Лабораторная работа № 5. Интерпретатор стекового языка программирования

13 января 2024 г.

Дмитрий Лимонов, ИУ9-12Б

Цели работы

На примере языка Scheme научиться писать собственный интерпретатор стекового языка программирования. # Реализация

```
(define (interpret prog stack)
  (let* ((stack stack)
         (prog prog)
         (cur 0)
         (ret '())
         (sl '()))
    (main prog cur stack ret sl 0)))

(define (main prog cur stack ret sl ign)
  (if (>= cur (vector-length prog))
      stack
      (let ((word (vector-ref prog cur)))
        (if (= 0 ign)
            (cond
              ((equal? '+ word) (main prog (+ 1 cur) (cons (+ (car stack) (cadr stack))
                                                           (cddr stack)) ret sl 0))
              ((equal? '- word) (main prog (+ 1 cur) (cons (- (cadr stack) (car stack))
                                                           (cddr stack)) ret sl 0))
              ((equal? '* word) (main prog (+ 1 cur) (cons (* (car stack) (cadr stack))
                                                           (cddr stack)) ret sl 0))
              ((equal? '/ word) (main prog (+ 1 cur) (cons (quotient (cadr stack) (car stack
                                                           (cddr stack)) ret sl 0))
              ((equal? 'mod word) (main prog (+ 1 cur) (cons (remainder (cadr stack) (car st
                                                             (cddr stack)) ret sl 0))
              ((equal? 'neg word) (main prog (+ 1 cur) (cons (- (car stack)) (cdr stack)) re
              ((equal? '= word) (main prog (+ 1 cur) (cons (if (= (car stack) (cadr stack))
                                                               -1
```

```
                                                0) (cddr stack)) ret sl 0))
((equal? '> word) (main prog (+ 1 cur) (cons (if (< (car stack) (cadr stack))
                                                -1
                                                0) (cddr stack)) ret sl 0))
((equal? '< word) (main prog (+ 1 cur) (cons (if (> (car stack) (cadr stack))
                                                -1`
                                                0) (cddr stack)) ret sl 0))
((equal? 'not word) (main prog (+ 1 cur) (cons (if (= 0 (car stack))
                                                  -1
                                                  0) (cdr stack)) ret sl 0))
((equal? 'and word) (main prog (+ 1 cur) (cons (if (= 0 (* (car stack) (cadr s
                                                  0
                                                  -1) (cddr stack)) ret sl 0)
((equal? 'or word) (main prog (+ 1 cur)
                         (cons (if (or (not (= 0 (car stack))) (not (= 0 (cadr
                                     -1
                                     0) (cddr stack)) ret sl 0))
((equal? 'drop word) (main prog (+ 1 cur) (cdr stack) ret sl 0))
((equal? 'swap word) (main prog (+ 1 cur) (cons (cadr stack)
                                                (cons (car stack) (cddr stack)
((equal? 'dup word) (main prog (+ 1 cur) (cons (car stack) stack) ret sl 0))
((equal? 'over word) (main prog (+ 1 cur) (cons (cadr stack) stack) ret sl 0))
((equal? 'rot word) (main prog (+ 1 cur)
                           (cons (caddr stack)
                                 (cons (cadr stack)
                                       (cons (car stack) (cdddr stack)))) ret s
((equal? 'depth word) (main prog (+ 1 cur) (cons (length stack) stack) ret sl
((equal? 'define word) (main prog (+ 1 cur) stack ret
                              (cons (list (vector-ref prog (+ 1 cur)) (+ 2 cur)
((equal? 'if word) (main prog (+ 1 cur) (cdr stack) ret sl (if (= 0 (car stack
                                                             1
                                                             0)))
((equal? 'else word) (main prog (+ 1 cur) stack ret sl 1))
((equal? 'end word) (main prog (car ret) stack (cdr ret) sl 0))
((equal? 'endif word) (main prog (+ 1 cur) stack ret sl 0))
((equal? 'exit word) (main prog (car ret) stack (cdr ret) sl 0))
((equal? 'while word) (main prog (+ 1 cur) (cdr stack) (cons cur ret) sl (if (
                                                                           3
                                                                           0
((equal? 'wend word) (main prog (car ret) stack (cdr ret) sl 0))
((equal? 'repeat word) (main prog (+ 1 cur) stack (cons cur ret) sl 0))
((equal? 'until word) (main prog (if (not (= 0 (car stack)))
                                     (+ 1 cur)
                                     (car ret)) (cdr stack) (cdr ret) sl 0))
((equal? 'tail word) (main prog (cadr (assoc (vector-ref prog (+ 1 cur)) sl))
((number? word) (main prog (+ 1 cur) (cons word stack) ret sl 0))
```

```
            (else (main prog (cadr (assoc word sl)) stack (cons (+ 1 cur) ret) sl 0)))

          (cond
            ((and (equal? 'endif word) (= ign 1)) (main prog (+ 1 cur) stack ret sl 0))
            ((and (equal? 'else word) (= ign 1)) (main prog (+ 1 cur) stack ret sl 0))
            ((and (equal? 'end word) (= ign 2)) (main prog (+ 1 cur) stack ret sl 0))
            ((and (equal? 'wend word) (= ign 3)) (main prog (+ 1 cur) stack (cdr ret) sl 0
            (else (main prog (+ 1 cur) stack ret sl ign))))
      )))
```

Тестирование

```
Welcome to DrRacket, version 8.11 [cs].
Language: R5RS; memory limit: 128 MB.
> (interpret #(   define abs
                  dup 0 <
                  if neg endif
                end
                abs    ) ; программа
           '(-9))        ; исходное состояние стека
(9)
> (interpret #(2 3 * 4 5 * +) '())
(26)
> (interpret #(   define -- 1 - end
                5 -- --      ) '())
(3)
> (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)
```

Вывод

Написание собственного интерпретатора - очень интересное, а главное полезное задание для понимания самих основ программирования. Многие интерпретаторы низкоуровневых языков программирования реализованы подобным способом, а значит такая практика позволяет углубить свои знания в устройстве и понимании языка. Также домашнее задание к данной лабораторной работе углубляет эту тему, заставляя реализовать более сложные конструкции, правила и особенности языка, что также является достаточно интересным и увлекательным занятием.