Отчет по лабораторной работе № 1
по курсу «Функциональное программирование»

Студент группы М8О-307-19 МАИ *Ежов Никита Павлович*, №9 по списку
Контакты: nikita.ejov2012@yandex.ru
Работа выполнена: 30.03.2022

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

1. Тема работы

Примитивные функции и особые операторы Common Lisp.

2. Цель работы

Научиться вводить S-выражения в Lisp-систему, определять переменные и функции, работать с условными операторами, работать с числами, используя схему линейной и древовидной рекурсии.

3. Задание (вариант № 1.29)

Реализовать на языке Common Lisp программу для вычисления значения заданной функции с помощью линейно-рекурсивного процесса. Оценить требуемые время вычисления и оперативную память. Функция f задаётся правилом:

$$f(n) = \begin{cases} 1 & n < 3 \\ f(n) = f(n-1) \cdot (f(n-2) + 2) \cdot (f(n-3) + 3) & n \geq 3 \end{cases} \tag{1}$$

4. Оборудование студента

Процессор Intel i7-4770 (8) @ 3.9GHz, память: 16 Gb, разрядность системы: 64.

5. Программное обеспечение

ОС Kubuntu 20.04.4 LTS, комилятор GNU CLISP 2.49.92, текстовый редактор Atom 1.58.0

6. Идея, метод, алгоритм

Рассмотрим случай, когда $n > 3$:

$$f(n) = f(n-1) \cdot (f(n-2) + 2) \cdot (f(n-3) + 3)$$

.

Введём условные обозначения:

$$y1 = f(n-1)$$

$$y2 = f(n-2)$$

$$y3 = f(n-3)$$

Посчитаем значение $f(n)$. Т.к. первое n, для которого значение функции $\neq 1$ это 3, распишем $f(3)$:

$$f(3) = f(2) \cdot (f(1) + 2) \cdot (f(0) + 3)$$

Теперь распишем то же самое для $f(4)$:

$$f(3) = f(3) \cdot (f(2) + 2) \cdot (f(1) + 3)$$

Как можно заметить, если мы идём снизу вверх, на шаге $n+1$ мы можем использовать аргументы и полученное значение из шага n, приравнивая переменные следующим образом:

$$y1 = (y1 \cdot (y2 + 2) \cdot (y3 + 3))$$

$$y2 = y1$$

$$y3 = y2$$

Таким образом, мы можем написать функцию, в которую на каждый шаг будем передавать обновленные значения $y1$ $y2$ $y3$, а в качестве результата ожидать набор конечных y.
Когда $n = arg$, останавливаем вычисление новых значений y и передаём в качестве ответа для вычисления конечного результата в главной функции. Почему не считать результат сразу в вспомогательной функции? Я хотел посмотреть, как работает multiple-value-binding.

Оценка сложности алгоритма: $O(n)$, т.к. используется линейная рекурсия.
Оценка по памяти: $O(n)$, т.к. при новом рекурсивном вызове мы копируем константное кол-во аргументов n кол-во раз.

7. Сценарий выполнения работы

8. Распечатка программы и её результаты

8.1. Исходный код

```
; 1.29

(defun funcHelper (y1 y2 y3 n res)
  (if (= n res)
      (values y1 y2 y3)
      (funcHelper (* y1 (+ y2 2) (+ y3 3)) y1 y2 (+ n 1) res)))

(defun func (n)
  (if (< n 3)
      1
      (multiple-value-bind (y1 y2 y3) (funcHelper 1 1 1 3 n)
       (* y1 (+ y2 2) (+ y3 3)))))
```

8.2. Результаты работы

```
;; Загружается файл lab1.lisp ...
;; Загружен файл lab1.lisp
[1]> (func 2)
1
[2]> (func 3)
12
[3]> (func 4)
144
[4]> (func 6)
17660160
[5]> (func 5)
8064
```

9. Дневник отладки

Дата	Событие	Действие по исправлению	Примечание

10. Замечания автора по существу работы

На практике сначала казалось, что выполнить эту работу при помощи циклов было бы гораздо проще, но использование линейной рекурсии позволило "забыть"о таких вещах, как, например, присвоение новых значений переменных после каждого шага.

11. Выводы

Я познакомился с синтаксисом языка Common Lisp. Было непривычно и сложно правильно расставить скобки, что и было основной трудностью.

Составленная программа работает за линейное время и использует линейное же количество памяти.