Отчет по лабораторной работе № 4 по курсу «Функциональное программирование»

Студент группы 8О-307 МАИ *Бронников Максим*, №4 по списку
Контакты: max120199@gmail.com
Работа выполнена: 13.05.2020

Преподаватель: Иванов Дмитрий Анатольевич, доц. каф. 806
Отчет сдан:
Итоговая оценка:
Подпись преподавателя:

1. Тема работы

Знаки и строки.

2. Цель работы

Научиться работать с литерами (знаками) и строками при помощи функций обработки строк и общих функций работы с последовательностями.

3. Задание (вариант №45)

Запрограммировать на языке Коммон Лисп функцию, принимающую один аргумент - текст.
Функция должна найти самое длинное слово в тексте (одно или несколько) и вернуть список таких слов без повторений. Слова не должны включать конечные знаки пунктуации.

```
(text-longest-words
        '("Блажен, кто смолоду был молод,"
          "Блажен, кто вовремя созрел."))
=> ("смолоду" "вовремя")
```

4. Оборудование ПЭВМ студента

Процессор Intel© Celeron© CPU @ 2.16GHz x 2, память: 4096Mb, разрядность системы: 64.

5. Программное обеспечение ЭВМ студента

OS Linux Mint 19.3 Cinnamon, среда SLIME 2.24 с реализацией языка SBCL 1.4.5.debian

6. Идея, метод, алгоритм

За основу взята демонстрационная функция `word-list`, взятая с портала *lisp.ystok.ru*, ссылка на которую была приведена в тексте моего задания. Позже она была переименована в `word-list-string`. Эта функция разбиваает переданную ей строку на слова, которые находятся между разделяющими символами. Проверку символа на разделитель делает `split-char-p` - расширенный знаками препинания аналог функции `whitespace-char-p` с того же самого примера с портала.

Так как текст - список строк, я написал функцию `word-list-text`, которая работает рекурсивно: добавляет результат выполнения `word-list-string` с головой поданного текста к результату выполнения `word-list-text` с хвостом текста. При этом добавляются только те слова, которые ещё не присутствуют в результирующем списке. За это отвечает функция `unique-concat` - модификация функции соединения двух списков из второй лабораторной, разница с которой заключается в уникальности получаемых элементов за счет проверки на вхождение добавляемого элемента посредством метода `member` с параметром *:test 'string=*, необходимым для корректного сравнения срок.

Основная функция `text-longest-words` сначала получает список уникальных слов текста при помощи `word-list-text`, поле чего определяет максимальную длину слова, вызывая функцию `max-len`, которая с помощью функции `max-len-search` рекурсивно сравнивает длины элементов списка с текущим максимальным значением, которое изначально равно 0. После нахождение максимальной длины слова вызывается функция `words-by-len`, которая возвращает список слов заданной длины, что и является искомым результатом.

7. Сценарий выполнения работы

8. Распечатка программы и её результаты

8.1. Исходный код

```
;;; Символы, по которым будет проводитя разбиение слов
(defun split-char-p (char)
  (member char '(#\Space #\Tab #\Newline
                 #\, #\. #\- #\; #\? #\!
                 #\: #\' #\")))

(defun word-list-string (string)
  ;; Разбить строки на слова, разделённые знаками whitespace
  ;; A la (split-seq-if #'whitespace-char-p string)
  (loop with len = (length string)
```

```
        for left = 0 then (1+ right)
        for right = (or (position-if #'split-char-p string
                                     :start left)
                        len)
        unless (= right left) ; исключить пустые слова
          collect (subseq string left right)
        while (< right len)))

;;; возвращает список уникальных строкэлементов- двух списков
(defun unique-concat (set1 set2)
    (if (null set1)
        set2
        (if (member (car set1) set2 :test #'string=)
            (unique-concat (cdr set1) set2)
            (unique-concat (cdr set1) (cons (car set1) set2)))))

;;; разбивает текст на слова и возвращает список уникальных
(defun word-list-text (text)
    (if text
        (unique-concat
            (word-list-string (car text))
            (word-list-text (cdr text)))))


;;; функция поиска максимальной длины слова и списка слов
(defun max-len-search (wset waslen)
    (if wset
        (if (> (length (car wset)) waslen)
            (max-len-search (cdr wset) (length (car wset)))
            (max-len-search (cdr wset) waslen))
        waslen))

;;; обертка для функции, вызывающая основную с параметром 0
(defun max-len (wset) (max-len-search wset 0))

;;; функция поиска слов заданной длины в списке слов
(defun words-by-len (wset len)
    (if wset
        (if (= (length (car wset)) len)
            (cons (car wset)
                (words-by-len (cdr wset) len))
            (words-by-len (cdr wset) len))))
```

```
;;; Основная функция, требуемая по заданию
;;; Берет список уникальных слов текста и возвращает слова,
;;; длина которых равна максиальной длине слова из списка
(defun text-longest-words (text)
   (let ((wset (word-list-text text)))
      (words-by-len wset (max-len wset))))
```

8.2. Результаты работы

```
* (base) max@max-Lenovo-B50-30:~/FuncProg/lab4$ sbcl
This is SBCL 1.4.5.debian, an implementation of ANSI Common Lisp.
More information about SBCL is available at <http://www.sbcl.org/>.

SBCL is free software, provided as is, with absolutely no warranty.
It is mostly in the public domain; some portions are provided under
BSD-style licenses.  See the CREDITS and COPYING files in the
distribution for more information.
* (load "main.lisp")

T
* (text-longest-words
        '("Блажен, кто смолоду был молод,"
          "Блажен, кто вовремя созрел."))

("смолоду" "вовремя")
* (text-longest-words '())

NIL
* (text-longest-words '("Люблю грозу в начале мая...."))

("начале")
* (text-longest-words '("Я памятник себе воздвиг нерукотворный,"
"К нему не зарастет народная тропа,"
"Вознесся выше он главою непокорной"
"Александрийского столпа."
""
"Нет, весь я не умру - душа в заветной лире"
"Мой прах переживет и тленья убежит -"
"И славен буду я, доколь в подлунном мире"
"Жив будет хоть один пиит."))
```

("Александрийского")
* (text−longest−words '(""))

NIL
* (text−longest−words '("oneword"))

("oneword")

9. Дневник отладки

Дата	Событие	Действие по исправлению	Примечание

10. Замечания автора по существу работы

Выполнение проверки на уникальность добавляемых элементов не в `word-list-text`, а в `max-len-search` не улучшит общей асимптотики времени выполнения программы, однако способно значительно улучшить её реальное время выполнения.

11. Выводы

Строки - тип данных, без котрого не может обойтись ни один язык программирования, однако нет единой очки зрения о том, как лучше всего их реализовать и представить. Например, в *Python* - это неизменяемая структура данных с огромным функционалом, когда как в родственном ему *C++* строка - вектор символов, который можно изменить, не создавая новый экземпляр. В процессе выполнения работы я познакомился со строками в *Common Lisp*, узнал как с ними обращаться и даже написал простейшую программу с использованием стандартных функций работы с ними.

Считаем, что встроенная функция `cons` работает за константное время. Тогда программа выполняется за время $O(n^2)$, поскольку функция `text-longest-words` последовательно вызывает в себе функцию `word-list-text` за $O(n^2)$ и функции `max-len`, `words-by-len` с линейным временем выполнения. Линейность `max-len` и `words-by-len` обусловлена тем, что они один раз проходят по списку, выполняя константные действия на каждом шаге. Тепер рассмотрим `word-list-text`: она делает k шагов производя разбиение строки размера m_i на сиок слов на i-ом шаге за время $O(m_i)$, при этом $\sum_{i=1}^{k} m_i = n$. После разбиения происходит m_i константных добавлений слов из списка с проверкой на уникальность за $O(n)$ для каждого добавления в общем случае. Следовательно, время выполнения `word-list-text`: $\sum_{i=1}^{k}(O(nm_i)+O(m_i)) = \sum_{i=1}^{k} O(nm_i) = O(n^2)$, а значит и вся программа работает за квадратичное время.