Задание №20
Макросы

1. Общая постановка задачи

Реализуйте функционал, описанный в задании с номером вашего варианта.

При описании можно использовать средства императивного программирования, но функционал должен быть реализован без побочных эффектов.

Все циклические процессы должны быть реализованы с помощью хвостовой рекурсии.

Все реализованные макросы не должны иметь «протечек» (стоит обратиться к книге «Practical Common Lisp», глава 8).

Порядок вычислений аргументов макроса — от первого до последнего в порядке обхода в глубину.

Функция, использующая макрос, не должна быть громоздкой. Вспомогательные алгоритмы следует оформлять в виде отдельных (глобальных) вспомогательных функций.

Файлу с программой дайте имя `task20-NN.lsp`, где вместо `NN` — номер вашего варианта. Полученный файл загрузите на портал в качестве решения задания.

2. Пример выполнения задания

0. Определите макрос для цикла `while-not-less`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(while-not-less expr exprs)
```

где `expr` — выражение, возвращающее числовое значение; `exprs` — тело цикла, состоящее из одного или нескольких выражений, в котором последнее выражение возвращает числовое значение.

Выражение `expr` должно вычисляться точно один раз. Выражения `exprs` должны вычисляться как минимум один раз. Вычисление выражений `exprs` повторяется пока полученное значение последнего в нем выражения меньше значения `expr`. Результат выполнения макроса — значение последнего выражения в последнем вычислении `body`.

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(let ((i N)  ; счетчик – параметр суммы
      (s 0)) ; аккумулятор суммы
  (if (>= i 2) ; так как цикл с постусловием, то нужно проверить на
               ; необходимость выполнения первой итерации
      (while-not-less 2 ; цикл закончится когда N-е слагаемое уже добавлено
        (incf s (sin i))  ; сумму увеличиваем на (sin i)
        (decf i)))        ; счетчик уменьшаем на 1
  s)
```

Выполните пример задания 11 с использованием разработанного макроса.

РЕШЕНИЕ:
Содержимое файла `task20-00.lsp` :

```
(defmacro while-not-less (expr &body exprs)
  ;; генерируем два вспомогательных идентитфикатора. Один сохраняем
  ;; в переменной func-name, второй – в переменной param-name.
  ;; Первый идентификатор будем использовать как имя нашей рекурсивной
  ;; функции для цикла, а второй – как имя формального параметра этой функции
  (let ((func-name (gensym))
        (param-name (gensym)))
    ;; определяем тело самой макроподстановки с указанными параметрами.
    ;; Заменяем вызов макроса на определение локальной функции
    ;; (с помощью labels) с дальнейшим вызовом этой функции.
    ;; Дальнейшие комментарии касаются ее функционирования.
    ;; Параметр локальной функции – значение выражения expr
    `(labels ((,func-name (,param-name)
                ;; вычисляем  последовательно все выражения, входящие в exprs
                ;; Для этого  поместим все элементы из exprs в тело выражения
                ;; progn. Вычисление такого progn выдаст нам значение
                ;; последнего выражения из exprs. Полученное значение
                ;; сохраняем в переменной new-exprs
                ;; после этого вычисления нет обращения к параметрам макроса
                ;; напрямую, поэтому использование new-exprs не приведет к
                ;; появлению протечек (дыр макроса)
                (let ((new-exprs (progn ,@exprs)))
                  ;; сравниваем полученное значение с параметром функции
                  (if (>= new-exprs ,param-name)
                      ;; если условие цикла выполняется, то вызываем функцию
                      ;; рекурсивно
                      (,func-name ,param-name)
                      ;; иначе выдаем вычисленный результат
                      new-exprs))))
       (,func-name ,expr)))) ; начальный вызов функции от выражения,
                             ; переданного в качестве expr. Выражение
                             ; вычислится только один раз – перед передачей
                             ; его значения в качестве фактического
                             ; параметра

;;; решение примера для задания 11

;; вспомогательная функция вычисляющая модуля разности
;; двух элементов
(defun abs-diff (x1 x2)
  (abs (- x1 x2)))

;; вспомогательная функция вычисления очередного элемента
;; по двум предыдущим: параметры соответствуют обозначениям
;; в функции elemLst в файле task11-00.hs
(defun func (x y0 y1)
  (/ (- (* x y1 y1) 5.0)
     (* y0 y1)))

;; основная функция
(defun f11 (x e n
            &aux          ; вспомогательные переменные
            (y0 1)        ; предпредыдущий элемент (сначала y_0)
            (y1 (/ x 2)) ; предыдущий элемент (сначала y_1)
            (res (list y1 y0)) ; список-результат
                               ; строится в обратном порядке
            tmp) ; вспомогательная переменная
  ;; из-за специфики опреатора while-less (оператор с постусловием)
  ;; сначала проверяем условия на выдачу списка из одного или двух элементов
  (cond ((= n 0) (cdr res)) ; выдаем список из одного элемента
        ((or (= n 1) (< (abs-diff y0 y1) e))
```

```
        (reverse res)) ; выдаем результат, предварительно перевернув
       ;; выход из цикла должен быть когда |y1 - y0| < e
       ;; т.е. цикл будем выполнять пока |y1 - y0| >= e
       (T (while-not-less e
            ;; для вычисления очередного элемента пересохраняем
            ;; два предыдущих в tmp и y0
            (setq tmp y0)
            (setq y0 y1)
            ;; вычисляем очередной элемент и сохраняем его в y1
            (setq y1 (func x tmp y0))
            ;; записываем новый элемент в список res
            (push y1 res)
            ;; уменьшаем каждый проход цикла на 1, чтобы организовать выход
            ;; когда (если) n станет равно 1
            (setq n (- n 1))
            ;; если n стало равно 1, то последним значением в while-less
            ;; выдадим 0, так как он заведомо меньше, чем e,
            ;; чтобы выйти из цикла
            ;; в противном случае выдадим модуль разности двух последних
            ;; вычисленных значений
            (if (= n 1) 0
                (abs-diff y0 y1)))
          ;; когда цикл окончен, в переменной res -
          ;; перевернутый список-результат
          ;; переворачиваем его и возвращаем
          (reverse res))))

;;; ТЕСТОВЫЕ ЗАПУСКИ
;;; чтобы результаты тестовых запусков совпадали с результатами в задании 11
;;; следует задавать аргументы в формате с плавающей точкой с двойной точностью
;;; например, 0.3d0 вместо 0.3
(fresh-line)
(princ "(f11 0.3d0 0.001 500)")
(defparameter *y1* (f11 0.3d0 0.001 500))
(print (nth 10 *y1*))
(print (nth 100 *y1*))
(print (nth 500 *y1*))
(fresh-line)
(princ "(f11 0.5d0 0.001 500)")
(defparameter *y2* (f11 0.5d0 0.001 500))
(print (nth 10 *y2*))
(print (nth 100 *y2*))
(print (nth 433 *y2*))
(print (nth 434 *y2*))
; (print (nth 435 *y2*))
(fresh-line)
(princ "(f11 1.7d0 0.001 500)")
(defparameter *y3* (f11 1.7d0 0.001 500))
(print (nth 10 *y3*))
(print (nth 100 *y3*))
(print (nth 500 *y3*))
```

Файл с примером можно загрузить с портала.

3. Необходимый минимум

Для выполнения работы потребуются сведения о следующих функциях, операциях и конструкциях:

- функции `cons`, `car`, `cdr`
- конструкцию **`if`**
- конструкции **`let`**, **`let*`**, **`labels`**
- логические, арифметические функции, функции сравнения
- конструкцию `defun`
- конструкцию `defmacro`
- конструкции `defparameter`, `defvar`
- конструкции **`setq`**, `setf`
- конструкции `incf`, `decf`, `push`
- конструкции `prog1`, **`progn`**
- предикат `null`
- функции `funcall`, `apply`

- функции вывода на экран `print`, `princ`, `terpri`, `fresh-line`, `format`
- функцию `nth` (только в тестовых запусках)
- математические функции, аналоги функций, используемых в решении заданий 5, 7, 11

Если вы считаете, что для выполнения какого-то из заданий необходима функция/конструкция, отсутствующая в перечислении, то задайте вопрос на форуме «Язык LISP».

4. Варианты заданий

1. Определите макрос для цикла с параметром `for`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(for (parameter start-value end-value step) body)
```

где `parameter` — идентификатор — параметр цикла; `start-value` — начальное значение параметра цикла; `end-value` — конечное значение параметра цикла; `step` — шаг значения параметра цикла — опциональный параметр, равный по умолчанию единице. После выполнения тела цикла значение параметра цикла должно увеличиваться на значения шага; `body` — тело цикла — одно или несколько выражений.

Если шаг цикла положительный, то цикл выполняется пока значение параметра не больше, чем конечное значение. Если шаг цикла отрицательный, то цикл должен выполняться, пока значение параметра не меньше, чем конечное значение.

Результат выполнения макроса — количество выполненных итераций цикла.

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(let ((s 0))
  (for (i 2 N)
    (incf s (sin i)))
  s)
```

Выполните свой вариант задания 7 с использованием разработанного макроса.

2. Определите макрос для цикла с параметрами `for`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(for ((parameter-1 start-value-1)
      ...
      (parameter-k start-value-k))
     (continue-expr cont-body)
     (result)
   body)
```

где `parameter-1` – `parameter-k` — идентификаторы — параметры цикла; `start-value-1` – `start-value-k` — выражения для вычислений начальных значений параметров цикла; `continue-expr` — условие выполнения цикла, проверяется перед каждой итерацией цикла; `cont-body` — одно или несколько выражений, которые выполняются после каждой итерации (после вычисления тела цикла); `result` — одно или несколько выражений, которые выполняются после последней итерации цикла; `body` — тело цикла — одно или несколько выражений. Результат выполнения макроса — значение последнего выражения в `result`.

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(for ((i 2)
      (s 0))
     ((<= i N) (incf i))
     (s)
  (incf s (sin i)))
```

Выполните свой вариант задания 5 с использованием разработанного макроса.

3. Определите макрос для цикла с предусловием `while`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(while condition body)
```

где `condition` — условие выполнения цикла; `body` — тело цикла — одно или несколько выражений. Тело цикла должно исполняться пока результат вычисления условия не равен `NIL`. Результат выполнения макроса — количество выполненных проходов тела цикла.

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(let ((s 0)
      (i 2))
  (while (<= i N)
    (incf s (sin i))
    (incf i))
  s)
```

Выполните свой вариант задания 5 с использованием разработанного макроса.

4. Определите макрос для цикла с постусловием `repeat`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(repeat (body) condition)
```

где `body` — тело цикла — одно или несколько выражений; `condition` — условие окончания цикла; Исполнение тела цикла происходит минимум один раз и повторяется до тех пор, пока результат вычисления условия не станет отличным от `NIL`. Результат выполнения макроса — количество выполненных проходов тела цикла.

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(let ((s 0)
      (i 2))
  (if (<= i N)
      (repeat
        ((setq s (+ s (sin i)))
         (incf i))
        (> i N)))
  s)
```

Выполните свой вариант задания 5 с использованием разработанного макроса.

5. Определите макрос для цикла `do-list`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(do-list (parameter list-expr) body)
```

где `parameter` — идентификатор — параметр цикла; `list-expr` — выражение, значением которого является список; `body` — тело цикла — одно или несколько выражений. Параметр цикла пробегает все значения в заданном списке. Тело цикла выполняется для каждого значения параметра цикла. Результат выполнения макроса — значение последнего выражения в теле `body`. Если список `list-expr` является пустым (т.е. цикл ни разу не выполнится), то результатом выполнения макроса должно быть значение `NIL`.

Опишите функцию `all-in-range`, принимающую два целочисленных аргумента `M` и `N` и возвращающую список всех целых чисел от `M` до `N`.

Так, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(let ((s 0))
  (do-list (i (all-in-range 2 N))
    (setf s (+ s (sin i)))))
```

Выполните свой вариант задания 7 с использованием разработанного макроса и функции `all-in-range`.

6. Определите макрос для цикла `do-range`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(do-range (start-value end-value parameter) body)
```

где `start-value` — начало диапазона; `end-value` — конец диапазона; `parameter` — идентификатор — параметр цикла; `body` — тело цикла — одно или несколько выражений. Тело цикла выполняется для каждого значения параметра в заданном диапазоне с шагом 1, если начальное значение меньше конечного, или с шагом -1, если конечное значение меньше начального. Результат выполнения макроса — значение последнего выражения в теле `body`.

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(let ((s 0))
  (if (>= N 2)
    (do-range (2 N i)
      (setf s (+ s (sin i)))))
  s)
```

Выполните свой вариант задания 5 с использованием разработанного макроса.

7. Определите макросы для суммирования `sum-range` и произведения `prod-range`, реализующие итерационный процесс через вызов функций с хвостовой рекурсией. Формат команд для вызова макроса:

```
(sum-range (parameter start-value end-value) body)
(prod-range (parameter start-value end-value) body)
```

где `parameter` — идентификатор — параметр суммирования/произведения; `start-value` — начальное значение параметра; `end-value` — конечное значение параметра; `body` — тело — одно или несколько выражений. Результат выполнения макросов — соответственно сумма и произведение значений последних выражений в теле `body` для каждого из значений параметра с шагом 1. Если начальное значение параметра больше конечного, то вызов макроса `sum-range` должен возвращать 0, а макроса `prod-range` — 1.

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(sum-range (i 2 N) (sin i))
```

Выполните свой вариант задания 7 с использованием разработанных макросов.

8. Определите макрос для цикла с предусловием `while`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(while (condition result) body)
```

где `condition` — условие выполнения цикла; `body` — тело цикла — одно или несколько выражений; `result` — одно или несколько выражений. Тело цикла должно исполняться пока результат вычисления условия не равен `NIL`. Результат выполнения макроса — значение последнего выражения в `result` (после выполнения `body` достаточного количества раз).

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(let ((s 0)
      (i 2))
  (while ((<= i N) s)
    (setf s (+ s (sin i)))
    (incf i)))
```

Выполните свой вариант задания 11 с использованием разработанного макроса.

9. Определите макрос для цикла с постусловием `repeat`, реализующий итерационный процесс через вызов функции с хвостовой рекурсией. Формат команды для вызова макроса:

```
(repeat (body) (condition result))
```

где `body` — тело цикла — одно или несколько выражений; `condition` — условие окончания цикла; `result` — одно или несколько выражений. Тело цикла должно исполняться до тех пор, пока результат вычисления условия не станет отличным от `NIL`. Результат выполнения макроса — значение последнего выражения в `result` (после выполнения `body` достаточного количества раз).

Например, для подсчёта суммы $\sum_{i=2}^{N} \sin i$ можно составить код с использованием макроса

```
(let ((s 0)
      (i 2))
  (if (> i N) s
      (repeat
        ((setf s (+ s (sin i)))
         (incf i))
        ((> i N) s)))
```

Выполните свой вариант задания 11 с использованием разработанного макроса.