Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное учреждение высшего образования

«Новгородский государственный университет имени Ярослава Мудрого»

Институт «Электронных и информационных систем»

Кафедра «Информационных технология и систем»

Лабораторная работа №3

**«МЕТОДЫ РАЗРАБОТКИ ФУНКЦИОНАЛЬНЫХ ПРОГРАММ»**

по дисциплине:

«Функциональное и логическое программирование»

**Отчёт**

Принял преподаватель:

\_\_\_\_\_\_\_\_\_\_ / Михайлов Д.В

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

Выполнил студент группы 8091:

\_\_\_\_\_\_\_\_\_\_ / Лехновский А. Д.

«\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 2021 г.

**Великий Новгород**

**2021**

1. **Цель и задачи данной лабораторной работы**

Целью работы является изучение основных методов разработки функциональных программ с позиций строго функционального языка.

Основные задачи:

* Освоить приемы нисходящего и восходящего проектирования функциональных программ;
* Научиться выделять основные и вспомогательные функции с учетом разбиения задачи на подзадачи;
* овладеть приемами использования накапливающих параметров во вспомогательных функциях;
* ознакомиться с упреждающим использованием результата вызова функции.

1. **Задания на лабораторную работу**

**Задача №1:**

Написать программу сортировки списка методом Шелла. Вычисление последовательности шагов сортировки производится в соответствии с методом, предложенным Дональдом Кнутом.

**Задача №2:**

Написать программу сортировки методом пузырька. Сравнить эффективность реализованной сортировки и сортировки Шелла реализованной в задаче №1.

**Задача №3:**

Написать программу объединения двух отсортированных списков в один. При этом порядок сортировки в списке-результате должен сохраняться.

**Задача №4:**

Написать программу, выполняющую следующее задание:

*Заданы глубина подсписка, позиция и s-выражение. Включить s-выражение во все имеющиеся подсписки заданной глубины и на заданную позицию.*

1. **Решение поставленных задач**

**Задача №1:**

Идея сортировки методом Шелла состоит в том, чтобы сортировать элементы отстоящие друг от друга на некотором расстоянии step. Затем сортировка повторяется при меньших значениях step, и в конце процесс сортировки Шелла завершается при step = 1 (а именно обычной сортировкой вставками).

Каждый проход в алгоритме характеризуется смещением  , таким, что сортируются элементы отстающие друг от друга на  позиций. Шелл предлагал использовать . Возможны и другие смещения, но  всегда.

* Начало.
* **Шаг 0.** .
* **Шаг 1.** Разобьем массив на списки элементов, отстающих друг от друга на . Таких списков будет .
* **Шаг 2.** Отсортируем элементы каждого списка сортировкой вставками.
* **Шаг 3.** Объединим списки обратно в массив. Уменьшим . Если  неотрицательно — вернемся к шагу 1
* Конец.

Количество шагов сортировки (разделение на группы) и шаги (расстояния) в данной программе вычисляются методом, предложенным Дональдом Кнутом.

Количество шагов сортировки вычисляется по формуле:

где – количество элементов в списке.

В программе параметр высчитывает функция knuth-number-step.

Смещение или шаг вычисляется по формуле:

В программе параметр определяется в функции knuth-sorting-step.

Первоначально расстояние у нас равно . На первом шаге каждая группа включает в себя элементы, расположенные друг от друга на расстоянии . Эти элементы сравниваются между собой, и, в случае необходимости, меняются местами. На последующих шагах также происходят проверка и обмен, но расстояние сокращается. Постепенно расстояние между элементами уменьшается, и на проход по массиву происходит в последний раз.

В случаях, когда список пустой или список содержит 1 элемент, нам не требуется высчитывать шаг и количество шагов, потому что такой список уже является отсортированным.

Код программы для решения задачи приведен в *Приложении 1*:

****Пример работы программы:

*Рисунок №1. Результаты работы программы,*

*при вызове (shell-sort '(21 0 12 7 8 33 -8 77 212 35 41 7 1 4 1 22 6 9 8 3 1 2))*

|  |  |
| --- | --- |
| *Входные данные* | *Результат, полученный с помощью программы* |
| (shell-sort '(21 0 12 7 8 33 -8 77 212 35 41 7 1 4 1 22 6 9 8 3 1 2)) | (-8 0 1 1 1 2 3 4 6 7 7 8 8 9 12 21 22 33 35 41 77 212) |
| (shell-sort '(55)) | (55) |
| (shell-sort '()) | () |
| (shell-sort '(13 7)) | (7 13) |
| (shell-sort '(21 24)) | (21 24) |
| (shell-sort '(77 8 36)) | (8 36 77) |

*Таблица №1. Тестовые наборы данных.*

**Задача №2:**

**Сортировка простыми обменами**, **сортировка пузырьком** (англ. *bubble sort*) — простой в реализации и малоэффективный алгоритм сортировки. Алгоритм состоит в повторяющихся проходах по сортируемому списку. На каждой итерации последовательно сравниваются соседние элементы, и, если порядок в паре неверный, то элементы меняют местами. За каждый проход по списку как минимум один элемент встает на свое место, поэтому необходимо совершить не более проходов, чем размер списка, чтобы отсортировать список.

Код программы для решения задачи приведен в *Приложении 2*:

Пример работы программы:

****

*Рисунок №2. Результаты работы программы,*

*при вызове (bubble\_sort '(10 5 24 8 11 15 123 0 67 35 5 1 11 2000) )*

|  |  |
| --- | --- |
| *Входные данные* | *Результат, полученный с помощью программы* |
| (bubble\_sort '(10 5 24 8 11 15 123 0 67 35 5 1 11 2000) ) | (0 1 5 5 8 10 11 11 15 24 35 67 123 2000) |
| (bubble\_sort '() ) | () |
| (bubble\_sort '(100) ) | (100) |
| (bubble\_sort '(100 55 27) ) | (27 55 100) |
| (bubble\_sort '(34 55) ) | (34 55) |
| (bubble\_sort '(55 34) ) | (34 55) |

*Таблица №2. Тестовые наборы данных.*

Эффективность сортировки пузырьком**:**

Эффективность сортировки Шелла, которая в качестве вычисления последовательности шагов использует метод, предложенный Дональдом Кнутом:

В сравнение этих двух алгоритмов по вычислительной сложности, более эффективным оказался метод сортировки Шелла, которая в качестве вычисления последовательности шагов использует метод, предложенный Дональдом Кнутом. Метод пузырька гораздо менее эффективен других алгоритмов, однако он имеет простую и понятную реализацию. Кроме того, пузырьковая сортировка может использоваться для работы с небольшими массивами данных.

**Задача №3:**

Данная программа имеет следующий алгоритм. На вход поступают два уже отсортированных списка, если один из списков оказывается пустым, то на выходе мы получаем другой список. В случае, когда списки не пустые, мы начинаем сравнивать головы списков. Затем меньшее значение головы списка записывается в результирующий список, после чего рекурсивно вызывается наша исходная функция, в которую теперь будет передаваться уже хвост списка, у которого до этого значение головы оказалось наименьшим. Рекурсия заканчивается, когда один из списков становится пустым.

Код программы для решения задачи приведен в *Приложении 3*:

Пример работы программы:



*Рисунок №3. Результаты работы программы,*

*при вызове (insert-ordered '(1 2 4 7 15) '(0 1 2 3 4 5 6 7 11 32))*

|  |  |
| --- | --- |
| *Входные данные* | *Результат, полученный с помощью программы* |
| (insert-ordered '(1 2 4 7 15) '(0 1 2 3 4 5 6 7 11 32)) | (0 1 1 2 2 3 4 4 5 6 7 7 11 15 32) |
| (insert-ordered '() '(1 2 3)) | (1 2 3) |
| (insert-ordered '() '()) | () |
| (insert-ordered '(222) '( 4 8 10 12 24 38)) | (4 8 10 12 24 38 222) |
| (insert-ordered '(3 7 9 11 33 55 321) '(777)) | (3 7 9 11 33 55 321 777) |
| (insert-ordered '(22 67 128 512) '()) | (22 67 128 512) |

*Таблица №3. Тестовые наборы данных.*

**Задача №4:**

В данной программе у нас имеются две функции: главная (task) и второстепенная (ins-in-pos). Второстепенная функция имеет три аргумента – это список, куда мы хотим вставить, вставка (наше s-выражение) и номер позиции, куда хотим вставить (индексация с нуля). Эта функция используется только тогда, когда мы дошли до нужной глубины списка и уже списке соответственной глубины производится вставка нашего s-выражения в нужную позицию. Кстати, если номер позиции в списке, который вы укажете больше самого списка, то вставка произойдёт в конец списка заданной глубины. В главной функции используется функция map, которая применяет функцию к каждому элементу списка по одному, так мы работаем с каждым элементом исходного списка, а затем через функцию atom? проверяем глубину подсписков в списке. Исходный список – это нулевая глубина.

Код программы для решения задачи приведен в *Приложении 4*:

Пример работы программы:



*Рисунок №4. Результаты работы программы,*

*при вызове (task '(1 (a v c) 4 (z x c) 3) 1 2 '(GGG))*

|  |  |
| --- | --- |
| *Входные данные* | *Результат, полученный с помощью программы* |
| (task '(1 (a v c) 4 (z x c) 3) 1 2 '(GGG)) | (1 (a v (GGG) c) 4 (z x (GGG) c) 3) |
| (task '(a b (c d (e f) ((g h))) i (((j)))) 0 1 '(\* \*)) | (a (\* \*) b (c d (e f) ((g h))) i (((j)))) |
| (task '(a b (c d (e f) ((g h))) i (((j)))) 1 1 '(\* \*)) | (a b (c (\* \*) d (e f) ((g h))) i (((j)) (\* \*))) |
| (task '(a b (c d (e f) ((g h))) i (((j)))) 1 5 '(\* \*)) | (a b (c d (e f) ((g h)) (\* \*)) i (((j)) (\* \*))) |
| (task '(a b (c d (e f) ((g h))) i (((j)))) 1 4 '(\* \*)) | (a b (c d (e f) ((g h)) (\* \*)) i (((j)) (\* \*))) |
| (task '(a b (c d (e f) ((g h))) i (((j)))) 4 1 '(\* \*)) | (a b (c d (e f) ((g h))) i (((j)))) |
| (task '(a b (c d (e f) ((g h))) i (((j)))) 3 1 '(\* \*)) | (a b (c d (e f) ((g (\* \*) h))) i (((j (\* \*))))) |

*Таблица №4. Тестовые наборы данных.*

1. **Вывод**

В данной лабораторной работе были разработаны функциональные программы с позиции строго функционального языка: сортировка Шелла, сортировка пузырьком, объединение двух отсортированных списков, а также включение s-выражения во все имеющиеся подсписки заданной глубины и на заданную позицию.

**Приложение 1**

Код программы:

;Сортировка Шелла с использованием метода Кнута

(define (shell-sort lst)

(if (< (length lst) 1)

lst

(shell-sort-step lst (knuth-number-steps lst))

)

)

(define (knuth-number-steps lst)

(- (log (length lst) 3) 1)

)

(define (shell-sort-step lst step)

(if (and (= step -1) (!= (length lst) 2))

lst

(if (= (length lst) 2)

(if (< (nth 1 lst) (nth 0 lst))

(list (nth 1 lst) (nth 0 lst))

lst

)

(sort-the-list step lst (knuth-sorting-step '() step) (knuth-sorting-step '() step) (knuth-sorting-step '() step))

)

)

)

(define (knuth-sorting-step lst n)

(cond

( (= (length lst) 0) (knuth-sorting-step '(1) n) )

( (> (length lst) n) (nth n lst) )

( (knuth-sorting-step(flat(list lst (+ (\* 3 (nth (- (length lst) 1) lst)) 1))) n) )

)

)

(define(sort-the-list step lst stepValue start currentIndex)

(cond

((= start (length lst)) (shell-sort-step lst (- step 1)))

((< (- currentIndex stepValue) 0) (sort-the-list step lst stepValue (+ start 1) (+ start 1)))

((< (nth currentIndex lst) (nth (- currentIndex stepValue) lst))

(sort-the-list step (swapping lst (- currentIndex stepValue) currentIndex) stepValue start (- currentIndex stepValue))

)

((sort-the-list step lst stepValue (+ start 1) (+ start 1)))

)

)

(define (swapping lst index1 index2)

(flat (list (test lst '() 0 index1) (nth index2 lst) (test lst '() (+ index1 1) index2) (nth index1 lst) (test lst '() (+ index2 1) (length lst))) )

)

(define (test lst newlst index1 index2)

(cond

((= index1 index2) newlst)

( true (test lst (add-element-list newlst (nth index1 lst)) (+ index1 1) index2))

)

)

(define (add-element-list lst element)

(cons lst element)

)

(shell-sort '(21 0 12 7 8 33 -8 77 212 35 41 7 1 4 1 22 6 9 8 3 1 2))

(shell-sort '(55))

(shell-sort '())

(shell-sort '(13 7))

(shell-sort '(21 24))

(shell-sort '(77 8 36))

**Приложение 2**

Код программы:

;Сортировка Пузырьком

(define (bubble\_sort newlst oldlst)

(if (= newlst oldlst)

newlst

(bubble\_sort (roundd newlst 0) newlst)

)

)

(define (roundd lst i)

(if (< (+ i 1) (length lst))

(roundd (swapp lst i) (+ i 1))

lst

)

)

(define (swapp lst j)

(if (< (nth (+ j 1) lst) (nth j lst))

(swapping lst j (+ j 1))

lst

)

)

(define (swapping lst index1 index2)

(flat (list (test lst '() 0 index1) (nth index2 lst) (test lst '() (+ index1 1) index2) (nth index1 lst) (test lst '() (+ index2 1) (length lst))) )

)

(define (test lst newlst index1 index2)

(cond

((= index1 index2) newlst)

( true (test lst (add-element-list newlst (nth index1 lst)) (+ index1 1) index2))

)

)

(define (add-element-list lst element)

(cons lst element)

)

(bubble\_sort '(10 5 24 8 11 15 123 0 67 35 5 1 11 2000) )

(bubble\_sort '() )

(bubble\_sort '(100) )

(bubble\_sort '(100 55 27) )

(bubble\_sort '(34 55) )

(bubble\_sort '(55 34) )

**Приложение 3**

Код программы:

;Программа объединения двух отсортированных списков в один.

;При этом порядок сортировки в списке-результате должен сохраняться.

(define (insert-ordered lst\_1 lst\_2)

(cond ( (null? lst\_1) lst\_2)

( (null? lst\_2) lst\_1)

(

(< (first lst\_1) (first lst\_2))

(cons (first lst\_1) (insert-ordered (rest lst\_1) lst\_2))

)

( true (cons (first lst\_2) (insert-ordered lst\_1 (rest lst\_2))) )

)

)

(insert-ordered '(1 2 4 7 15) '(0 1 2 3 4 5 6 7 11 32))

(insert-ordered '() '(1 2 3))

(insert-ordered '() '())

(insert-ordered '(222) '( 4 8 10 12 24 38))

(insert-ordered '(3 7 9 11 33 55 321) '(777))

(insert-ordered '(22 67 128 512) '())

**Приложение 4**

Код программы:

;Заданы глубина подсписка, позиция и s-выражение.

;Включить s-выражение во все имеющиеся подсписки заданной глубины и на заданную позицию.

;; Вставить в список lst в позицию n

(define (ins-in-pos lst v n)

(cond ( (null? lst) (list v) )

( (= n 0) (cons v lst))

(true (cons (first lst) (ins-in-pos (rest lst) v (- n 1))))

)

)

;(ins-in-pos '(1 2 3 4 5) '(u u) 3)

;(ins-in-pos '(1 2 3 4 5) '(u u) 5)

;(ins-in-pos '(1 2 3 4 5) '(u u) 14)

(define (task lst lv p v)

(if (> lv 0)

(map

(lambda (x)

(if (atom? x)

x

(task x (- lv 1) p v)

)

) lst

)

(ins-in-pos lst v p)

)

)

(task '(1 (a v c) 4 (z x c) 3) 1 2 '(GGG))

(task '(a b (c d (e f) ((g h))) i (((j)))) 0 1 '(\* \*))

(task '(a b (c d (e f) ((g h))) i (((j)))) 1 1 '(\* \*))

(task '(a b (c d (e f) ((g h))) i (((j)))) 1 5 '(\* \*))

(task '(a b (c d (e f) ((g h))) i (((j)))) 1 4 '(\* \*))

(task '(a b (c d (e f) ((g h))) i (((j)))) 4 1 '(\* \*))

(task '(a b (c d (e f) ((g h))) i (((j)))) 3 1 '(\* \*))