Введение в синтаксис

Последние изменения: 21:23 13.09.2010

Оглавление

Введение в синтаксис

Особенности языка

Форт относится к стековым языкам программирования, это значит что вся работа представляет собой манипуляции с одним или несколькими стеками (другие языки так же работаю со стеком, но не так явно для программиста). Для удобства работы со стеком запись выражений производится в обратной польской нотации (постфиксной) это значит что операторы записываются после переменных.

Например, сложение двух чисел 3 + 2 на форте будет выглядеть как 3 2 + на первый взгляд удобство такой записи неочевидно, поэтому попытаюсь изобразить наглядно.

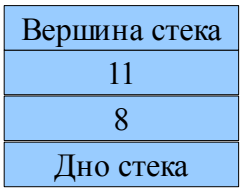
Возьмем выражение посложнее, например (8 + 11) * (12 / 3) на форте это может выглядеть так:

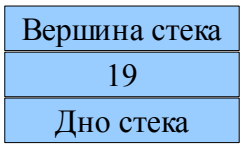
```
8 11 + 12 3 / *
```

Разделителем в форте является пробел — т.е. мы получаем 7 атомов, каждый из которых будем выполнять последовательно, начиная с «8» и заканчивая «*».

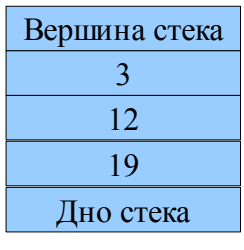
Первые два атома это целые числа они просто кладутся на вершину стека. Стек это такая структура данных похожая на стопку книг, вы кладете книги одну на другую, стек еще называется линейным списком типа FILO (первый вошел последний вышел), и действительно, книга, которую вы положили самой первой вы достанете последней, если будете снимать книги по одной.

Итак «8» и «11», если класть их в пустой стек по порядку, мы получим такую картину — вначале на дно ляжет «8», а затем сверху на нее «11».

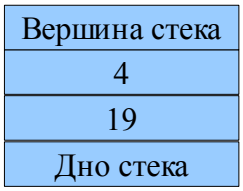




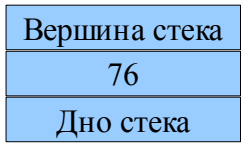
Следующим у нас идет «+», поскольку он точно не является число, интерпретатор ищет в своем списке атомов атом «+» и запускает связанную с ним подпрограмму. В случае с «+» произойдет следующее — из стека извлекутся два числа, а в вершину стека будет помещена их сумма, следовательно, исчезнут «11» потом «8», найдется сумма «8 + 11 = 19» и число «19» отправится в стек.



После этого на стек кладутся еще два числа «12» и «3» а за ними следует атом «/» т.е. деление — снова извлекаются два числа «3» и «12» результат деления «12 / 3 = 4» помещается в вершину стека.



Сразу за «/» идет «*» (умножение), действуем по знакомой схеме — извлекаем два числа из стека, умножаем и помещаем в стек. Результат = у нас в стеке число «76» — результат выражения (8 + 11) * (12 / 3). Что бы программа выглядела законченной, можно добавить в конце «.» — этот атом извлекает число или строку из вершины стека и печатает на экране.



Одним из полюсов обратной польской нотации является отсутствие приоритета операций и как следствие каких либо скобок — очередность всегда линейна и не вызывает разногласий.

Атомы не обязательно берут два верхних значения из стека, они могут брать одно, или вовсе не брать, а могут «съесть» весь стек, например abs — извлекает из вершины число и кладет в стек его модуль — т.е. размер стека не меняется.

Переменные

Переменные в VForth не имеют строгой типизации, и могут без проблем присваиваться друг другу, однако имеются операторы для строгого приведения типа. Большинство операций с переменными происходит на стеке, но вы всегда можете назначить переменной имя.

Числовые

Все числа, вне зависимости то типа и точности занимают одну ячейку в стеке.

Целые

Целые числа можно записывать как

```
1 2 3 $1 $a3fe
```

т.е. в десятичной и шеснадцатиричной форме.

Вещественные

Вещественные числа записываются просто в виде десятичной дроби

```
1,2 7,8
```

(десятичный разделитель может быть точка или запятая, в зависимости от региональных настроек). Либо в виде 2e5 3e-8.

Натуральные

Натуральные числа можно получить, записав их в виде отношения 1/3 7/5. при операциях с такими числами количество разрядов в числителе и знаменателе растет, но периодически числа упрощаются. Поскольку операция упрощения (поиска наибольшего кратного) может быть достаточно продолжительной — она работает в отдельном потоке.

Комплексные

Записываются в своей естественной комплексной форме

```
2+4i
```

Строковые

Строковые константы записываются в двойных кавычках

```
"Привет мир!"
```

Массивы

Для создания массива применяется атом vector (size --). Он извлекает целое число из вершины стека и создает массив заданной размерности. В режиме выполнения размер

массива можно менять. Подробнее о работе с массивами буде рассказано ниже.

Структуры

Записываются между ключевыми словами record..end вес что кладется в стек между этими словами, пакуется в одну переменную. Доступ к полям можно производить по индексу или по имени.

Присвоение имени

Любой переменной в стеке можно присвоить имя делается это при помощи атома var.

```
0 var i
```

атом var считывает следующее за собой значение (у нас это «i») и назначает это в качестве имени переменной в вершине стека. Выполнение продолжается после «i».

Если теперь в коде написать «i» то в вершине стека окажется еще одна переменная «i», это очень тонкий момент. Для работы с переменными существуют атомы «@» и «!» Первый читает значение переменной и помещает ее в стек, а второй устанавливает значение переменной. В следующем примере используются еще один атом «.s» он ни как не влияет на ход выполнения программы, просто отображает на экране стек.

```
0 var i .s
Integer(i)=0
```

По порядку вначале в стек помещается значение «0». Затем в дело входит «var», он считывает следующий за собой атом «i» и назначает его в качестве имени переменной в стеке (т.е. i=0). Атом «i» будучи уже задействованным не выполняется, и управление переходит на «.s» который печатает нам содержимое стека «Integer(i)=0». Здесь мы видим, что в вершине стека лежит число «0» с именем «i».

```
i .s
Integer(i)=0
Integer(i)=0
```

В этом случае атом «i» не является частью «var», поэтому выполняется. В результате в стек снова попадает «i». Обратите внимание, что это та же самая переменная. Теперь попробуем первый атом «@», который извлекает значение из переменной

```
@ .s
Integer=0
Integer(i)=0
```

Все просто, атом «@» извлекает значение из переменной в вершине стека. Если раньше у нас в вершине была «i=0» то теперь там просто «0».

Второй атом «!» назначает значение переменной, записывается это как «значение переменная !»

```
13 i ! .s
Integer=0
Integer(i)=13
```

Мы видим что значение «i» изменилось на «13» попробуем теперь сделать i = i + i

```
i i .s
Integer(i)=13
Integer(i)=13
Integer=0
Integer(i)=13
+ .s
Integer=26
Integer=0
Integer(i)=13
i ! .s
Integer=0
Integer(i)=26
```

Если в предыдущем примере сделать вот так

```
i @ i @ + i !
```

То результат будет тот же самый, но тогда мы будем складывать не «i + i» а «13 + 13». Как я уже говорил — этот момент довольно тонкий, очень хороший пример в данном случае цикл do..loop.

Еще один важный момент — переменая не может иметь больше одного имени, поэтому повторный вызов var переименует переменную.

Win32

Для работы с функциями из dll требуется строгая типизация, поэтому имеется ряд атомов, которые могут явно задать тип переменной (байт, слово, двойное слово, Ansi-строка, Unicode-строка и т.д.). Такой тип ни как не повлияет на переменную в пределах VForth, но при вызове некоторых атомов, эта информация буде учитываться:

W32Bool — Булевая величина (1 байт);

w32Byte — Байт;

w32Word — Слово (2 байта);

w32Int — Цлое (4 байта);

w32CharA — Символ Ansi;

w32CharW — Символ юникода;

w32PCharA — Строка Ansi;

w32PCharW — Юникодная строка;

w32Pointer — Указатель;

Попробуем создать структуру tagWNDCLASSA

```
tagWNDCLASSA = record
```

```
    style: UINT;
    lpfnWndProc: TFNWndProc;
    cbClsExtra: Integer;
    cbWndExtra: Integer;
    hInstance: HINST;
    hIcon: HICON;
    hCursor: HCURSOR;
    hbrBackground: HBRUSH;
    lpszMenuName: PAnsiChar;
    lpszClassName: PAnsiChar;
  end;
record
    0 w32Int ( style )
    0 w32Int ( lpfnWndProc )
    0 w32Int ( cbClsExtra )
    0 w32Int ( cbWndExtra )
    0 w32Int ( hInstance )
    0 w32Int ( HICON )
    0 w32Int ( HCURSOR )
    0 w32Int ( HBRUSH )
    "#32770" w32PCharA ( lpszMenuName )
    "Привет мир!" w32PCharA (  lpszClassName )
end
```

В круглых скобках записываются комментарии, скобки должны быть отделены пробелами,

так например (Привет мир) — не это уже атом, а (Привет) мир) закроется после слова «мир» потому что между «мир» и «)» стоит пробел.

Стоит упомянуть что структура на самом деле является массивом. Т.е. что бы создать массив из трех елементов мы можем смело написать.

```
Record 1 2 3 end
```

Все что было положено в стек после Record и до End будет извлечено из него и помещено в массив.

Работа со стеком

Для манипуляций со стеком есть несколько основных атомов

Stack и Stack@

В VForth существует 10 стеков (с «0» по «9», первые восемь пользователь может использовать под свои нужды, 8-й и 9-й используются самой VFroth-машиной и в них лечше

ни чего не трогать.

«Stack» извлекает из стека целое число и переключает машину на стек с этим номером.

«Stack@» кладет в стек номер текущего стека.

8 стеков возможно слишком много, но зато вы точно не будете испытывать в них недостатков.

Swap

«Swap» меняет местами два верхних значения в стеке.

Dup

Делает копию верхнего элемента в стеке. Вот пример возведения в квадрат.

```
4 dup * .s
16
```

Вначале в стек кладется 4, потом делается копия — получаем стек (4 4) а затем умножаем следует умножение.

Dup*

Делает то же что и Dup но копирует не значение, а саму переменную, более наглядно.

```
10 var i dup dup .s
Integer=10
Integer=10
Integer(i)=10
10 var i dup* dup* .s
Integer(i)=10
Integer(i)=10
Integer(i)=10
```

В первом случае мы получим «i 10 10», а во втором «i i i».

Over

Копирует значение второго элемента стека и помещает его в вершину.

```
1 2 over .s
1 2 1
```

Rot

Drop

Просто удаляет из стека верхнее значение.

Атомы

Атомы это своего рода подпрограммы в языках высокого уровня. Вспоминм в прошлый раз мы возводили число в квадрат

```
4 dup * .
16
```

А теперь введем новое слово, которое будет возводить число в квадрат:

```
:sqr2 dup *;
```

Ни чего не произошло, но мы добавили новый атом, теперь когда мы будем писать «sqr» вместо него будет выполняться «dup *»

```
:4 sqr2 .
16
```

Что бы удалить атом необходимо вызвать «Forget»

```
Forget sqr2
4 sqr2
Variable "sqr2" not found
```

Получаем сообщение об ошибке.

Ветвление и циклы

if..else..then

Оператор ведвления, при встече if из стека извлекается число, если оно не равно нулю, то выполняется код между «If...Else» если нет, то между «Else...Then». Так же имеет краткую форму «If...Then» тогда кода выполняется только в случае не нулевого значения в вершине стека.

```
:IsZero if "Не ноль" else "Ноль" then .;
1 IsZero
Не ноль
0 IsZero
Ноль
```

Do...Loop

Begin...Until

Цикл доходит до «Until» и извлекает число из стека, если это «0» то управление возвращается на «Begin», если нет, выполняется код после «Until»

Ввод-вывод. Файлы.

Консольный ввод-вывод

Для консольного ввода-вывода существует несколько простых атомов.

«.» - точка

Выводит на экран содержимое переменной из вершины стека

Emit

Извлекает из вершины стека целое число и выводит его как символ ansi.

Space

Выводит на экране пробел, аналогичен последовательности

```
" " .
```

Spaces

Выводит на экране столько пробелов, сколько содержит в себе число в вершине стека.

Cr

Переход на новую строку, аналогично

```
10 13 . .
```

.in

(-- String)

Запрашивает у пользователя строку и помещает ее в стек.

.out

(String --)

Аналогично «.»

.err

(String --)

Аналогично «.», только используется для вывода ошибок.

Файлы

VForth обладает не сильно блгатым, но минимально полезным набором функций для работы с файлам.

Двоичные файлы

Представляют собой последовательность байт которую можно читать как угодно, например как последовательность 8 чисел по 1 байту или двух 4-х байтных, или как строку, завершающуюся нулем.

При чтении-записи используется информация о Win32Type, поэтому нужно быть максимально внимательным.

FOpen

(OpenMode FileName -- File)

OpenMode это строка — аналог FileMode в си, может принимать значения «w» «r» «a» «w+» «r+» «a+».

Второй параметр — FileName — строка, содержащая абсольтный или относительный путь к файлу.

Стоит заметить что следующие две функции ведут себя по разному в зависимости от того, какой файл в вершине стека.

FRead

(Var File -- Var)

Берет из стека файл, а затем записывает в переменную, оказавшуюся в вершине стека данные их файла, согластно ее Win32Type.

Предположим у нес есть файл в котром записано следующее:

«Привет мир!\0\52\65537» - строка (ansi), потом нулевой символ (признак конце строки) потом однобайтное значение и четырехбайтное значение:

```
«r» «config.data» fopen var Cfg ( открываем файл на чтение
                                  и даем переменной имя )
«» w32PCharA ( создаем переменную без имени и делаем ее
               Ansi-строкой )
    Cfg Fread ( в стеке у нас уже есть переменная, кладем
                сверху ссылку на файл и читаем, после этого
                ссылка на файл извлекается, а в переменной
                будет строка )
    . Cr      ( которую мы и выведем на экран )
    ( аналогично с байтом и двойным словом )
0 w32Byte cfg Fread . Cr
0 w32Int cfg Fread . Cr
( файл закрывается автоматически, при уничтожении последней,
  связанное с ним ссылки )
```

```
drop
```

FWrite

(Var\Value File --)

Берет файл из вершины стека и записывает в него содержимое переменной из стека, согластно ее Win32Type.

Теперь пример записи в файл. Процедура мало отличается от Fread — выходе мы получим то самый файл, который читали выше.

```
«w» «config.data» fopen var cfg
    «Привет мир!» w32PCharA cfg Fwrite
    52 w32Byte cfg Fwrite
    65537 w32Int cfg Fwrite
drop
```

Файлы конфигурации

FOpenCfg

(FileName -- File)

Файл конфигурации всегда открывается для чтения\записи и в случае каких либо изменений, записывает содержимое обратно.

FRead

(Var Key File -- Var)

Читает из фала File значение, соответствующее ключю Key, и помещает его в Var

Имеем исходный файл cfg.txt

```
foo=Hello
bar=world!
```

Прочитаем его содержимое

```
«cfg.txt» FopenCfg var cfg
«» «foo» cfg fread . space
«» «bar» cfg fread .
Drop
Hello world!
```

FWrite

(Var\Value Key File --)

Записывает пару Key=Var в файл File.

```
«user.txt» FopenCfg var cfg
```

```
«Благодарев» «Фамилия» cfg Fwrite
«Евгений» «Имя»  cfg Fwrite
«Сергеевич» «Отчество» cfg Fwrite
drop
```

В результате будет создан файл user.txt

```
Фамилия= Благодарев
Имя=Евгений
Отчество= Сергеевич
```

Еще важное замечаниее, при присвоенни ключу пустой строки, ключ удаляется.

Fclose

(File --)

Используется для закрытия файла.

Исключения

Raise

(String --)

Возбуждает исключение в виде строки.

Простые примеры кода

Цикл выводит массив 9 x 9

```
0 var i
1 9 do
    1 9 do
        i 1 + 1 ! ( увеличиваем i на 1 )
        i .
        9 emit ( символ табуляции )
    loop
    cr
loop
drop ( уничтожаем i )
```

```
1   2   3   4   5   6   7   8   9
10  11  12  13  14  15  16  17  18
19  20  21  22  23  24  25  26  27
28  29  30  31  32  33  34  35  36
37  38  39  40  41  42  43  44  45
46  47  48  49  50  51  52  53  54
55  56  57  58  59  60  61  62  63
64  65  66  67  68  69  70  71  72
73  74  75  76  77  78  79  80  81
```

Использование переменной в цикле

```
0 var i
i 9 do ( переменная будет изменяться в цилке )
    i . space
loop
( тут в i еже лежить «9» )
drop
```

```
0 1 2 3 4 5 6 7 8 9
```

Для вложенных циклов важно не забывать обнулять переменные.

```
1 var i
1 var j
i 9 do
    j 9 do
```

```
        i j * .
        9 emit
    loop
    cr
    1 j ! ( j = 9 сбрасываем j, иначе цикл j..9 просто не
            выполнится второй раз )
loop
drop drop
```

```
1   2   3   4   5   6   7   8   9
2   4   6   8   10  12  14  16  18
3   6   9   12  15  18  21  24  27
4   8   12  16  20  24  28  32  36
5   10  15  20  25  30  35  40  45
6   12  18  24  30  36  42  48  54
7   14  21  28  35  42  49  56  63
8   16  24  32  40  48  56  64  72
9   18  27  36  45  54  63  72  81
```

А вот такая запись будет не совсем верной:

```
0 var i 9 do i . space loop
```

В таком случае переменная i исчезнет из стека при входе в цикл, и вы не сможете к ней обратиться. Хотя если вы не будете использовать переменную в цикле (0 var i 9 do «hello! » . loop) то код будет работать.

Процедура логирования

```
:log ( line -- )
    var line
    "log.inf" FOpenCfg var file
    line now DateTime file Fwrite
drop ( file )
drop ( line );
```

Очень простоя процедура для ведения лога достаточно написать «Hello world!» log и строка «Hello world!» Запишется в файл «Log.inf».

Есть небольшое ограничение — ее нельзя вызывать чаще чем раз в секунду, иначе сохранится только последняя запись.

Алфавитный указатель