**Подпрограммы. Метки.**

**Программы, подпрограммы, процедуры и функции**

###### *Программа*

Программа (routine) — это, как всегда, именованный блок кода, предназначенный для выполнения некоторого набора операций. Имя программы не должно содержать знаков подчеркивания ("\_"), тире ("-") и точки с запятой (";"). Используются три расширения имени .mac, .inc и .int. Исходный текст программы, написанной на языке Cache Basic имеет расширение .bas. Файл исходного кода с расширением .mac обрабатывается макропрепроцессором, который транслирует макросы и встроенные SQL и HTML, получая файл с расширением .int. А этот файл транслируется в исполняемый объектный код с расширением obj. Тексты HTML используются в серверных страницах, которые в рамках занятий не рассматриваются. Создавать программу можно в любом текстовом редакторе, но лучше воспользоваться Студией, с которой мы уже познакомились.

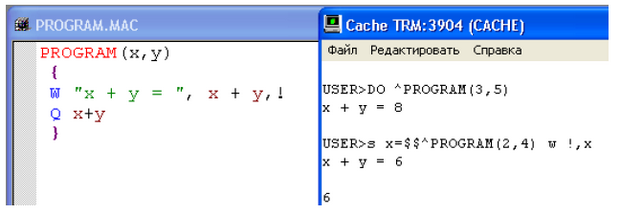
Кроме определённых в разделе 3.2.2 простейших командных строк, состоящих из команд записанных через пробел, в программах используются другие типы командных строк:

* Строка с меткой. Имя метки удовлетворяет соглашению об именах, но может начинаться с цифры. Записывается всегда, начиная с первой позиции строки. Вслед за меткой через пробел может следовать простейшая командная строка.
* Строка с комментарием, начинающимся с символа ";" или с "//". Все символы после точки с запятой игнорируются и при компиляции удаляются.
* Строка с комментарием, начинающимся с символа ";;". Такие комментарии при генерации объектного кода не удаляются и могут быть использованы, например, для хранения констант в тексте программы.
* Многострочный комментарий вида /\*текст комментария\*/. Все символы игнорируются.

По поводу меток необходимо сделать несколько разъяснений. Во-первых, метки могут быть общедоступными (public) и внутренними (private). По умолчанию устанавливается значение public. Такая метка может быть вызвана из любой программы, а метка private — только из текущей программы.

Во-вторых, метки можно снабдить списком формальных параметров, заключённым в круглые скобки. Подпрограмма, которая будет вызвана с такой метки, воспримет переданные параметры.

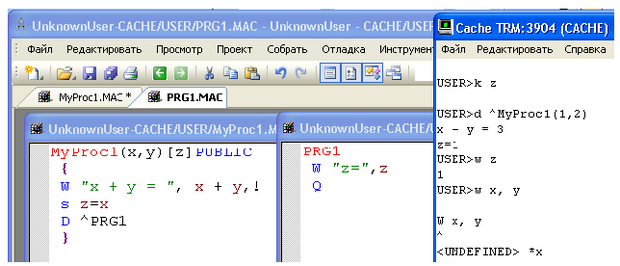
В зависимости от того, возвращает ли программа значение, вызывать её из другой программы или терминала можно двумя способами, командой D[O] или помещением в правую часть присваивания (рисунок 3.18)



**Рис. 3.18.** Два способа вызова программы-функции

Программа может содержать одну или несколько меток. Одна из них может помещаться в самом начале программы перед первой командой. Уже упоминалось, что имя метки пишется в Студии с первой позиции, в отличие от команд, перед которыми помещается минимум один пробел. Простейшая программа приведена на рисунке 3.18.

Тексты двух программ с метками и с вызовом программы приведены на рисунке 3.19. В программе MyProc1 в первой строке указано, что метка снабжена параметрами MyProc1(x,y), а переменная z заключена в квадратные скобки, указывающие, что она глобальная. В терминале видно, что программа MyProc1 просто создала локал z, а переменные x и y не доступны вне MyProc1.



**Рис. 3.19.** Программы и метка public

Существует некоторая "нелогичность" в записи вызова по метке с параметрами. Следует писать фактические параметры после имени программы:

имя\_метки\ \hat{}\ имя\_программы(фактические\_параметры)

По умолчанию сама программа определяется как public. Вариант private смысла не имеет. Программа может содержать в себе подпрограммы, процедуры и функции.

###### *Процедура*

Процедура это именованный блок кода программы, имеющий формат:

имя\_процедуры(формальные\_параметры)\verb*| |[общедоступные\_переменные] \verb*| |доступ\verb*| |{код\_тела\_процедуры}

Доступ определяет, можно ли вызывать процедуру из других программ. Это либо слово private, либо необязательное ключевое слово public (общедоступная).

Процедура начинается с метки-имени процедуры, за которым могут следовать список формальных параметров и список общедоступных переменных. Все метки внутри процедуры имеют тип private и могут использоваться только внутри процедуры. Указание слова public вызовет ошибку. Команда GOTO внутри процедуры может ссылаться только на внутренние её метки. Перед закрывающей фигурной скобкой выполняется неявная команда QUIT. Косвенность и команда XECUTE выполняются как внешняя по отношению к процедуре программа. Все переменные процедуры, за исключением описанных в списке общедоступных переменных, внутренние. Пример процедуры уже рассмотрен нами (рисунке 3.19).

###### *Функция*

Функции бывают внутренние и внешние. Внутренняя функция — это размещённая внутри программы процедура, возвращающая значение. Для этого в команде завершения после QUIT записывают выражение, значение которого возвращается, например, Q x+y, как в программе на рисунке 3.18.

Итак, функция отличается от подпрограммы только способом завершения. Для этого в теле функции помещается команда

quit\ выражение

Для вызова внутренней функции достаточно указать её имя. При вызове внешней функции перед именем помещается два знака доллара.

Пример: процедуру, описанную следующим образом в программе:

^demofunc

sum(a,b)

{

W "Вычисляем сумму "\_а\_"и"\_b,!

Q a+b

}

можно вызвать и как процедуру

USER>DO sum^demofunc(1,2)

Вычисляем сумму 1 и 2

и как функцию — с возвратом значения

USER>WRITE $$sum^demofunc(1,2)

Вычисляем сумму 1 и 2

3

###### *Вызов по ссылке и по значению*

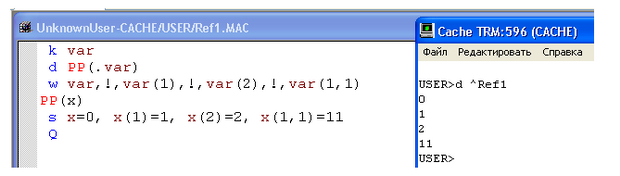
Вызовы по ссылке и по значению понимаются в обычном смысле. При вызове по значению для формальных параметров в начале исполнения вызываемой программы выполняется неявная команда NEW, а по завершении её неявная команда KILL. Поэтому формальные параметры существуют только в вызываемой программе.

При вызове по ссылке вызываемая программа получает указатель на переменную вызывающей программы. Поэтому любое действие в вызываемой программе — присваивание и даже удаление командой KILL — влияет на эти переменные.

Имя переменной вызываемой по ссылке отмечается точкой перед именем.

Два важных замечания:

* Параметром, вызываемым по ссылке, может быть локальная индексированная переменная.
* При вызове по ссылке переменная, записанная в качестве фактического параметра, может не существовать (рисунок 3.20).



**Рис. 3.20.** Вызов по ссылке

Заметим, что в последнем примере при вызове по ссылке с неопределённым фактическим параметром .var после выхода из подпрограммы образовался локал с древесной структурой var, var(1), var(2), var(1,1), копирующий структуру локала x заданного в подпрограмме.

###### *Как хранятся тексты программ*

Вспомним, что в ObjectScript используется четыре типа файлов программ с расширениями имён .mac, .inc, .int и .obj. Исходный текст программы (макрокод) помещается в файл с расширением .mac.

Файлы .inc содержат программы, включаемые в основную программу .mac с помощью команды include, что определило расширение имени. Основное назначение — построение макробиблиотек.

Файл .mac компилируется в файл промежуточного кода с расширением .int, а затем в исполняемый код, помещаемый в файл .obj. Для исполнения программы используется только объектный код.

Формат команды include:

\verb*|#Include имя|\_файла

Исполняемая программа загружается в пространство адресной памяти, называемое разделом. Промежуточный код, представляемый как файл с расширением .int хранится в глобале ^ROUTINE, находящемся в том же пространстве имён, что и сама программа. Структура глобала:

^ROUTINE("имя программы",0,0) = количество строк

^ROUTINE("имя программы",0,1) = первая строка

^ROUTINE("имя программы",0,2) = вторая строка и

т.д.

Остальные типы программ тоже размещаются в глобалах: .mac в ^rMAC, .inc в ^rINC, .obj в ^rOBJ. Структура файла ^rMAC такая же, как у ^ROUTINE. При вызове программы по имени без расширения сначала ищется файл .mac, а затем .int. Хранятся старые версии файлов .mac и .inc. Предпоследние версии имеют расширения .mac.1 и .int.1. У более старых версий расширения .mac.2 и .int.2.

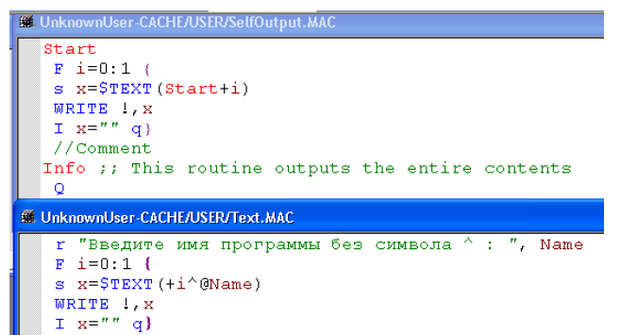
Если, например, командой SET изменить узлы ^ROUTINE, то объектный код не будет совпадать с исходным текстом. Однако, Cache не пытается синхронизировать версии .int, .mac, и .obj. Только после новой компиляции они будут соответствовать друг другу.

Для чтения исходных текстов будем использовать функцию $T[EXT]. Её синтаксис:

\$TEXT(мeтка+смeщeниe\ \hat{}\ программа)

Можно использовать косвенность $TEXT(@выражeниe). Функция возвращает строку исходного текста программы, хранящуюся в указанном месте.

Разберитесь с текстами программ (рисунок 3.21), одна из которых распечатывает свой текст (SelfOutput.mac), а вторая (Text.mac) может распечатать текст любой программы.



**Рис. 3.21.** Программы, распечатывающие программы

###### Программы, изменяющие себя и другие программы

В ObjectScript программа может изменять себя и другие программы. Основные команды, участвующие в этом процессе, удобно рассматривать на основе имеющейся несколько архаичной возможности создания и изменения программы непосредственно из терминала. Введём в терминале несколько командных строк, начиная их знаком табуляции:

USER><TAB> командная\_строка

…

USER><TAB> командная\_строка

Затем без символа табуляции наберём команду сохранения и компиляции программы

USER>ZSAVE \verb*| |имя\_программы

Ниже приведен пример ввода программы в терминале и её исполнения.

USER> W "Исполнена первая строка",!

USER> W "Исполнена вторая строка",!

USER>ZSAVE test

USER>D ^test

Исполнена первая строка

Исполнена вторая строка

USER>S y="ZINSERT "" W 77"":+2"

USER>X y

USER>D ^test

Исполнена первая строка

Исполнена вторая строка

77

Вставка в программу в качестве третьей строки команды W 77 выполнена с использованием XECUTE для предварительно сформированной строки, в которой определён аргумент в виде команды, которая будет вставлена "w 77" и задано смещение от начала программы +2. Это ZINSERT "W 77":+2.

Команда

\verb*|ZR[EMOVE] смещение|

удаляет строку.

С помощью Студии проверьте существование этой программы. Учтите, что она создаётся с расширением .int.

Для работы с текстами программ используют ещё две команды:

Удаление программы из раздела \verb*|ZR[EMOVE] имя|

загрузка программы в раздел \verb*|ZLOAD имя|

*Замечание*. Для практических целей следует использовать программу ^%r и класс %Library.Routine, которые работают гораздо быстрее.

**Измерение времени исполнения**

В COS имеется уникальная возможность измерения промежутков времени с разрешением по-видимому до нескольких микросекунд. С этой целью используется функция $ZH, которая фиксирует временную метку:

USER> w $zh

3819.544684

Первая часть числа 3819 —это число секунд от включения машины до времени исполнения $ZH, а вторая часть это дробные доли секунды.

Если отметить время до и после выполнения команды, командной строки или другой конструкции, то разность временных меток даёт длительность процесса.

Значительные разбросы результатов получаются из-за того, что невозможно остановить процессы операционной системы, не связанные с исполнением нашей программы. Поэтому для получения достоверных результатов необходимо многократно повторять измерения и усреднять оценки.

Исполняя программу ^zhtime.mac (пример 3.37) при одном и том же числе повторений и при разном их числе можно прийти к выводу о том, что единичное исполнение функции $ZH, даёт большие разбросы, а при числе повторений более 105 получаются стабильные результаты (таблица 3.2).

r "Введите число повторений: ", N

s time=0

f i=1:1:N {

s t0=$zh,t1=$zh,dt=t1-t0,time=time+dt

}

s time=time/N

w !,"Время исполнения $zh = ",time\*10e6, " мкс"

q

Пример 3.37. Промежуток между двумя засечками времени

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Таблица 3.2. Время между двумя исполнениями $ZH | | | | | | | | | |
| Повторов | 1 | 10 |  |  |  |  |  |  |  |
| Время (мкс) |  |  |  |  |  |  |  |  |  |

На некоторой машине с Windows XP время между двумя последовательно выполняемыми засечками времени составляет около 2,34 мкс.

Как показывают результаты экспериментальной проверки, полученное время между исполнениями $ZH не следует вычитать из общего времени, полученного в результате применения конструкции вида

s\verb*| |t0=$ZH\verb*| |исследуемая\_структура\verb*| |s\verb*| |t1=$ZH, dt=t1-t0

Это было бы необходимо, если бы задержка происходила только в результате исполнения программы. По-видимому, задержка определяется в основном ограниченными возможностями аппаратной части компьютера, не обеспечивающей более частые обращения к средствам хранения системного времени.

Для измерений длительности выполнения созданной вами программной конструкции вставьте её между командами S t0=$ZH и S t1=$ZH в программу в (примере 3.37).