**Разветвления и циклы. Виды косвенности. XECUTE.**

**Разветвления, циклы**

Разветвления в COS могут организовываться тремя способами:

* условными опциями команд (командное постусловие);
* разветвлениями языка М (команда IF, отдельная команда ELSE, системная переменная $TEST, хранящая логическое значение истинности последней проверки условия);
* классической командой ветвления (команда IF-ELSEIF-ELSE, переменная $TEST при работе с условиями не используется).

Главное различие между вариантами IF в том, что классическая команда основана на разбиении программы на блоки, а старая команда языка М этого не предполагает, хотя и допускает.

Блоком называется последовательность команд COS, заключенная в фигурные скобки. Блок может быть помещен везде, где может помещаться единственная команда. Блоки могут быть вложены.

###### *Условные опции команд*

Любая команда может быть снабжена необязательным постусловием, записываемым через двоеточие:

имя\_команды[:<логическое выражение>]

Команда с постусловием выполняется только если логическое выражение принимает значение ИСТИННО. Вспоминаем, что в COS принята следующая трактовка числовых значений истинности: 0 это ложно, любое другое значение — истинно. В качестве примера вводите разные значения, исполняя командную строку

R z W:z=1 !,"z=1'"

В команде передачи управления GOTO постусловия могут следовать как за командным словом GOTO, так и за любым ее аргументом:

G[OTO][:логическое_выражение]?[аргумент][,аргумент\dots],

где

аргумент ::= метка[:логическое\_выражение] | @простейшее\_выражение

Последний вариант @простейшее\_выражение это косвенность, которой мы займёмся позже.

Команда

\sqcup G:x>=1 METKA1:y=1, METKA2:y>1,METKA3 работает, если выполнено условие x > 1, при этом переход производится к первой метке если y = 1, ко второй если y > 1 и к третьей в остальных случаях (y < 1). Необходимость простановки пробела перед G объясняется тем, что команду имеет смысл использовать не в командной строке, а в программе. В строке программы команды записываются со второй позиции, а сами программы рассмотрим чуть позже.

Заметим, что переход из GOTO в другую программу или другую строку той же программы выполняется без возврата в следующую за вызвавшей строку. Если требуется возврат управления, используют команду D[O], в которой за командным словом также может помещаться постусловие.

*Замечание*. В COS метка — это любое имя. В инструментальном средстве "Студия", которую мы рассмотрим на следующих занятиях, метка записывается с первой позиции строки.

###### *Разветвления языка М*

Формат команды IF:

\sqcup I[F] <аргумент>[,<аргумент>...] <все\_команды\_до\_-конца\_строки>,

где

<аргумент>::=<логическое\_выражение>|@<проcтейшее_выражение>

Каждый аргумент — это условие, возвращающее логическое значение. Системная переменная $TEST хранит логическое значение последней проверки. Набор аргументов команды IF образует условие IF (пример 3.7)

USER>Set x=1 If x=1 Write "x=",x, " $TEST="\_$TEST

Пример 3.7.

Если в IF имеется несколько аргументов, то они образуют сложное логическое условие в котором аргументы соединены логическим оператором И (AND).

Команда ELSE сделана независимой от IF. Ее формат:

E[LSE]\sqcup\sqcup<все\_команды\_до\_конца\_строки>

Часть строки, идущая вслед за ELSE выполняется, если $TEST=0. В противном случае управление передается следующей строке. Обратите внимание, что после ELSE записывается два пробела (пример 3.8)

USER>S x=1 I x=2 W "x=", x

USER>W "$TEST="\_$TEST

USER>E W x

Пример 3.8.

Существует еще безаргументный IF, записываемый в виде:

I[F]\sqcup\sqcup<все\_команды\_до\_конца\_строки>

Понятно, что он использует последнее значение $TEST, установленное другими командами. При сложных проверках, осуществляемых в ветвящихся и циклических программах, использование такого синтаксиса может сильно затруднить отладку.

Заметим, что постусловия, в отличие от команды IF, не используют переменную $TEST и потому они не могут повлиять на команду ELSE.

###### Классическая команда ветвления

Все ветви разветвления передаются единственной командой:

\verb*|I[F] <аргумент>[,<аргумент>,...]{<блок>}|

\verb*|ELSEIF <аргумент>[,<аргумент>,...]{<блок>}|

\verb*|ELSE {<блок>}|

Например:

\verb*| R x|

\verb*| I x=2 \{W !,"x=2"}|

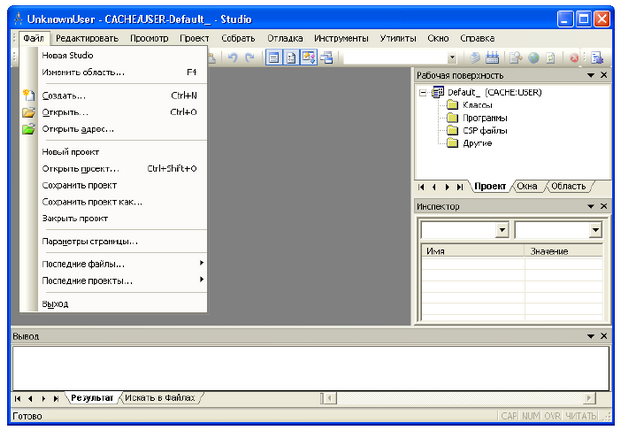
\verb*| ELSEIF x=3 \{W !,"x=3"}|

\verb*| ELSE \{W !,"???"}|

\verb*| Q|

Вы, конечно, обратили внимание, на необычную запись последнего примера и приведённой выше команды GOTO. Дело в том, что многострочная запись команды в терминале не работает и реализовать ее можно только в программе, создаваемой в инструментальном средстве Студия. А там действует правило: с первой позиции в строке записываются только метки программ, а команды занимают позиции со второй и далее. Упомянутая команда GOTO также предназначена для использования в программе.

Запустим из меню куба Cache ещё один инструмент — Студию (рисунок 3.15).

[](https://www.intuit.ru/EDI/06_03_17_3/1488752433-23564/tutorial/959/objects/3/files/03_15.png)

**Рис. 3.15.** Студия

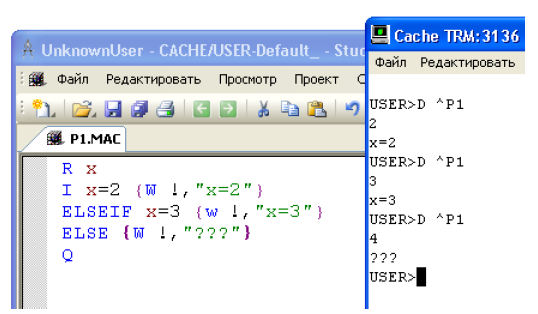
При выборе в подменю позиции "Создать" будет предложен выбор варианта. Для создания программы будем останавливаться на выборе по умолчанию "Cache ObjectScript программа". В "Объектные модели данных" в студии будут создаваться классы (вариант "Класс Cache"). Многочисленные остальные возможности в рамках занятий не рассматриваются.

Наберем текст IF в точности так, как указано в последнем примере. Попробуйте набрать какую-нибудь команду с начала строки. На поле слева появится красная стрелка - признак ошибки в строке. Текст команды будет подчёркнут красным.

Есть два варианта компилирования программы. В первом выберите в меню последовательность "Файл> Сохранить как", затем укажите имя программы, например P1. Расширение имени пока роли не играет. Оставьте предлагаемое расширение .mac. Обратите внимание на то, что имя программы чувствительно к регистру, так что P1 и p1 это разные имена.

Во втором варианте выберите в головном меню позицию "Собрать", а затем "Компилировать" (или просто наберите комбинацию клавиш Ctrl+F7). В любом варианте выполняется компиляция.

Поскольку программа создана успешно и сохранена в виде глобала AP1.mac, ее можно вызвать из терминала с помощью команды D[O] —ВЫПОЛНИТЬ. Расширение имени указывать не обязательно. Проверьте в терминале её работу (рисунок 3.16)



**Рис. 3.16.** Первая программа

Переменная $TEST в этой форме разветвления не задействована.

###### Циклы

В ObjectScript, как и в большинстве языков программирования, имеются три основных типа циклов:

* цикл с параметрами "FOR";
* цикл с предусловием "WHILE";
* цикл с постусловием "DO ...WHILE".

Структуры команды FOR:

\verb*|F[OR] счётчик\_цикла=параметр программная\_строка|

\verb*|F[OR] счётчик\_цикла=параметр {блок}|

F[OR] блок

В однострочном варианте тело цикла образуют все команды, помещённые после команды FOR до конца строки. В простейшем случае параметр имеет одну из форм:

список\_выражений\_разделённых\_запятой;

начальное\_значение:инкремент;

начальное\_значение:инкремент:конечное\_значение

Значения счётчика цикла могут и возрастать и убывать и принимать числовые и строчные значения вперемешку.

В сокращённых версиях цикла отсутствует проверка конца. Поэтому: необходимо в теле цикла организовать выход из цикла, например,

For i=1:1 Write i, ! set i=i+1 Quit:i>5

Примеры:

F i=1:1:5 W i, !

В Студии можно использовать циклы с блоками (пример 3.9)

FOR {

READ !,"Number:", num

QUIT:num=""

W !,num

}

Пример 3.9. Цикл с блоками

Заметьте, здесь использована простейшая безаргументная форма цикла.

Можно использовать сколько угодно параметров FOR.

Пример: F i=1:1:4, "X", 23:2:30 w !, i

При зацикливании программы наберите сочетание клавиш Ctrl/C. А теперь сами попытайтесь предсказать, как будет работать цикл: F i=1:(i<3):4 W !, i

WHILE — цикл с предусловием, DO —WHILE — цикл с постусловием. Их форматы:

\verb*|WHILE условие блок|

и

\verb*|DO блок WHILE условие|

Условие завершения в цикле WHILE выполняется до выполнения блока, а в цикле DO-WHILE после. Если значение истинно (то есть отлично от 0), то осуществляется новая итерация цикла WHILE, в противном же случае цикл завершается.

Пример цикла WHILE приведен в примере 3.10

SET i=0

WHILE (i<10) {

SET i=i+1 WRITE i,!

}

QUIT

Пример 3.10. Пример цикла WHILE

Результатом выполнения данной программы будет вывод на экран чисел от 1 до 10.

Выход из тела цикла возможен с помощью команды передачи управления по метке (GOTO).

**Косвенность и XECUTE**

В зависимости от промежуточных результатов полученных программой может меняться ход вычислений. Разветвления и циклы меняют направление прохода по алгоритму и число повторений циклов. Может потребоваться значительно больше: изменить имена, параметры или индексы, сформировав их в программе. Эти проблемы решаются использованием косвенности. Максимально гибкими оказываются программы, которые могут исполнить на ходу сформированные фрагменты текста. Для этого используется команда со странным названием XECUTE, образованным от слова execute (исполнить).

Заметим, что программы, изменяющие себя самих без всяких ограничений, конечно же, интересны, но вряд ли удобны для работы с бизнесом. Представьте, что, написав программу сегодня, назавтра вы уже не знаете, с какой именно программой работаете. Полезными могут оказаться программы, у которых меняются отдельные разделы.

Косвенность позволяет сформировать меняющиеся имена переменных и программ, аргументы, индексы и элементы шаблонов. Различают четыре варианта:

* косвенное имя
* косвенный аргумент
* индексная косвенность
* косвенный шаблон

###### *Косвенное имя*

Рассмотрим косвенное имя. Косвенность в имени обозначается символом @, проставляемым перед любым простым выражением. Простое выражение — это любое выражение языка, не содержащее бинарных операторов. Можно использовать и выражение с бинарными операторами, заключенное в скобки.

Начнем с простого примера (пример 3.11, первая командная строка).

USER>Set a="b", b=77 write a,!, @a

b

77

USER>S a1=1, a2=2, a3=3 For i=1:1:3 Write @("a"\_i), " "

1 2 3

USER>S a1=1,a2=2,a3=3, ab="ab" F i=1:1:3,"b" {W @("a"\_i)}

123ab

Пример 3.11. Косвенное имя

Здесь мы задали переменной a значение "b", переменной b присвоили значение 77 и распечатали командой WRITE сначала значение а, затем @a. Поскольку использована косвенность, печаталось значение той переменной, имя которой записано в а, то есть 77.

Следующие два примера (пример 3.11, вторая и третья командные строки) показывают формирование имени переменной в цикле с помощью косвенности. Имена переменных создаются в цикле с помощью конкатенации. Это позволяет в цикле читать набор переменных a1, a2, a3 не образующих массива. Скобки в операторе W обязательны. Иначе косвенность распространялась бы только на переменную a, тогда как необходимо использовать выражение "a"\_1.

Косвенность вкладывается в косвенность на любую глубину, например, S a="b", b="c", c=9 w a, !, @a, !, @@a

Предскажите, а затем проверьте результат выполнения следующей строки:

S a1=1, a2=2, a3=3 F i=1:1:3 W @"a"\_i

Для того, чтобы завершить тему косвенного имени рассмотрим пару примеров, которые рекомендуется выполнить после изучения программ в ObjectScript. Пусть имеется любая программа PRG.INT с меткой MET1. Косвенность применима к именам программ и к меткам:

USER>Set rout="prg" Do A@rout

USER>S a="MET1", b="PRG" Do @aAPRG, MET1A@b, @aA@b

В последнем примере три раза подряд вызывается PRG с метки MET1.

Вызов программы с косвенным именем может снабжаться постусловием: USER>D A@rout:x=1

Порядок действий при использовании косвенности следующий: сначала раскрывается косвенность, затем выясняется существование программы и только потом проверяется условие. Предположим, что вызываемая программа не существует. Если бы сначала проверялось условие, то при его ложности, так и не удалось бы установить, что программа не существует.

###### *Косвенный аргумент*

Косвенный аргумент позволяет программе задать как один аргумент, так и список аргументов (пример 3.12, первые четыре командных строки).

USER>Set b=77, a="b=2"

USER>Set @a W b

2

USER>K b,c S x="b=2,c=3",@x W b," ",c

2 3

USER>S b=2,x="b=2" I @x W x

b=2

USER>S b=2,x="b=2" W:@x x

S b=2,x="b=2" W:@x x

^

<SYNTAX>

Пример 3.12. Косвенный аргумент

В первой командной строке переменной b присвоили начальное значение 77. Значением a во второй строке делаем текст "b=2". Косвенный аргумент в команде второй строки "S @a" раскрывает эту команду как "S b=2", и b присваивается значение 2.

А вот попытка переписать эту строку с постусловием во WRITE не удастся (пример 3.12, последняя командная строка). Дело в том, что условие, помещаемое после двоеточия это не имя, и не аргумент. Косвенность здесь не применима.

Аргумент цикла FOR не может быть косвенным, то есть нельзя писать, например,

S x="i=1:1:3" F @x W "Z",

но счетчик цикла FOR может использовать косвенность:

S u="i" F @u=1:1:3 W @u

В цикле WHILE ситуация аналогична, то есть единственный вариант использования косвенности это что-то вроде: WHILE @u<3 {S i=i+1 W i}

В сложных случаях форму косвенности определить не легко, хотя общее правило просто и понятно: имеется та форма косвенности, после раскрытия которой получается синтаксически правильное выражение.

###### *Индексная косвенность*

Дерево в COS представляется многомерными разреженными массивами. Программа для работы с деревьями должна обеспечить доступ к узлам на любом уровне. Индексная косвенность как раз позволяет формировать индексы во время работы программы. Запись имени узла с использованием индексной косвенности:

@имя\_узла@(список\_индексов)

Как видите, особенность синтаксиса индексной косвенности в том, что знак @ повторяется дважды.

Создадим локал, имеющий древесную структуру (пример 3.13) и используя индексную косвенность сформируем команду W, извлекающую значение узла a(1,1,1) .

USER>S a(1)=1,a(2)=2,a(1,1)=11

USER>S a(1,1,1)=111,a(1,1,2)=112

USER>S x="a(1)" w "a(1,1,1)="\_@x@(1,1)

a(1,1,1)=111

Пример 3.13. Индексная косвенность

Обратите внимание на то, что индексная косвенность может использоваться только в приведенной выше форме. Иначе говоря, нельзя опускать один из знаков @ и потому, например, команда "S a(1,1)@(2)=77" вызовет сообщение об ошибке.

*Косвенный шаблон*

Оператор соответствия шаблону – **?**

Оператор **?** – используется в логических выражениях и проверяет соответствие строки шаблону. Синтаксис:

<Строка>?<шаблон>

Если <Строка> соответствует <шаблон>, то выражение возвращает 1 (true), если нет то 0 (false).

<шаблон> состоит из одного или нескольких шаблонов, следующего вида:

<количество><чего>[<количество><чего>]

где <количество> это фактор повторения шаблона, <чего> – класс символов, которые могут повторяться. В первой таблице ниже приведены форматы факторов повторения, а в следующей таблице классы символов.

*Форматы факторов повторения*

|  |  |
| --- | --- |
| Количество | Значение |
| 3(число) | Шаблон должен повторяться ровно столько раз |
| 1.3 | Шаблон должен повторяться от 1 до 3 раз |
| .3 | Шаблон должен повторяться самое большее 3 раза |
| 3. | Шаблон должен повторяться не меньше 3 раз |
| . | Шаблон должен повторяться любое число, включая 0 |

*Классы символов*

|  |  |  |  |
| --- | --- | --- | --- |
| Код | Значение | Код | Значение |
| A | Буква (в верхнем или нижнем регистре) | C | Управляющий символ |
| U | Буква в верхнем регистре | E | Любой символ |
| L | Буква в нижнем регистре | ANP | Комбинация кодов |
| N | Число | “abd” | Литеральная строка |
| P | Пунктуация |  |  |

Примеры:

1. 348?3N – выдаст истину, т.к. строка – это 3 цифры
2. в переменной date находится дата в формате дд.мм.гггг. Следующая строка кода может служить проверкой правильности даты:

if date?2N1P2N1P4N write “Ввод даты верен”.

1. Для даты также может использовать следующий шаблон: 2N1”.”2N1”.”4N.
2. 1U.AP – первый символ – прописная буква, остальные любое количество букв и знаков препинания.

*Косвенный шаблон*

С помощью косвенного шаблона появляется возможность при сравнении по шаблону рассматривать полный шаблон косвенно. Пример:

Set lvmuster=”1.3N”

If eingabe ‘?@lvmuster Do error

###### Команда XECUTE

Команда XECUTE (сокращённо X) позволяет исполнить однострочную подпрограмму, хранимую как значение некоторой переменной:



USER>Set z="Write 22,",

v="!,1+7"

USER>Xecute z\_v

22

8

Write 22,!,1\_7

В результате будет исполнена сформированная командная строка "W 22,!,1+7 ".

Заметим, что исполнение команды может передать управление куда угодно, но адрес возврата из XECUTE остаётся прежний — на командную строку, следующую за XECUTE. Для проверки создадим подпрограмму с именем AA следующего содержания:

W "В подпрограмме "

Q

Исполним команду XECUTE, вызывающую передачу управления:

USER>X "G AA W 77" W "После подпрограммы",! Q

В подпрограмме После подпрограммы

Команда WRITE 77 не была исполнена, что и следовало ожидать.