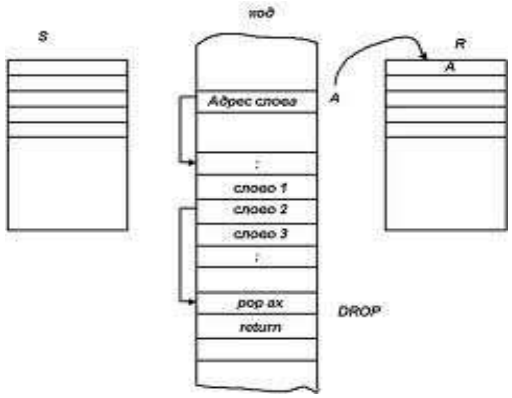
Язык программирования, не имеющий синтаксиса



Программа на любом языке программирования состоит из последовательности слов, обозначающих её объекты или играющих роль знаков препинания. Объекты программы – это числа, переменные, имена процедур, операторы. Знаки препинания – разделители, как и в естественных языках, такие как точки, запятые. К разделителям относятся и так называемые ключевые слова: begin, end, if и т.п. На профессиональном языке они называются *лексемами*. Лексемы строится из фиксированного набора символов, образующих *алфавит* языка. В комментариях допускаются любые символы. Программа представляет не произвольную последовательность лексем, а последовательность, построенную по определенным правилам, заданным *синтаксисом* языка.

А может ли язык не иметь никакого синтаксиса? Оказывается, может! Это язык Форт (Forth), придуманный Чарльзом Муром в 60-е годы прошлого века. Его еще называют космическим языком (видимо, из-за того, что одной из первых систем, написанной на нем самим Муром, была система управления телескопом). В этом языке любая последовательность слов, составленных из любых символов, может быть программой! Но как же исполняется такая программа?

На вход интерпретатора поступает последовательность слов, которые немедленно исполняются. Поэтому, конечно, если слово не известно интерпретатору, исполнение прекращается с фиксацией ошибки. Так поступают все интерпретаторы. Хотя можно было бы сделать интерпретатор, который просто пропускает неизвестные слова. Если все слова неизвестны, результат пустой.

В первую очередь нужно уметь производить вычисления. Поэтому слова, представляющие целые числа и знаки арифметических операций, считаются известными. Основной механизм исполнения – *стек*. (Надеемся, что читателям известна это структура данных, работающая по принципу LIFO: последним пришёл – первым вышел). Исполнение числа – помещение его в стек. Исполнения слова «+» – извлечение из стека двух чисел, их сложение и помещение результата в стек. Слово «.» выталкивает слово из вершины стека и выводит его на консоль. Так, если ввести последовательность

```
2 3 + .,
```

на выходе получим в следующей строке

```
5 Ok.
```

В программировании такая запись арифметических выражений называется польской инверсной записью (ПОЛИЗ), которая было предложена польским математиком Лукасевичем в 1920 году.

Некоторые интерпретаторы, например, SP-FORTH[], после слова «Ok» выводят значения, находящиеся в стеке:

```
5 DUP DUP * *
Ok ( 125 ).
```

Здесь слово «DUP» дублирует значение в вершине стека. После исполнения двух таких слов в стеке оказываются три пятерки, которые затем перемножаются и в стеке остается 5 в кубе.

Интерпретатору SP-FORTH версии 4.20 известно 946 стандартных слов. Чтобы определить новое слово, используются «служебные» слова «:» и «;». Первое считывает следующее за ним слово, выделяет память в словаре – основной механизм Форта – и «транслирует» все последующие слова в словарь, пока не встретит слово «;». Например, после

```
: куб DUP DUP * * ;
```

в словарь заносится слово «куб», исполнение которого определяется указанными четырьмя действиями. Теперь можно использовать это новое слово:

```
3 куб ( 125 от предыдущего примера )
Ok ( 125 27 ).
```

Нет никаких ограничений, каким может быть новое слово. Можно даже переопределять стандартные слова. Например, определить слово «1» как «2»:

```
: 1 2 ; 1 1 + . ( должно быть 2, но 1 заменено на 2, поэтому 4 )
4 Ok ( 125 27 ).
```

Старые значения слов сохраняются в словаре. Если теперь выполнить команду FORGET 1, слово «1» будет забыто, т.е. стерто из словаря вместе со всеми словами, определенными позже. В SP-FORTH вместо FORGET используется MARKER.

Среди слов Forth'а есть такие, которые не выполняются в консольном режиме, а могут присутствовать только внутри определений. К ним относятся слова, управляющие последовательностью исполнения: IF, THEN, DO, LOOP и др. Пример определения слова для вычисления факториала:

```
: factorial ( N ---> N! )
  DUP 2 < IF
     DROP 1
  ELSE DUP
    BEGIN
      1 - SWAP OVER * SWAP DUP 1 =
    UNTIL
    DROP
  THEN ;
```

В этой программе слово DROP удаляет значение из стека, SWAP – переставляет два верхних значения в стеке, а OVER – дублирует на вершину второе сверху значение в стеке. Если после выполнения слова «<» (сравнение аргумента с числом 2) в стеке оказывается истинное значение, выполняется часть кода между IF и ELSE, иначе – между ELSE и THEN. Тело цикла выполняется несколько раз, пока после «=» (сравнение уменьшающегося значения аргумента с 1) не окажется в стеке истинное значение. Для лучшего понимания работы алгоритма необходимо расписать пошаговое состояние стека.

Покажем, как может быть выполнена любая последовательность слов. Предположим мы хотим, чтобы выполнялись программы типа

```
Числа один два три сложить,
```

где можно было бы писать различные числа и действия. Для этого предварительно надо «научить» интерпретатор этим словам:

```
: Числа 0 ; один 1 ; : два 2 ; : три 2 ; / и так все числа ☺
: сложить BEGIN + OVER 0 = UNTIL SWAP DROP ;
: умножить BEGIN * OVER 0 = UNTIL SWAP DROP ;
```

Слова между словами «BEGIN» и «UNTIL» повторяются до тех пор, пока перед выполнением «UNTIL» в стеке не окажется истинное значение. Это, как известно, называ-

ется *циклом*. Логическое значение вырабатывается словом «=». В стеке перед эти находится очередное число и занесенный ноль. Число будут складываться справа налево, пока не встретится ноль (слово «Числа»), играющийт роль ограничителя.

Я познакомился с Фортом, когда приобрел свой первый персональный компьютер БК-0010, на котором были еще языки программирования Бэйсик и Фокал. Язык Фокал как-то не приглянулся, а с Фортом было очень интересно работать. Впоследствии, когда появились IBM XT и система программирования Турбо-Паскаль, я написал свой интерпретатор Форта на Паскале (потом на Delphi).

Из всех языков высокого уровня Форт отличается простотой реализации интерпретатора. Как подсказку для тех, кто решится его реализовать, приведу фрагмент программы со списком слов, реализованных в ядре, т.е. на базовом языке:

```
{ 1}  '+',       '-',       '*',       'mod',     'div',
{ 6}  '.',       'dup',     'drop',    'over',    'rot',
{11}  'swap',    ':',       ';',       '(',       'ekran',
{16}  'voc',     'load',    'edit',    'here',    'pick',
{21}  '@1',      '@2',      '@3',      '@4',      '@5',
{26}  '@6',      '@7',      '@8',      '!1',      '!2',
{31}  '!3',      '!4',      '!5',      '!6',      '!7',
{36}  '!8',      '!G',      '1+',      'neg',     'and',
{41}  'or',      'roll',    'sqrt',    'not',     '<',
{46}  '=',       '>',       '0<',      '0=',      '0>',
{51}  '."',      'create',  'does>',   'i@',      ',',  ',b', 'b,', '()',
{59}  '!',       '@',       'allot',   'key',     'emit',
{64}  'expect',  'c@',      'c!',      '(DO)',    '(LOOP)',
{69}  'I',       'LEAVE',   'compile', '<resolve','>resolve',
{74}  '<mark',   '>mark',   'immediat','"',
{78}  '?g+',     '?g-',     'g+',      'g-'
```

Как видно, вовсе нет необходимости реализовывать все 946 слов. На самом деле большая часть слов в любой версии Форта реализуются средствами самого языка обычным определением через «:». Число базовых слов может быть уменьшено. Например, слово «over» могло бы реализовано так:

```
: over swap dup rot rot ; ( ab ba baa aab aba )
```

Более интересно реализуются управляющие слова:

```
: IF compile ?g+ >mark ; immediat
: THEN >resolve ; immediat
: ELSE compile ?g+ >mark swap >resolve ; immediat
: DO compile (DO) >mark ; immediat
: LOOP compile (LOOP) >resolve ; immediat
: BEGIN <mark ; immediat
: UNTIL compile ?g- <resolve ; immediat
```

Интерпретатор Форта работает в двух режимах: *интерпретации* и *компиляции*. В режиме интерпретации слова исполняются, в режиме компиляции – компилируются. Исполнение слова «:» переводит интерпретатор в режим компиляции. Теперь слова, точнее их адреса, переносятся в словарь. Но некоторые слова, называемые словами периода компиляции, выполняются. Определение новых слов периода компиляции завершается словом «immediat». Если перед словом стоит «compile», оно компилируется. Ключевую роль в приведенных выше определениях играют слова *периода компиляции* «>mark» и «>resolve», позволяющие генерировать команды перехода «?g+» и «?g-» соответственно вперед и назад. Механизм компиляции аналогичен тому, который используется в компиляторах традиционных языков.

Подобным образом можно реализовать и такие слова как WHILE, CASE или SWICTH, которые будут работать точно так же, как в известных языках Паскаль или C. Есть примеры реализации и объектно-ориентированных возможностей.

Современные Форт-системы самодостаточны, имеют неограниченные возможности и позволяют создавать приложения в самых различных областях. Встроенный интерпретатор, о котором идет речь, также может оказаться полезным. Для реализации интерфейсов, базовых алгоритмов используются освоенные технологии традиционных языков, а встроенный Форт-интерпретатор служит для описания данных или действий (сценариев).

Одним из приложений с использованием встроенного Форт-интерпретатора была система для построения и отображения роликов с трёхмерной графикой. Для этого базовый словарь был расширен словами для элементов графики. Вот некоторые из них:

```
p          x y z  --> i     создать точку  (x,y,z)
L           n1 n2 --> l     создать отрезок на точках n1 и n2
g     i1 ... ik k --> n     создать грань по k точкам
body     nP nL nG --> nB    создать заголовок многогранника
show              -->       перерисовать картинку
rx,ry,rz        a -->       поворот на 0,01*a градусов
move     dx dy dz -->       сдвиг картинки на вектор (dx, dy, dz)
cadr              -->       отобразить кадр
```

Используя набор примитивов, можно определять слова для сложных объектов. Вот как выглядит определение для правильной четырехгранной пирамиды, задаваемой координатами вершины, половиной стороны основания и высотой, параллельной оси Ox:

```
: 4pyr create  ( x0 y0 z0 a h ---> )
  5 8 5 body ,          ( 5 вершин, 8 ребер и 5 граней )
  5 !G                  ( загрузка параметров в переменные @1 @2 … )
  @4 2 * !6
  @1      @2      @3      p
  @1      @2 @6 + @3      p
  @1      @2 @6 + @3 @6 + p
  @1      @2      @3 @6 + p
  @1 @5 + @2 @4 + @3 @4 + p
  5 !G                                  ( загрузка из стека номеров вершин )
  @1 @2 L  @2 @3 L @3 @4 L @4 @1 L      ( ребра основания )
  @1 @5 L  @2 @5 L @3 @5 L @4 @5 L      ( боковые ребра )
  @1 @2 @3 @4 4 g                       ( основание )
  @1 @5 @2 3 g  @2 @5 @3 3 g            ( боковые грания )
  @3 @5 @4 3 g  @1 @4 @5 3 g
  13 0 DO drop LOOP                     ( удаление лишних данных из стека )
  show  does> i@ ;                      ( отображение и запись в словарь  )
```

Нельзя сказать, что такое описание слишком сложно. Думаю, читатель без труда мог бы написать, например, определение параллелепипеда. В другом приложении на Форте описывались действия, производимые в графическом редакторе. Например, для описания действий по построению отрезка использовалась следующая последовательность:

```
Click Click Line
```

При выполнении слово «Click» система ожидала нажатия кнопки мыши и размещала ее координаты в стек, слово «Line», естественно, рисовала линию.

В заключении отметим два факта, свидетельствующие о полезности технологии, положенной в основу Форта. Первый: в 1994 году был принят ANSI-стандарт языка, а второй, что существуют процессоры и контроллеры, поддерживающие вычислительную модель языка на аппаратном уровне.

1. Официальный сайт SP-Forth. http://spf.sourceforge.net/.
2. Progopedia. Forth – язык программирования. http://progopedia.ru/language/forth/
3. Баранов С. Н., Ноздрунов Н. Р. Язык Форт и его реализации. — Л.: Машиностроение, 1988. — 157 с.
4. Броуди Л. Начальный курс программирования на Форте / Пер. с англ.; предисл. И. В. Романовского. — М.: Финансы и статистика, 1990. — 352 с.