Лабораторная работа №4

Интерпретатор стекового языка программирования

Условие задачи

Реализуйте интерпретатор стекового языка программирования, описание которого представлено ниже. Интерпретатор должен вызываться как процедура *(interpret program stack)* которая принимает программу на исходном языке *program* и начальное состояние стека данных *stack* и возвращает его состояние после вычисления программы. Программа на исходном языке задана вектором литеральных констант, соответствующих словам исходного языка. Исходное и конечное состояния стека данных являются списком, голова которого соответствует вершине стека.

Примеры вызова интерпретатора (здесь и далее в примерах код на исходном языке выделен синим цветом):

```
(interpret #(   define abs
                  dup 0 <
                  if neg endif
                end
                abs    ) ; программа
           '(-9))         ; исходное состояние стека
  ⇒ (9)
```

При реализации интерпретатора избегайте императивных конструкций, используйте модель вычислений без состояний. Для хранения программы и состояния интерпретатора **запрещается** использовать глобальные переменные. Перечисленные ниже встроенные слова обязательны для реализации и будут проверены сервером тестирования.

Описание языка

Язык, интерпретатор которого следует реализовать, является видоизмененным ограниченным подмножеством языка Forth.

В нашем языке операции осуществляются с целыми числами. Используется постфиксная запись операторов. Все вычисления осуществляются на стеке данных. Стек данных является глобальным. При запуске интерпретатора стек может быть инициализирован некоторыми исходными данными или быть пустым.

Программа на исходном языке представляет собой последовательность слов. Интерпретатор анализирует слова по очереди. Если слово является целым числом, то оно число помещается на вершину стека данных. В противном случае слово интерпретируется как оператор (процедура). Если в программе уже встретилось определение этого слова (статья), то выполняется код этого определения. В противном случае слово рассматривается как встроенное в интерпретатор и выполняется соответствующей процедурой интерпретатора. Затем осуществляется возврат из процедуры (переход к слову, следующему за последним вызовом). Выполнение программы заканчивается, когда выполнено последнее слово.

Процедуры (операторы) снимают свои аргументы с вершины стека данных и кладут результат

вычислений также на вершину стека данных.

Ввод-вывод или какое-либо взаимодействие с пользователем не предусматривается.

Например:

```
(interpret #(2 3 * 4 5 * +) '()) ⇒ (26)
```

Встроенные слова

Ниже представлен список встроенных слов с кратким описанием их значений. Состояние стека до и после интерпретации каждого слова показаны с помощью схем — стековых диаграмм. Порядок, в котором элементы были помещены в стек, отражен в индексах элементов. Например, программа:

```
1 2 3
```

может быть показана стековой диаграммой () → (1 2 3)

Внимание! В нашем интерпретаторе в качестве стека используется список. Голова этого списка является вершиной стека, поэтому вершина стека в этих диаграммах находится *слева*! Такая запись отличается от традиционных стековых диаграмм, принятых, например, в языке Forth, в которых голова стека записывается *справа*.

Арифметические операции

+	(n2 n1) → (сумма)	Сумма n1 и n2
−	(n2 n1) → (разность)	Разность: n1 − n2
*	(n2 n1) → (произведение)	Произведение n2 на n1
/	(n2 n1) → (частное)	Целочисленное деление n1 на n2
mod	(n2 n1) → (остаток)	Остаток от деления n1 на n2
neg	(n) → (−n)	Смена знака числа

Операции сравнения

=	(n2 n1) → (флаг)	Флаг равен −1, если n1 = n2, иначе флаг равен 0
>	(n2 n1) → (флаг)	Флаг равен −1, если n1 > n2, иначе флаг равен 0
<	(n2 n1) → (флаг)	Флаг равен −1, если n1 < n2, иначе флаг равен 0

Таким образом, булевы значения представлены с помощью целых чисел: −1 соответствует значению «истина», 0 — значению «ложь».

Логические операции

not	(n) → (результат)	НЕ n
and	(n2 n1) → (результат)	n2 И n1
or	(n2 n1) → (результат)	n2 ИЛИ n1

Эти операции также должны давать правильный результат, если в одном или обеих операндах «истина» представлена любым ненулевым целым числом.

Операции со стеком

При выполнении вычислений на стеке часто возникает необходимость изменять порядок

следования элементов, удалять значения, копировать их и т.д. Для этого реализуйте следующие операции:

drop	(n1) → ()	Удаляет элемент на вершине стека
swap	(n2 n1) → (n1 n2)	Меняет местами два элемента на вершине стека
dup	(n1) → (n1 n1)	Дублирует элемент на вершине стека
over	(n2 n1) → (n1 n2 n1)	Копирует предпоследний элемент на вершину стека
rot	(n3 n2 n1) → (n1 n2 n3)	Меняет местами первый и третий элемент от головы стека
depth	(...) → (n ...)	Возвращает число элементов в стеке перед своим вызовом

Управляющие конструкции

define *word*	() → ()	Начинает словарную статью — определение слова *word*
end	() → ()	Завершает статью
exit	() → ()	Завершает выполнение процедуры (кода статьи)
if	(флаг) → ()	Если флаг не равен 0, то выполняется код в теле if..endif, иначе выполнение кода до endif пропускается
endif	() → ()	Завершает тело if

Пусть слово define *word* начинает определение слова *word*. В теле определения (словарной статьи) следуют слова, которые надо вычислить, чтобы вычислить слово word. Статья заканчивается словом end. Определенное таким образом слово может быть использовано в программе так же, как и встроенное. Например, унарный декремент может быть определен, а затем использован так:

```
(interpret #(   define -- 1 - end
                5 -- --      ) '())
  ⇒ (3)
```

Завершить выполнение процедуры до достижения её окончания end можно с помощью слова exit.

В статьях допускаются рекурсивные определения. Вложенные словарные статьи не допускаются.

Конструкции if...endif могут быть вложенными. В программах ниже даны примеры их использования.

Примеры программ

Ниже представлены программы, которые будут выполнены сервером тестирования с помощью вашего интерпретатора (наряду с более короткими примерами).

```
(interpret #(   define abs
                    dup 0 <
                    if neg endif
                end
                 9 abs
                -9 abs      ) (quote ()))
  ⇒ (9 9)

(interpret #(   define =0? dup 0 = end
                define <0? dup 0 < end
                define signum
                    =0? if exit endif
```

```
                    <0? if drop -1 exit endif
                    drop
                    1
                end
                 0 signum
                -5 signum
                10 signum       ) (quote ()))
  ⇒ (1 -1 0)

(interpret #(   define -- 1 - end
                define =0? dup 0 = end
                define =1? dup 1 = end
                define factorial
                    =0? if drop 1 exit endif
                    =1? if drop 1 exit endif
                    dup --
                    factorial
                    *
                end
                0 factorial
                1 factorial
                2 factorial
                3 factorial
                4 factorial     ) (quote ()))
  ⇒ (24 6 2 1 1)

(interpret #(   define =0? dup 0 = end
                define =1? dup 1 = end
                define -- 1 - end
                define fib
                    =0? if drop 0 exit endif
                    =1? if drop 1 exit endif
                    -- dup
                    -- fib
                    swap fib
                    +
                end
                define make-fib
                    dup 0 < if drop exit endif
                    dup fib
                    swap --
                    make-fib
                end
                10 make-fib     ) (quote ()))
  ⇒ (0 1 1 2 3 5 8 13 21 34 55)

(interpret #(   define =0? dup 0 = end
                define gcd
                    =0? if drop exit endif
                    swap over mod
                    gcd
                end
                90 99 gcd
                234 8100 gcd    ) '())
  ⇒ (18 9)
```

Рекомендации

В составе интерпретатора определите главную процедуру, которая будет обрабатывать каждое слово программы. Пусть состояние интерпретатора описывают аргументы этой процедуры: вектор слов, счетчик слов (индекс текущего слова), стек данных, стек возвратов и словарь (ассоциативный список).

Главная процедура классифицирует слово, на которое указывает счетчик, и интерпретирует его как число или слово (определенное в программе или встроенное). Встроенные слова принимают состояние интерпретатора и возвращают его измененным согласно семантике слова.

Изменяться могут счетчик, стек данных, стек возвратов и словарь. Не храните ни их, ни интерпретируемую программу в глобальных или статических переменных (почему?).

Если в программе встречается определение статьи, то в словарь помещается новое слово (ключ) и индекс первого слова в статье (значение).

При вызове такой статьи в стек возвратов помещается индекс слова, следующего за вызовом. Он будет снят с вершины стека и возвращен в качестве значения счетчика слов при возврате из статьи (слова end и exit). Такой подход позволяет интерпретировать вложенные и рекурсивные вызовы. Также в коде интерпретатора целесообразно определить словарь соответствий слов исходного языка встроенным процедурам интерпретатора.

При необходимости организуйте отложенные вычисления. В процессе разработки используйте юнит-тестирование.