает единицу, если элемент в такой позиции существует и имеет значение. Если нет такой позиции или значение элемента не определено, то вернётся ноль (примере 3.30)

USER>kill set x=$LB("Один", ,y,"","Пять")

USER>s pos="Позиция "

USER>for i=0:1:6 w pos,i,?15,"$LD(x,",i,")=",$LD(x,i),!

Позиция 0 $LD(x,0)=0

Позиция 1 $LD(x,1)=1

Позиция 2 $LD(x,2)=0

Позиция 3 $LD(x,3)=0

Позиция 4 $LD(x,4)=1

Позиция 5 $LD(x,5)=1

Позиция 6 $LD(x,6)=0

Пример 3.30. Функция $LISTDATA

В $LISTDATA, как в остальных функциях для работы со списками, "список" —это выражение, дающее имя существующего списка. Если такого списка нет, возникает ошибка UNDEFINED.

###### Функция $LISTFIND

Функция $LISTFIND это наша старая знакомая $FIND приспособленная для работы со списками. Она ищет первое вхождение указанного значения в качестве элемента списка. Совпадение должно быть точным. Формат:

\$LISTFIND(список,  значение, начальная\_позиция)

или

\$LISTFIND(список, значение)

Если элемент найден, возвращается его номер, а если не найден, вернётся 0 (пример 3.31).

USER>S x=$LB("A","BB","B","C","A")

USER>W "$LF(x,""A"")=",$LF(x,"A")

$LF(x,"A")=1

USER>W "$LF(x,""B"")=",$LF(x,"B")

$LF(x,"B")=3

USER>W "$LF(x,""BB"")=",$LF(x,"BB")

$LF(x,"BB")=2

USER>W "$LF(x,""D"")=",$LF(x,"D")

$LF(x,"D")=0

USER>W "$LF(x,""A"",2)=",$LF(x,"A",2)

$LF(x,"A")=5

Пример 3.31. Функция $LISTFIND

Прикрепление пустого элемента, как к голове, так и к хвосту списка, не изменяет список (пример 3.32).

USER>ZZDUMP x

0000: 03 01 41 04 01 42 42 03 01 42 03 01 43 ..A..BB..B..C

USER>ZZDUMP ""\_x

0000: 03 01 41 04 01 42 42 03 01 42 03 01 43 ..A..BB..B..C

USER>ZZDUMP x\_""

0000: 03 01 41 04 01 42 42 03 01 42 03 01 43 ..A..BB..B..C

Пример 3.32. Подключение пустого элемента в голову или хвост не меняет список

Необходимо помнить, что сравнение выполняется после приведения данных к так называемой канонической форме. Для списка созданного командой

S y=$LB("1.0","+2","003","2\*2")

все следующие функции возвращают 0 потому, что числа не приводятся к текстовым константам: $LF(y,1.0), $LF(y,+2), $LF(y,003), $LF(y,4). А для списка созданного командой S y=$LB(7,6) обе следующие функции возвращают 1 потому, что вторые аргументы этих функций согласуются с элементами списка: $LF(y,7.000), $LF(y, 006).

###### Функция $LISTGET

Возвращает элемент списка в указанной позиции или значение по умолчанию, если этот элемент не определён. Формат:

\$LISTGET(список, позиция [, значение\_по\_умолчанию])

или

\$LG(список, позиция [, значение\_по\_умолчанию])

В примере 3.33 показан пример использования функции $LISTGET и сравнение её с функцией $LIST, которая выдаёт ошибку при появлении неопределённого элемента.

USER>S y=$LB("A",,"B")

USER>F i=1:1:3 W "Элемент ",i,?12,$LG(y,i,"null"),!

Элемент 1 A

Элемент 2 null

Элемент 3 B

USER>F i=1:1:3 W "Элемент ",i,?12,$LI(y,i),!

Элемент 1 A Элемент 2

F i=1:1:3 W "Элемент ",i,?12,$LI(y,i),!

^

<NULL VALUE>

USER>

Пример 3.33. Сравнение функций $LISTGET и $LIST

###### Функция $LISTNEXT

Извлекает элементы списка последовательно. Работает очень быстро. Формат:

\$LISTNEXT(список,  указатель, значение)

Локальная переменная "указатель" перед первым вызовом функции должна быть установлена в 0. Тогда список будет анализироваться от начала.

Локальная переменная "значение" не может быть массивом. Инициализировать её не следует. Функция будет помещать в неё значения элементов списка (пример 3.34).

USER>S y=$LB("A","B","C"), p=0

USER>F i=1:1:10 W "i=",i,?6,"Ф=",$LISTNEXT(y,p,v),?10,"p=",p,?20,"v=",v,!

i=1 Ф=1 p=589827 v=A

i=2 Ф=1 p=589830 v=B

i=3 Ф=1 p=589833 v=C

i=4 Ф=0 p=0 v=C

i=5 Ф=1 p=589827 v=A

i=6 Ф=1 p=589830 v=B

i=7 Ф=1 p=589833 v=C

i=8 Ф=0 p=0 v=C

i=9 Ф=1 p=589827 v=A

i=10 Ф=1 p=589830 v=B

Пример 3.34. Функция $LISTNEXT

Когда $LISTNEXT достигнет конца списка, она вернёт 0, переустановит указатель в 0 и оставит предыдущее извлечённое "значение". Продолжится анализ списка с начала. Элементы вложенных списков не выделяются. Удобно использовать функцию с циклом WHILE. Проверьте командную строку: S p=0 WHILE $LISTNEXT(y,p,v) {W !,v}.

###### Функция $LISTFROMSTRING

Создаёт список из строки с разделителями. Формат:

\$LISTFROMSTRING(строкa, ограничитель) 

или

\$LFS(строкa, ограничитель)

Пример приведен в примере 3.35.

USER>S string="A^B^C", list=$LFS(string,"^")

USER>ZZDUMP list,string

0000: 03 01 41 03 01 42 03 01 43 ..A..B..C

0000: 41 5E 42 5E 43 A^B^C

USER>

Пример 3.35. Преобразование строки в список

###### Функция $LISTTOSTRING

Преобразует список в строку с разделителями. Если разделитель не указан, выбирается запятая. Отсутствие флага или нулевое его значение определяет появление ошибки <NULL VALUE>. Формат:

\$LISTTOSTRING(список[,огрaничитeль][,флaг])

или

\$LTS(список[,огрaничитeль][,флaг])

Обратите внимание на то, что функция $LTS не проверяет наличие в исходном списке разделителя, который будет использован в преобразовании. Поэтому в последнем примере в примере 3.36 ошибочно получена строка из трех элементов с разделителем "пробел".

USER>S x=$LB("A","B"), y=$LTS(x,"разделитель")

USER>ZZDUMP y

0000: 0041 0440 0430 0437 0434 0435 043B 0438 Aраздели

0008: 0442 0435 043B 044C 0042 тельB

USER>S z=$LB("A B","C")

USER>S y=$LTS(z," ")

USER>ZZDUMP y

0000: 41 20 42 20 43 A B C

USER>

Пример 3.36. Функция $LISTTOSTRING