МІНІСТЕРСТВО ОСВІТИ І НАУКИ, МОЛОДІ ТА СПОРТУ УКРАЇНИ
ТАВРІЙСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
імені В. І. ВЕРНАДСЬКОГО

Кафедра нової та новітньої історії

С. П. ГОРЛЯНСЬКИЙ

РОЗВ'ЯЗУВАННЯ ЗАДАЧ МОВОЮ ПРОГРАМУВАННЯ ЛІСП

Навчально-методичний посібник з дисципліни «Комп'ютерні технології та історична наука»

для студентів 5 курсу денної форми навчання та 6 курсу заочної форми навчання спеціальностей – 8.030301 – «історія» освітньо-кваліфікаційного рівня «магістр» і 7.030301 – «історія» освітньо-кваліфікаційного рівня «спеціаліст» професійного напряму підготовки 0203 «Гуманітарні науки»

Сімферополь 2014

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ,
МОЛОДЕЖИ И СПОРТА УКРАИНЫ
ТАВРИЧЕСКИЙ НАЦИОНАЛЬНЫЙ УНИВЕРСИТЕТ
имени В. И. ВЕРНАДСКОГО

Кафедра новой и новейшей истории

С. П. ГОРЛЯНСКИЙ

РЕШЕНИЕ ЗАДАЧ
НА ЯЗЫКЕ ПРОГРАММИРОВАНИЯ ЛИСП

**Учебно-методическое пособие по дисциплине
«Компьютерные технологии и историческая наука»**

для студентов 5 курса дневной формы обучения и 6 курса заочной формы обучения специальностей – 8.030301 – «история» образовательно-квалификационного уровня «магистр» и 7.030301 – «история» образовательно-квалификационного уровня «специалист» профессионального направления подготовки 0203 «Гуманитарные науки»

Симферополь 2014

Рекомендовано к печати заседанием кафедры от 30 ноября 2011 г., протокол № 5.

Рекомендовано к печати учебно-методическим советом Таврического национального университета имени В. И. Вернадского от 22 декабря 2011 г., протокол № 2.

Рецензент:
В. И. Донской, доктор физико-математических наук, профессор, заслуженный деятель науки и техники Украины.

Горлянский С. П.
Решение задач на языке программирования Лисп. - Симферополь: Таврический национальный университет имени В. И. Вернадского, 2014 – 540 с.

Данное пособие является частью мультимедийной системы подготовки студентов, обучающихся по специальности «история», составлено в соответствии с программой курса «Компьютерные технологии и историческая наука», содержит варианты решений учебных задач на языке программирования Лисп (ANSI Common Lisp) и предназначено для промежуточного контроля знаний и навыков, а также самопроверки и самоподготовки студентов-историков.

Оглавление

Предисловие

> ...ум наш утратил известную долю своей крепости, восприимчивости и гибкости: обязательная специализация развила в нем односторонность и парализовала его восприимчивость в других областях.
>
> Ипполит Тэн «Наполеон Бонапарт»
>
> ...учителя должны готовить студентов к их будущему, а не к своему прошлому.
>
> Ричард Хэмминг «Искусство науки»

Наблюдаемая в начале XXI века смена технологических подходов в научных исследованиях и образовании с неизбежностью приводит не только к количественным, но и качественным изменениям в структуре и содержании подготовки студентов, обучающихся по специальности «история».

Во-первых, универсальный характер современного университетского образования все в большей степени предполагает освоение студентами-историками компьютерных технологий для сбора, хранения, классификации, анализа и представления исторической информации, реконструкции исторических событий. Во-вторых, последнее десятилетие привело также к росту использования издательских возможностей компьютерных систем в оформлении результатов студенческих исследований и повсеместному переходу студентов к компьютерному набору текстов курсовых, дипломных и магистерских работ.

Условием успешности освоения историками современных компьютерных технологий являются знания, умения и навыки, приобретенные ими в ходе изучения логики, информатики, латинского языка и современных иностранных языков. Вместе с тем, важной составляющей подготовки студентов-историков наряду со знанием языков, умением пользоваться текстовыми процессорами, электронными таблицами и базами данных, является понимание того, как составляются и выполняются компьютерные

программы, а также овладение навыками разработки алгоритмов и решения задач с применением компьютерных технологий.

Программирование можно представить как беседу с компьютером: вы говорите команды, инструкции, запросы и ждете, что он скажет в ответ. Чтобы быть понятым компьютером, необходимо говорить с ним на его языке. В мире насчитывается 6912 живых языков и 8512 языков программирования, мы познакомимся с одним из самых элегантных языков – Лиспом, который был создан выдающимся американским информатиком, основоположником функционального программирования и автором термина «искусственный интеллект» профессором Стэнфордского университета Джоном Маккарти в 1958 году. Лисп является вторым по возрасту (после Фортрана) прошедшим проверку временем и используемым языком программирования высокого уровня.

Критериями выбора языка для обучения решению задач были избраны гибкость и простота (элегантность). Существуют и более лаконичные языки (K, J, Python, Haskell), но предлагаемый ANSI Common Lisp является одним из немногих языков "стройного замысла", как и Smalltalk. Синтаксис языка настолько прост для изучения, что позволяет многим заявлять, что Лисп не имеет синтаксиса.

Для того чтобы объяснить синтаксис языка, достаточно одного абзаца. Все данные в Лиспе могут быть представлены в виде списка, заключенного в круглые скобки '(a b c d). Если перед открывающей скобкой убрать апостроф, то список (a b c d) будет выполнен как функция a с параметрами b, c, d. Привычная математическая запись сложения (2 + 3) представляется как (+ 2 3).

Следует заметить, что быстрое знакомство с синтаксисом избавит вас от необходимости припоминать правила и очередность выполнения функций, но этого будет недостаточно для решения задач по программированию на Лиспе. Также, как «одно изучение правил передвижения шахматных фигур не сделает вас гроссмейстером» (Хал Абельсон). Для того чтобы писать программы необходимо познакомиться со многими приемами и сотнями встроенных функций.

Существует простой и единственный быстрый способ научиться программированию – улучшать готовые решения и писать собственные программы. Данное пособие составлено с целью помочь вам приобрести вкус к решению задач и научиться программировать на Лиспе.

Установка программного обеспечения

Инструменты, которыми мы пользуемся, оказывают тонкое и глубокое влияние на наши способы мышления, а, следовательно, и на нашу способность мыслить.

Эдсгер Дейкстра

Книги наши толще, а вера наша полнее…
Николай Лесков «Сказ о тульском косом левше и о стальной блохе»

Выгоды от использования Лиспа заключаются в переживаниях и впечатлениях от использования Лиспа.

Питер Зибель «Практичный Common Lisp»

Чтобы стать кузнецом надо ковать.

Французская пословица

Следуй своему блаженству.

Джозеф Кэмпбелл «Власть мифа»

Искусство программирования заключается в формулировании пошаговых инструкций. Примером программы является последовательность шагов по установке программного обеспечения и решению следующей задачи – определению функции, которая удваивает число, посланное ей в качестве аргумента. Пример решения на Лиспе:

Далее представлено решение для Windows*:

1. Скачать LispBox: http://www.common-lisp.net/project/lispbox/test_builds/lispbox-0.7-ccl-1.6-windowsx86.zip

---

* для OS X неоходимо установить: http://common-lisp.net/project/lispbox/test_builds/Lispbox-0.7-with-ccl-1.6-darwinx86.zip

**для Linux 32-bit установить: http://common-lisp.net/project/lispbox/test_builds/lispbox-0.7-ccl-1.6-linuxx86.tar.gz, для Linux 64-bit: http://common-lisp.net/project/lispbox/test_builds/lispbox-0.7-ccl-1.6-linuxx86-64.tar.gz (Версия Linux требует GTK2 и некоторые другие общие библиотеки, и имеет проблемы с glibc на некоторых Linux дистрибутивах), а также скрипты для автоматизации загрузки, компиляции и настройки средств разработки для Лиспа: https://github.com/LinkFly/lisp-dev-tools

2. Разархивировать lispbox-0.7-ccl-1.6-windowsx86.zip и выполнить C:\Program Files (x86)\LispBox\lispbox.bat (в старых версиях C:\Program Files (x86)\LispBox\RunLispBox.bat).

или:

1. Скачать Lisp Cabinet: http://sourceforge.net/projects/lispcabinet/files/Lisp%20Cabinet%200.3.4/LispCabinetSetup-0.3.4.exe/download

2. Выполнить C:\Program Files (x86)\LispCabinet\LispCabinet.exe, затем нажать на клавиатуре клавишу ALT и, не отпуская ее, нажать клавишу X, набрать slime и нажать клавишу ENTER.

далее:

3. Перейти к вводу имени файла (поиску): CTRL-X CTRL-F

4. Ввести имя файла, например double.lisp (программы в Лиспе хранятся в файлах с расширением .lisp), и нажать ENTER (если в папке C:\LispBox\ нет файла с таким именем, то он будет создан C:\LispBox\double.lisp)

5. Набрать программу, которая будет сохранена в файле double.lisp:

```
(defun double (x)
  (* x 2))
```

6. Скомпилировать программу CTRL-C CTRL-K

7. Выйти из сообщения об ошибках Q, исправить ошибки, вернуться к пункту 6.

8. Перейти в REPL (Read Evaluate Print Loop): CTRL-C CTRL-Z

9. После приглашения CL-USER> ввести (double 100) и нажать ENTER

10. Программа выдаст ответ: 200

Дальнейшие шаги: http://lisper.ru/pcl/pcl.pdf

Скачать документацию: ftp://ftp.lispworks.com/pub/software_tools/reference/HyperSpec-7-0.tar.gz

Команды редактора Emacs SLIME

```
                    Vi is the god of editors. Emacs is the edi-
                    tor of gods.
                                                              ~
```

Для того, чтобы работать с редактором Emacs в режиме SLIME (Superior Lisp Interaction Mode for Emacs) необходимо знать несколько клавиатурных комбинаций: открыть файл – c-x c-f, то есть, удерживая ctrl, нажать x, затем удерживая ctrl, нажать f, как сохранить файл – c-x c-s и как выйти из Emacs – c-x c-c (N.B. в краткой записи m – meta означает нажатие клавиши alt)

Файлы

Открыть файл	c-x c-f
Открыть файл в другом окне	c-x 4 c-f
Открыть файл только для чтения	c-x c-r
Сохранить	c-x c-s
Сохранить как	c-x c-w
Отменить	c-g
Выйти	c-x c-c

Буфер

Список буферов	c-x c-b
Следующий буфер	c-x o
Сменить буфер	c-x b
Зарыть окно	c-x 0
Уничтожить буфер	c-x k
Оболочка (shell)	m-x
[ ]	c-x 1
[-]	c-x 2
[|]	c-x 3

Курсор

Вперед на один значок	c-f или ->
Назад на один значок	c-b или <-
Вверх на один значок	c-p
Вниз на один значок	c-n
Вперед на одно слово	e-f

Назад на одно слово	e-b
Начало строки	c-a или home
Конец строки	c-e или end
Предыдущее предложение	m-a
Следующее предложение	m-e
Начало параграфа	m-{
Конец параграфа	m-}
Страницу вверх	m-v
Страницу вниз	c-v

Редактирование

Отменить последние изменения	c-_ или c-/
Копировать выделенный участок	m-w
Вырезать выделенный участок	c-w
Вставить выделенный участок	c-y
Поменять местами значки	c-t
Поменять местами слова	m-t
Поменять местами s-выражения	c-m-t
Поменять местами строки	c-x c-t
Удалить значок	c-d
Удалить до конца слова	m-d
Удалить до конца строки	c-k
Удалить до начала слова	c-bsp
Выделить	c-space или c-@

Поиск

Поиск вперед	c-s
Поиск назад	c-r

Замена

Заменить	m-%

Завершение

Завершить символ	m-/ или c-c c-i или c-m-i
Показать варианты завершения символа	c-c m-i
Заполнить форму	c-c c-s

Закрытие

Закрыть скобки	c-c c-q

Отступы

Создать отступ строки	tab
Создать отступ s-выражения	c-m-q

Компиляция

Компилировать функцию	c-c c-c
Компилировать и загрузить файл	c-c c-k
Переключиться на выводной буфер	c-c c-z

Отладчик

Выход	q

Навигация

Следующая команда в истории repl	m-n
Предыдущая команда в истории repl	m-p

Команды

Сменить директорию	,cd
Выход	quit

Макрос

```
c-x (
(#\ c-f m-spc ' c-e ') c-n c-a
c-x )
```

то есть:

Начать запись макроса	c-x (
Затем вставить символы	(#\
Перейти вперед на один символ	c-f
Удалить все пробелы кроме одного	m-spc
Вставить символ	'
Перейти в конец строки	c-e
Вставить символы	')
Перейти в начало следующей строки	c-n c-a
Завершить запись макроса	c-x)
Выполнить макрос	c-x e
Повторить макрос 4 раза	c-x e e e e

Пакеты

> Although high general intelligence is common among hackers, it is not the sine qua non one might expect. Another trait is probably even more important: the ability to mentally absorb, retain, and reference large amounts of 'meaningless' detail, trusting to later experience to give it context and meaning. A person of merely average analytical intelligence who has this trait can become an effective hacker, but a creative genius who lacks it will swiftly find himself outdistanced by people who routinely upload the contents of thick reference manuals into their brains.
>
> New Hacker's Dictionary

> Programming languages should be designed not by piling feature on top of feature, but by removing the weaknesses and restrictions that make additional features appear necessary.
>
> Robert Findler, Jacob Matthews

> Schemer: Buddha is small, clean, and serious. Lispnik: Buddha is big, has hairy armpits, and laughs.
>
> Nikodemus Siivola

> The limits of my language mean the limits of my world.
>
> Ludwig Wittgenstein

Пакет – это лисповский объект, который связывает имена с символами. Текущий пакет хранится в глобальной переменной *package*. Сразу после старта текущим пакетом будет common-lisp-user.

Создать новый пакет:

```
CL-USER> (defpackage :bar (:use :common-lisp))
#<PACKAGE BAR>
```

Поместить файл в пакет:

```
CL-USER> (in-package :bar)
#<PACKAGE BAR>
BAR>
```

Синтаксис

> With experience we learn that simplicity is king and less is more, but it takes effort.
>
> Gustavo Duarte

> …скобочки обнимают слова, и слова становятся добрее.
>
> ~

Лисп наделен однородным, последовательным и предсказуемым синтаксисом, который описывается всего двумя правилами построения выражений, которые легко запомнить:

программа = список = (атомы и подсписки)

Атомы и списки называются символьными выражениями или s-выражениями (s-expression).

Список – перечень, последовательность, элементами которой являются атомы и/или подсписки. Список всегда начинается с открывающей круглой скобки и заканчивается закрывающей круглой скобкой. Элементы списка разделяются пробелами. Например, (a b c) или (a (b c)).

Таким образом, список – это многоуровневая или иерархическая структура данных, в которой открывающие и закрывающие скобки находятся в строгом соответствии.

Список, в котором нет ни одного элемента, называется пустым списком и обозначается () или символом nil. Он выполняет ту же роль, что и нуль в арифметике.

Список можно использовать для представления знаний:
(German Empire (Chancellor 1871 1890 (Otto Bismarck 1815 1898)))

Одним из основных отличий языка Лисп от традиционных языков программирования является запись в виде списков не только данных, но и программ. Например, список (+ 2 3) можно интерпретировать либо как список, состоящий из трех элементов, где quote, ' или ` блокируют вычисление выражения:

```
> '(+ 2 3)
(+ 2 3)
```

либо как действие (функцию + с аргументами 2 и 3), дающее в результате число 5:

```
> (+ 2 3)
5
```

Выборочно разблокировать список (s-выражение) можно с помощью запятой:

```
> `(,(+ 2 3) '(+ 2 3))
(5 '(+ 2 3))
```

Принятая в Лиспе скобочная префиксная запись является одной из наиболее привлекательных черт этого языка, превращающих программирование в удовольствие.

Общепринятая инфиксная нотация (запись):

```
> (2 + 5)
; Evaluation aborted
```

Синтаксис Лиспа основан на префиксной нотации - структура программы: (функция, аргумент, аргумент ...):

```
> (+ 2 5)
7
```

Префиксная нотация делает Лисп лаконичным языком, например:

```
> (+ 2 3 4 5 6 7); префиксная нотация
27
```

Правила интерпретации списков определяются первым элементом списка, которым является функция.

Синтаксис языка устанавливает простой порядок действий:

```
> (+ 2 (* 3 5))
17
```

Существует и постфиксная нотация, как например, в языке программирования Forth:

```
> 2 5 +
; Evaluation aborted
```

Как вы смогли заметить – язык программирования Лисп синтаксически регулярен и концептуально элегантен (прост): все выражается функцией - даже математические операторы +, -, * и /.

ТИПЫ ДАННЫХ

> Все языки программирования делятся на две большие группы, в одну из них входит Лисп, в другую – все остальные языки программирования.
>
> Хювёнен Э., Сеппянен И. «Мир Лиспа»

> Целью вычислений является понимание, а не числа.
>
> Ричард Хэмминг

> У вас есть единицы? Везучий ублюдок! Всё, что было у нас — это нули.
>
> ~

Чтобы быть уверенным, что Лисп поймет, о чем вы говорите, нужно ссылаться только на тот мир, который ему известен, и использовать понятные ему термины.

Символы

В программировании на языке Лисп используются символы и построенные из них символьные структуры.

Символ – ограниченная пробелами последовательность знаков, другими словами - последовательность непробельных знаков (sembolon - греч. знак, условное обозначение, художественный образ, обозначающий какую-нибудь мысль, идею).

Символ – это имя переменной (функции), состоящее из букв без различия между прописными буквами и строчными, цифр и специальных значков !-@#$%^&*_+/[]{}<>.? за исключением (),’`|”:\; которое обозначает какой-нибудь предмет, объект, вещь, действие из реального мира.

Символы – это имена переменных, у которых есть свойства.

```
> 'history
HISTORY
> 'lisp1958
LISP1958
```

В Лиспе, в отличие от большинства других языков программирования, допустимо начинать имя символа с цифры:

```
> '33heroes
```

```
33HEROES

> 'ansi-common-lisp-1954
ANSI-COMMON-LISP-1954

> 'santa came back
; Evaluation aborted

> '|santa came back|
|santa came back|

> 'january-25-2011-santa-came-back
JANUARY-25-2011-SANTA-CAME-BACK

> 'santa-came-back!-@#~$%^&*_+/[]{}<>.?
SANTA-CAME-BACK!-@#~$%^&*_+/[]{}<>.?

(defun !-@#$%^&*_+/[]{}<>.? (n) (+ n n))

> (!-@#$%^&*_+/[]{}<>.? 200)
400

Запрещено использование знаков (),’`|”:\;

> 'z(
; Evaluation aborted

> 'z)
; Evaluation aborted

> 'z,
; Evaluation aborted

> 'z'
; Evaluation aborted

> 'z`
; Evaluation aborted

> 'z|
; Evaluation aborted

> 'z:
; Evaluation aborted

> 'z"
; Evaluation aborted

> 'z\
|Z
|

> 'z; последний предваряет комментарии.
```

```
Z
```

Корректным именем функции, с точки зрения Лиспа, будет символ: a+a*[a+1]+a*[a+1]*[a+2+...+a*[a+1]*...*[a+1] - см. Задача 5.7.

Перевод строки в символ

```
> (read-from-string "MODERN-AGE")
MODERN-AGE
10
```

Логические значения t и nil

Символы t и nil имеют в Лиспе специальное всегда одно и то же фиксированное значение. Символы t и nil также как и числа являются константами, их нельзя использовать в качестве имен других лисповских объектов: t - истина – да – есть – true, а nil - ложь – нет – пусто – false,

```
> 750
T

> '()
NIL
```

Nil - единственный символ, который является атомом и списком одновременно - обозначает пустой список, подобно Кромвелю, для которого после принятия «Билля о самоотречении» было сделано исключение: он остался в армии, несмотря на то, что был депутатом Палаты общин.

Числа

Число – ограниченная пробелами последовательность знаков. Числа все же не являются полновесными символами, так как не могут представлять иные лисповские объекты кроме самих себя или своего числового значения, т. е. есть числа являются константами.

```
> 5
5
> -100
-100
> 12.4
12.4
> 31/5
31/5
> (float 31/5)
6.2
```

Перевод строки в число

```
> (parse-integer "10")
10
```

Символы и числа представляют те простейшие объекты (атомы), из которых строятся остальные структуры.

Все символы, за исключением чисел и логических значений t и nil, если они не объявлены как константы при помощи директивы defconstant, исполняют роль переменных (variables), которые используются для обозначения других лисповских объектов.

Кроме этого система Common Lisp содержит глобальные специальные переменные, такие как символ pi, представляющий значение числа «пи» 3.141592653589793238 5L0.

Буквы

> …хочу отметить четыре важнейшие черты, общие для математики, музыки и других наук и искусств: первое — красота, второе — простота, третье — точность и четвертое — безумные идеи.
>
> Израиль Гельфанд

Отдельная буква определяется как #\m. С помощью функции char можно как читать, так и писать знаки строки. (char "Oliver Cromwell" n) возвращает n-ый символ строки:

```
> (char "Oliver Cromwell" 10)
#\m
```

Каждой букве соответствует какое-либо простое число:

```
> (char-code #\m)
109

> (code-char 110)
#\n
```

Перевод строки в список букв

```
> (coerce "modern-age" 'list)
(#\m #\o #\d #\e #\r #\n #\- #\a #\g #\e)

> (loop for c across "brett" collect c)
(#\b #\r #\e #\t #\t)
```

```
> (map 'list #'identity "brett")
(#\b #\r #\e #\t #\t)

> (concatenate 'list "brett")
(#\b #\r #\e #\t #\t)
```

Список кодов букв строки

```
> (map 'list #'char-code "brett")
(98 114 101 116 116)
```

Строки

Строки – это векторы (направленные массивы), состоящие из буквенных значков. Для определения строки достаточно заключить набор букв в двойные кавычки:

```
> "modern age"
"modern age"
```

Сравнение строк

```
> (string-equal "modern age" "MODERN-AGE")
T
```

Перевод символа в строку

```
> (string 'modern-age)
"MODERN-AGE"
```

Перевод числа в строку

```
> (write-to-string 1792)
"1792"
```

Перевод списка букв в строку

```
> (coerce '(#\a #\b #\c) 'string)
"abc"
```

Поиск подстроки в строке

```
> (search "b" "abc")
1
```

Присоединение буквы к строке

```
> (concatenate 'string (string #\a) "bc")
"abc"
```

Соединение строк

```
> (concatenate 'string "Bonny and " "Clyde")
"Bonny and Clyde"

> (format nil "~a and ~a" "Bonny" "Clyde")
"Bonny and Clyde"
```

Сортировка знаков строки

```
> (sort "Giuseppe Garibaldi" #'char<)
" GGaabdeeiiilpprsu"
```

Список слов строки

```
> (read-from-string (concatenate 'string "(" "modern age" ")"))
(MODERN AGE)
12
```

Список однобуквенных подстрок строки

```
> (mapcar 'string (coerce "brett" 'list))
("b" "r" "e" "t" "t")

> (map 'list #'string "brett")
("b" "r" "e" "t" "t")

> (loop for c across "brett" collect (string c))
("b" "r" "e" "t" "t")
```

Возврат части строки

```
> (subseq "modern age" 0 6)
"modern"

> (subseq "modern age" 7)
"age"
```

Последовательности

К последовательностям в Лиспе относятся списки и векторы (в том числе и строки).

Индексы векторов, строк и списков начинаются с нуля, так первый по счету элемент имеет индекс 0:

```
> (elt '(a b c d) 0)
A

> (elt '(a b c d) 1)
B

> (nth 1 '(a b c d))
B

> (nthcdr 1 '(a b c d))
(B C D)

> (position #\W "Wellington")
0

> (position #\w "Wellington")
NIL

> (position #\n "Wellington")
5

> (position #\n "Wellington" :start 6)
9

> (position #\n "Wellington" :from-end t)
9

> (position #\a "baobab" :from-end t)
4

> (position-if #'oddp '((1) (2) (3) (4)) :start 1 :key #'car)
2

> (position 595 '())
NIL

(position-if-not #'integerp '(1 2 3 4 5.0))
```

```
4

> (length '(a b c))
3

> (count 'a '(a a b))
2

> (apply #'min '(7 4 6 4 5)) *CLISP до 4096 элементов, иначе – reduce
4
> call-arguments-limit
4096

> (reduce #'max '(7 4 6 4 5))
7

> (search '(b) '(a b c))
1

> (find 'b '(a b c))
B

> (reverse '(c b a))
(A B C)

> (remove 'a '(a b a c))
(B C)

> (remove-if #'evenp '(1 2 3 4 5 6 7))
(1 3 5 7)

> (remove-if-not #'plusp '(-1 0 1 2 3))
(1 2 3)

> (remove-duplicates '(a a a b b c))
(A B C)

> (substitute 'z 'a '(a a a b b c))
(Z Z Z B B C)
```

В следующем разделе мы продолжим знакомиться с функциональным программированием на Лиспе. В отличие от процедурного подхода программирование при помощи функций соответствует следующим правилам:

- программа состоит из вызова функций, то есть функция принимает значение другой функции в качестве параметра;

- значения переменным не присваиваются, а передаются в качестве параметров.

ФУНКЦИИ

> Lisp is lisp, it has no future and no past,
> it neither endorses nor combats competitors
> and it is entirely self sufficient.
>
> ~
>
> My analyst warned me, but metaprogramming
> was so beautiful I got another analyst.
>
> ~

Лисп – это набор функций. Функцией в математике называется отображение (mapping), которое однозначно отображает одни значения на другие. Например, запись y = f (x) ставит в соответствие каждому элементу x единственный элемент y из множества значений функции f. Это соответствие также можно записать в следующем виде f (x) -> y, где функция f от аргумента x имеет значение y = f (x).

Функция — в программировании — один из видов подпрограммы. Особенность, отличающая её от другого вида подпрограмм — процедуры, состоит в том, что функция возвращает значение, а её вызов может использоваться в программе как выражение.

У функции может быть произвольное количество аргументов (в том числе их может не быть совсем, и она будет иметь постоянное значение).

Примером является арифметическая функция сложения:

```
> (+ 2 3)
5
```

Базовые функции

Мы познакомились с функцией сложения, однако, с самого начала Лисп рассматривался в качестве языка для создания искусственного интеллекта и был предназначен для работы с символами и списками.

Основные функции обработки списков

В Лиспе для построения, разбора и анализа списков существуют очень простые базовые функции. Простота базовых функций и их малое число – это характерные черты Лиспа. С ними связана математическая элегантность языка.

Функция создания списка - cons (конструирующая функция):

Функция cons включает новый элемент в начало списка, т.е. второй аргумент обязательно должен быть списком, иначе результатом будет точечная пара (dotted pair) > (cons 'a 'b) -> (A . B):

```
> (cons 'a '())
(A)
> (cons 'a nil)
(A)
> (cons 'a '(b))
(A B)
```

Функции разбора списка - car и cdr (селективные функции):

car – возвращает в качестве значения первый элемент списка (head - голову списка)

```
> (car '(a b c))
A
```

cdr – возвращает в качестве значения список, получаемый из исходного списка после удаления из него первого элемента (tail - хвост списка)

```
> (cdr '(a b c))
(B C)
```

Функции car и cdr являются сокращениями от наименований регистров вычислительной машины IBM 605 Contents of Address Register и Contents of Decrement Register. Автор Лиспа Джон Маккарти реализовал первую Лисп-систему именно на этой машине и использовал регистры car и cdr для хранения головы и хвоста списка. В Common Lisp наряду с именами car и cdr используют и более громоздкие имена first (первый) и rest (остаток).

```
> (first '(a b c))
A
> (rest '(a b c))
(B C)
```

Функции car и cdr определены лишь для аргументов, являющихся списками:

```
> (car 'a)
; Evaluation aborted
> (cdr 'a)
; Evaluation aborted
```

Функции проверки

Функции опознавания и анализа или базовые функции-предикаты, которые проверяют наличие некоторого свойства и возвращают T или NIL:

Базовая функция-предикат atom проверяет, является ли аргумент атомом:

```
> (atom 'a)
T
> (atom 234)
T
> (atom '(a b))
NIL
> (atom nil)
T
> (atom t)
T
```

Базовая функция-предикат eq проверяет тождественность двух символов:

```
> (eq 'Madrid 'Madrid)
T
> (eq 'Athens 'Madrid)
NIL
> (eq '() nil)
T
> (eq '(a b) '(a b))
NIL
> (eq 3.14 3.14)
NIL
```

Стандартные функции

Lisps are only good when they're huge!
Grue

Начнем знакомство со стандартными арифметическими функциями:

```
> (+ 1000 5)
1005
> (- 1000 5)
995
> (* 1000 5)
5000
> (/ 1000 5)
200
```

Функция возведения в степень – expt:

```
> (expt 2 3)
8
```

Некоторые встроенные функции (предикаты) проверяют выполнение некоторого условия и возвращают T или NIL.

Функция-предикат listp идентифицирует списки:

```
> (listp '(a b))
T
> (listp 'a)
NIL
```

Функция null проверяет на пустой список:

```
> (null '())
T
> (null '(a b))
NIL
```

Функция-предикат = сравнивает числа различных типов:

```
> (= 5 5.0)
T
> (= 5 10/2)
T
> (= 5 5.0 10/2)
T
```

Функция-предикат eql сравнивает числа одинаковых типов:

```
> (eql 3.14 3.14)
T
```

Функция-предикат equal проверяет одинаковость двух списков:

```
> (equal '(a b) '(a b))
T
> (equal 2 'cat)
NIL
```

Функция-предикат equalp сравнивает произвольные лисповские объекты. Этот предикат может потребоваться, когда нет уверенности в типе сравниваемых объектов или в корректности использования других предикатов сравнения:

```
> (equalp '(a b) '(a b))
T
```

Недостатком универсальных предикатов типа equalp является то, что их применение требует от системы несколько большего объема вычислений, чем использование специализированных предикатов.

```
> (numberp 1775)
T
> (numberp 'Orange)
NIL

> (symbolp 'Grant)
T
```

```
> (symbolp '1861)
NIL

> (zerop 0)
T
> (zerop 7)
NIL

> (evenp '1054)
T
> (evenp '1125)
NIL

> (oddp '1805)
T
> (oddp '1812)
NIL

> (atom 1852)
T
> (atom '(Washington Dublin London Canberra))
NIL

> (< 2 3)
T

> (< 3 2)
NIL

> (> 3 2)
T

> (> 2 3)
NIL

> (< 3 5 7)
T

> (> 8 6 4)
T
```

Вложение функций

```
> (/ 31 5)
31/5
> (float 31/5)
6.2
> (float (/ 31 5))
6.2

> (cons '+ '(2 3))
```

```
(+ 2 3)
> (* (cons '+ '(2 3)) 4)
; Evaluation aborted
> (* (eval (cons '+ '(2 3))) 4)
20
```

Например, найти наименьшее среди тех элементов первого списка, которые не входят во второй список:

```
> (apply #'min (set-difference '(3 5 7) '(3 4))
5
```

Стандартные функции (продолжение)

Обработку списков всегда можно свести к описанным ранее базовым функциям cons, car, cdr, atom и eq, но программирование лишь с их использованием было бы очень сложным. Поэтому в Лисп-систему включено свыше 700 встроенных функций для различных действий и различных ситуаций.

Функция list создает список из элементов:

```
> (list 'a 'b 'c)
(A B C)

> (cons 'a (cons 'b (cons 'c nil)))
(A B C)
```

Вложенные вызовы car и cdr можно записывать в сокращенном виде:

```
> (car (cdr '(a b c)))
B
> (cadr '(a b c))
B
```

или

```
> (car (cdr (cdr (cdr '(a b c d)))))
D
> (cadddr '(a b c d))
D
```

или

```
> (car (cdr (cdr (car '((a b c d) (e f g h))))))
C
> (caddar '((a b c d) (e f g h)))
C
```

Функция member проверяет, входит ли элемент в список:

```
> (member 'b '(a b c))
```

```
(B C)
```

Функция nth выделяет n-й элемент списка:

```
> (nth 2 '(Washington Dublin London Canberra))
LONDON
```

Функция nth возвращает элементы списка, начиная с n-го элемента:

```
> (nthcdr 2 '(Washington Dublin London Canberra))
(LONDON CANBERRA)
```

Функция append соединяет списки:

```
> (append '(a b) '(c d) '(e))
(A B C D E)
```

Функция revappend переворачивает список и соединяет его со вторым:

```
> (revappend '(a b c) '(d e f))
(C B A D E F)
```

Функция last возвращает последний элемент списка:

```
> (last '(a b c d))
(D)
```

Функция butlast возвращает все элементы списка кроме последнего:

```
> (butlast '(a b c d))
(A B C)
```

Функция length возвращает длину списка:

```
> (length '(a b c d e f g h i j k l m n o p q r s t u v w x y z))
26
```

Функция subseq возвращает подпоследовательность:

```
> (subseq '(a b c d) 1 2)
(B)

> (subseq '(a b c d) 1)
(B C D)
```

Функция reverse переворачивает список:

```
> (reverse '(a b c))
(C B A)
```

Если элемент не входит в список, то функция adjoin ставит его на первое место:

```
> (adjoin 'c '(a b c))
(A B C)
> (adjoin 'h '(a b c))
(H A B C)
```

Функция union объединяет множества:

```
R> (union '(a b c) '(c b d))
(A C B D)
```

Функция intersection возвращает пересечение множеств:

```
> (intersection '(a b c) '(b b c))
(B C)
```

Функция set-difference возвращает разность множеств:

```
> (set-difference '(a b c d e) '(b c))
(A D E)
```

Функция set-exclusive-or возвращает элементы списков, которые входят в один из списков, но не входят в оба списка:

```
> (set-exclusive-or '(a b c d e) '(b e))
(A C D)
```

Возвращает позицию первого элемента, начиная с которого списки отличаются:

```
> (mismatch '(a b c) '(a b d))
2
```

Функция merge соединяет и сортирует последовательности:

```
> (merge 'list '(1 2 3) '(1 2 5) #'<)
(1 1 2 2 3 5)

> (merge 'string "bank" "account"  #'string<)
"abaccnkount"
```

Функция every проверяет условие для всех элементов списка:

```
> (every #'oddp '(1 3 5 7))
T

> (every #'> '(1 3 5) '(0 2 4))
T
```

Функция some проверяет условие для некоторых элементов списка:

```
> (some #'evenp '(1 2 5 7))
T
```

Функция apply применяет функцию к одному и более аргументу, последним из которых должен быть список:

```
> (apply #'* '(8 9 10))
720

> (apply #'* 8 9 '(10))
720
```

Функция funcall вызывает функцию с аргументами:

```
> (funcall #'* 8 9 10)
720
```

Среди функций, принимающих функции в качестве аргументов, часто применяются функции сортировки sort и чистки remove-if. Первая из них является функцией сортировки общего назначения. Она принимает список аргументов и предикат, а возвращает список, отсортированный пропусканием через предикат каждой пары элементов:

```
> (sort '(4 2 5 1 7 6 3) #'<)
(1 2 3 4 5 6 7)
```

Функция чистки списка remove-if. принимает функцию и список, а возвращает все элементы списка, для которых функция вернула false:

```
> (remove-if #'evenp '(1 2 3 4 5 6 7))
(1 3 5 7)
```

Логические функции

```
> (and 5 (< 1 2))
T

> (or (< 2 1) (< 1 2))
T

> (not (< 2 1))
T
```

Ветвление вычислений

```
> (when (< 1 2) 'yes)
YES

> (if (< 1 2) 'yes 'no)
YES

(defun check (n)
```

```
  (cond ((zerop n) 'zero)
        ((plusp n) 'plus)
        ('minus)))

> (check 0)
ZERO
```

Создание (определение) функций

> Lisp isn't a language, it's a building material.
>
> Alan Kay

В Лиспе программы состоят из функций, которые вызывают друг друга. Определение функции состоит из имени функции, за которым в скобках следуют формальные параметры функции (формальность параметров означает, что их можно поменять на любые другие символы, и это не отразится на вычислениях, определяемых функцией) и тело функции.

Дать имя и определить новую функцию можно с помощью функции defun: (defun add-two (a b) (+ a b)) - определяет функцию с именем add-two и списком аргументов (a b); (+ a b) - тело функции, которое состоит из последовательности выражений. Функция возвращает результат последнего выражения, в приведенном примере (+ a b).

```
> (add-two 25 4000)
4025

(defun div-two (a b) (/ a b))

> (div-two 8000 5)
1600
```

Аналог встроенной функции seventh:

```
(defun -seventh (w) (car (cdr (cdr (cdr (cdr (cdr (cdr w))))))))

> (-seventh '(1 2 3 4 5 6 7 8 9 10))
7
```

Сумма элементов одноуровневого числового списка:

```
(defun sum-list (w) (if (null w) 0 (+ (car w) (sum-list (cdr w)))))

> (sum-list '(1 2 3 4))
10
```

Аналог встроенной функции member:

```
(defun -member (m w)
  (cond ((atom w) nil)
```

```
        ((eql (car w) m) w)
        ((-member  m (cdr w)))))

> (xmember 'c '(a b c d e))
(C D E)
```

Рекурсивные функции

À la guerre comme à la guerre.
французская пословица

Война это ... война!
Григорий Горин «Тот самый Мюнхгаузен»

Рекурсивной называется функция, которая для повторяющихся вычислений вызывает саму себя. Примерами рекурсии являются смерть Кощея, матрешка, временная петля ("День сурка"), фальстарт, калейдоскоп, девочка, прыгающая на скакалке, и собака, носящаяся за своим хвостом.

Функция возведения в степень является рекурсивной:

```
(defun -expt (m n) (if (= n 1) m (* m (-expt m (1- n)))))

> (-expt 2 3)
8
```

Функция -expt вызывает себя до тех пор, пока ее аргумент n (счетчик) не уменьшится до 1 (то есть n – 1 раз), после чего функция вернет значение m. При каждой передаче значения m на предыдущий уровень, результат умножается на m. Так m окажется перемноженным на себя n раз.

Другим примером рекурсии является функция нахождения факториала (произведения данного числа на все целые положительные числа, меньшие данного). Например, факториал 4 есть 1 * 2 * 3 * 4 = 24.

```
(defun factorial (n)
  (if (zerop n) 1 (* n (factorial (- n 1)))))

> (factorial 4)
24
```

Рекурсивные функции хорошо подходят для работы со списками, поскольку списки часто рекурсивно содержат подсписки.

Простая рекурсия

Рекурсия является простой, если вызов функции встречается в некоторой ветви лишь один раз:

```
(defun x-length (w)
  (cond ((null w) 0)
        ((1+ (x-length (cdr w))))))

> (x-length '(a b c d))
4

(defun insert (a w)
  (cond ((null w) (cons a nil))
        ((< a (car w)) (cons a w))
        ((cons (car w) (insert a (cdr w))))))

> (insert 4 '(1 3 7))
(1 3 4 7)
```

Хвостовая рекурсия

Хвостовой рекурсией является функция, чей рекурсивный вызов самой себя является ее последней операцией:

```
(defun t-length (w &optional (acc 0))
  (cond ((null w) acc)
        ((t-length (cdr w) (1+ acc)))))

> (t-length '(a b c d))
4
```

&optional - указатель на то, что далее следуют необязательные (опциональные) параметры.

Параллельная рекурсия

Функция называется параллельной, если она встречается одновременно в нескольких аргументах функции.

Следующая функция принимает два списка-аргумента w и v, возвращает t, если v является подсписком w. Элементами списков могут быть атомы и списки любой вложенности:

```
(defun subp (v w)
  (cond ((null w) nil)
        ((equal v (car w)) t)
        ((atom (car w)) (subp v (cdr w)))
        (t (subp v (car w)) (subp v (cdr w)))))

> (subp '(a) '((a) b))
T
```

Здесь рекурсия применяется как к голове, так и к хвосту списка.

Взаимная рекурсия

Волки от испуга скушали друг друга.
Корней Чуковский «Тараканище»

Рекурсия является взаимной между двумя или более функциями, если они вызывают друг друга. Таковыми являются функции reverse-list и rearrange, которые обращают элементы на всех уровнях списка:

```
(defun reverse-list (w)
  (cond ((atom w) w)
        ((rearrange w nil))))

(defun rearrange (v acc)
  (cond ((null v) acc)
        ((rearrange (cdr v) (cons (reverse-list (car v)) acc)))))
```

```
> (reverse-list '((a b) c (d e) f))
(F (E D) C (B A))
```

Функция MAPCAR

Функция mapcar принимает в качестве аргументов функцию и один или более списков (один на каждый параметр функции) и применяет функцию последовательно к элементам каждого списка:

```
> (mapcar #'1- '(100 100 100))
(99 99 99)

> (mapcar #'list '(a b c) '(1 2 3))
((A 1) (B 2) (C 3))

> (mapcar #'list '(a b c d) '(1 2 3))
((A 1) (B 2) (C 3))

> (mapcar #'+ '(10 100 1000) '(1 2 3))
(11 102 1003)

> (mapcar #'car '((Ottawa CA) (Canberra AU) (Wellington NZ)))
(OTTAWA CANBERRA WELLINGTON)

> (mapcar #'cadr '((Ottawa CA) (Canberra AU) (Wellington NZ)))
(CA AU NZ)
```

Достаточно описать в виде функции, что нужно сделать с элементом списка и затем применить последовательно эту функцию ко всем элементам списка.

Макрос LAMBDA

Мы уже видели, насколько удобно иметь возможность обращаться с функциями, как с данными. Во многих языках, даже если мы могли бы передать функцию в качестве аргумента чему-то, вроде mapcar, она бы, тем не менее, была функцией, определенной заранее в каком-нибудь исходном файле. Если бы требовался именно цельный код, добавляющий 10 к каждому элементу списка, мы бы определили функцию с именем 10+ или каким-то подобным только для этого единственного применения:

```
(defun 10+ (n)
  (+ n 10))

> (mapcar #'10+ '(1 2 3 4 5))
(11 12 13 14 15)
```

Для этой цели в Лиспе существует макрос lambda, при помощи которого вызывается безымянная (анонимная) временная функция, определяемая в месте использования:

```
> (mapcar #'(lambda (n) (+ n 10)) '(1 2 3 4 5))
(11 12 13 14 15)
```

Макрос LOOP

При помощи макроса loop организуются циклы:

```
(defun row (n)
  (loop for i from 1 to n
        collect i))

> (row 10)
(1 2 3 4 5 6 7 8 9 10)

(defun factorial (n)
  (reduce #'* (loop for a from 1 to n collect a)))

> (factorial 4)
24

(defun cube-row (n)
  (loop for i from 1 to n
        collect (expt i 3)))

> (cube-row 10)
(1 8 27 64 125 216 343 512 729 1000)

(defun 10+ (w)
  (loop for i in w
        collect (+ i 10)))
```

```
> (10+ '(1 2 3 4))
(11 12 13 14)

(defun repeat (a n)
  (loop for i from 1 to n
        collect a))

> (repeat 2 10)
(2 2 2 2 2 2 2 2 2 2)

(defun rand (r n)
  (loop for i from 1 to n
        collect (random r)))

> (rand 10 5)
(5 3 4 4 6)

(defun sum (n)
  (loop for i from 1 to n
        sum i))

> (sum 8725)
38067175

(defun even/odd (w)
  (loop for i in w
        if (evenp i)
        collect i into evens
        else collect i into odds
        finally (return (values evens odds))))

> (even/odd '(1 2 3 4 5 6))
(2 4 6)
(1 3 5)
```

Функция FORMAT

```
Год создания Лиспа:
> (format nil "~r" 1958)
"one thousand, nine hundred and fifty-eight"
> (format nil "~:r" 1958)
"one thousand, nine hundred fifty-eighth"
> (format nil "~@r" 1958)
"MCMLVIII"
> (format nil "~:@r" 1958)
"MDCCCCLVIII"

Год падения Западной Римской империи (Imperium Romanum Occidentale):
> (format nil "~:@r" 476)
"CCCCLXXVI"
```

Легенда об изобретателе шахматной игры - количество зерен на 64-й клетке шахматной доски:

```
> (format nil "~r" (1- (expt 2 64)))
"eighteen quintillion, four hundred and forty-six quadrillion, seven
hundred and forty-four trillion, seventy-three billion, seven hundred
and nine million, five hundred and fifty-one thousand, six hundred and
fifteen"

> (format nil "~r" (1- (expt 10 66)))
"nine hundred and ninety-nine vigintillion, nine hundred and ninety-
nine novemdecillion, nine hundred and ninety-nine octodecillion, nine
hundred and ninety-nine septendecillion, nine hundred and ninety-nine
sexdecillion, nine hundred and ninety-nine quindecillion, nine hundred
and ninety-nine quattuordecillion, nine hundred and ninety-nine trede-
cillion, nine hundred and ninety-nine duodecillion, nine hundred and
ninety-nine undecillion, nine hundred and ninety-nine decillion, nine
hundred and ninety-nine nonillion, nine hundred and ninety-nine octil-
lion, nine hundred and ninety-nine septillion, nine hundred and nine-
ty-nine sextillion, nine hundred and ninety-nine quintillion, nine
hundred and ninety-nine quadrillion, nine hundred and ninety-nine
trillion, nine hundred and ninety-nine billion, nine hundred and nine-
ty-nine million, nine hundred and ninety-nine thousand, nine hundred
and ninety-nine"
```

Множественное число –s:

```
> (format nil "~r war~:p" 1)
"one war"
> (format nil "~r war~:p" 2)
"two wars"
```

Множественное число –y или –ies:

```
> (format nil "~r histor~:@p" 1)
"one history"
> (format nil "~r histor~:@p" 2)
"two histories"
```

В императивных языках программа представляет собой набор команд (среди которых могут находиться и вызовы функций). В функциональных языках программа представляет собой совокупность функций. Функции определяются через другие функции.

Функциональное программирование позволяет представить всю программу в виде одной функции, которая вызывает другие функции. Примеры применения функций к решению задач представлены в следующем разделе.

Примеры решений

```
Программы - самые сложные артефакты челове-
чества. Сущности, которыми они оперируют,
иногда не вмещаются в нашу Вселенную.

                                    Сергей Зефиров

Вещи, сделанные без примеров, таковы, что их
сущности надо бояться.
                                    Вильям Шекспир
```

Структура 1 (функция ATOM) > ATOM

Задача 1.1 inc.lisp

```
Определить функцию, которая увеличивает число на 1.

Решение 1.1.1

(defun inc (a)
  (+ a 1))

> (inc 100)
101
```

Задача 1.2 dec.lisp

```
Определить функцию, которая уменьшает число на 1.

Решение 1.2.1

(defun dec (a)
  (- a 1))

> (dec 100)
99
```

Задача 1.3 inc-n.lisp

```
Определить функцию, которая увеличивает число на n.

Решение 1.3.1
```

```
(defun inc-n (a n)
  (+ a n))

> (inc-n 1000 10)
1010
```

Задача 1.4 triple-number.lisp

Определить функцию, которая умножает число на 3.

Решение 1.4.1

```
(defun triple-number (number)
  (* number 3))

> (triple-number 1000)
3000
```

Решение 1.4.2

```
(defun triple-number (number)
  (+ number number number))

> (triple-number 1000)
3000
```

Задача 1.5 fibonacci.lisp

Написать функцию, возвращающую n-е число Фибоначчи.

Решение 1.5.1

```
(defun fibonacci (n)
  (if (> n 2) (+ (fibonacci (- n 1)) (fibonacci (- n 2))) n))

> (fibonacci 10)
89
> (fibonacci 11)
144
```

Задача 1.6 digits-expt-125-100.lisp

Найти из скольких цифр состоит число 125 в степени 100.

Решение 1.6.1

```
> (expt 125 100)
490909346529772655309577195498627564297521551249944956511154911718710525472171585646009788403733195227718357156513187851316791861042471890280751482410896345225310546445986192853894181098439730703830718994140625
> (write-to-string (expt 125 100))
```

```
"490909346529772655309577195498627564297521551249944956511154911718710
525472171585646009788403733195227718357156513187851316791861042471890 2
807514824108963452253105464459861928538941810984397307038307189941406 2
5"
> (loop for a across (write-to-string (expt 125 100)) collect (string
a))
("4" "9" "0" "9" "0" "9" "3" "4" "6" "5" "2" "9" "7" "7" "2" "6" "5"
"5" "3" "0" "9" "5" "7" "7" "1" "9" "5" "4" "9" "8" "6" "2" "7" "5"
"6" "4" "2" "9" "7" "5" "2" "1" "5" "5" "1" "2" "4" "9" "9" "4" "4"
"9" "5" "6" "5" "1" "1"  "1" "5" "4" "9" "1" "1" "7" "1" "8" "7" "1"
"0" "5" "2" "5" "4" "7" "2" "1" "7" "1" "5" "8" "5" "6" "4" "6" "0"
"0" "9" "7" "8" "8" "4" "0" "3" "7" "3" "3" "1" "9" "5" "2" "2" "7"
"7" "1" "8" "3" "5" "7" "1" "5" "6" "5" "1" "3" "1" "8" "7" "8" "5"
"1" "3" "1" "6" "7" "9" "1" "8" "6" "1" "0" "4" "2" "4" "7" "1" "8"
"9" "0" "2" "8" "0" "7" "5" "1" "4" "8" "2" "4" "1" "0" "8" "9" "6"
"3" "4" "5" "2" "2" "5" "3" "1" "0" "5" "4" "6" "4" "4" "5" "9" "8"
"6" "1" "9" "2" "8" "5" "3" "8" "9" "4" "1" "8" "1" "0" "9" "8" "4"
"3" "9" "7" "3" "0" "7" "0" "3" "8" "3" "0" "7" "1" "8" "9" "9" "4"
"1" "4" "0" "6" "2" "5")
> (length (loop for a across (write-to-string (expt 125 100)) collect
(string a)))
210
```

Решение 1.6.2

```
(defun digits-expt-125-100 ()
  (length (loop for a across (write-to-string (expt 125 100))
                collect (string a))))

> (digits-expt-125-100)
210
```

Решение 1.6.3

```
(defun digits-expt-100 (n)
  (length (loop for a across (write-to-string (expt n 100))
                collect (string a))))

> (digits-expt-100 125)
210
```

Решение 1.6.4

```
(defun digits-expt (n m)
  (length (loop for a across (write-to-string (expt n m))
                collect (string a))))

> (digits-expt 125 100)
210
```

Решение 1.6.5

```
(defun digits-expt-125-100 ()
```

```
  (loop for a across (write-to-string (expt 125 100))
        counting a))

> (digits-expt-125-100)
210
```

Решение 1.6.6

```
(defun digits-expt (n m)
  (loop for a across (write-to-string (expt n m))
        count a))

> (digits-expt 125 100)
210
```

Решение 1.6.7

```
(defun digits-expt (n m)
  (count-digits (expt n m)))

(defun count-digits (a)
  (cond ((< a 1) 0)
        (t (1+ (count-digits (/ a 10))))))

> (digits-expt 125 100)
210
```

Решение 1.6.8

```
(defun digits-expt (n m)
  (count-digits (expt n m)))

(defun count-digits (a)
  (if (< a 1) 0 (1+ (count-digits (/ a 10)))))

> (digits-expt 125 100)
210
```

Решение 1.6.9

```
(defun digits-expt (n m)
  (labels ((count-digits (a)
             (if (< a 1) 0 (1+ (count-digits (/ a 10))))))
    (count-digits (expt n m))))

> (digits-expt 125 100)
210
```

Задача 1.7 ctg.lisp

Определить функцию для вычисления котангенса.

Решение 1.7.1

```
(defun ctg (a)
  (/ (cos a) (sin a)))

> (ctg 90)
-0.5012028
```

Решение 1.7.2

```
defun ctg (a)
  (/ 1.0 (tan a)))

> (ctg 90)
-0.50120276
```

Решение 1.7.3

```
defun ctg (a)
  (/ (tan a)))

> (ctg 90)
-0.50120276
```

Задача 1.8 archive-string.lisp

A well-defined problem is half solved.

Michael Osborne

Есть строка, в ней надо заменить подстроки, которые состоят из 4х и больше идущих подряд одинаковых символов на знак ! + количество подряд идущих одинаковых символов + сам символ. Например:
> (f “aabbaaabbbcsdkAAAAbbbbb”)
“aabbaaabbbcsdk!4A!5b”

Решение 1.8.1

```
(defun ar (n ac)
  (concatenate 'string "!" (write-to-string n) (car ac)))

(defun chive (w &optional ac acc (n 0) &aux (a (car w)) (d (cdr w)))
  (cond ((null w)
         (reverse (if (> n 3) (cons (ar n ac) acc) (nconc ac acc))))
        ((equal a (car ac)) (chive d (cons a ac) acc (+ n 1)))
        ((> n 3) (chive d (list a) (cons (ar n ac) acc) 1))
        ((chive d (list a) (nconc ac acc) 1))))

(defun archive-string (s)
  (apply #'concatenate 'string (chive (map 'list #'string s))))

> (archive-string "aabbaaabbbcsdkAAAAbbbbb")
```

```
"aabbaaabbbcsdk!4A!5b"
```

Задача 1.9 circle-area.lisp

Известна длина окружности. Найти площадь круга, ограниченного этой окружностью.

Решение 1.9.1

```
(defun circle-area (c)
  (/ (* pi (expt (/ c pi) 2)) 4))

> (circle-area 10)
7.957747154594766788L0
```

Решение 1.9.2

```
(defun circle-area (c)
  (* pi  (/ c pi 2) (/ c pi 2)))

> (circle-area 10)
7.957747154594766788L0
```

Решение 1.9.3

```
(defun circle-area (c)
  (* (/ c 2) (/ c pi 2)))

> (circle-area 10)
7.957747154594766788L0
```

Решение 1.9.4

```
(defun circle-area (c)
  (/ (* c c) 4 pi))

> (circle-area 10)
7.957747154594766788L0
```

Решение 1.9.5

```
(defun circle-area (c)
  (/ (expt c 2) 4 pi))

> (circle-area 10)
7.957747154594766788L0
```

Решение 1.9.6

```
(defun circle-area (c)
  (/ (* c c 1/4) pi))
```

```
> (circle-area 10)
7.957747154594766788L0
```

Решение 1.9.7

```
(defun circle-area (c)
  (/ (* pi (expt (/ c pi) 2)) 4))

> (circle-area 10)
7.957747154594766788L0
```

Задача 1.10 trim-words.lisp

Создать программу, которая удаляет первую букву каждого слова в предложении.

Решение 1.10.1

```
(defun cut (s)
  (mapcar #'(lambda (a) (subseq a 1))
          (mapcar #'string (read-from-string
                            (concatenate 'string "(" s ")")))))

(defun glue (w)
  (cond ((null w) nil)
        ((concatenate 'string (car w) " " (glue (cdr w))))))

(defun trim-words(s)
  (string-right-trim " " (glue (cut s))))

> (trim-words "abc abc abc")
"BC BC BC"
```

Решение 1.10.2

```
(defun cut (s)
  (mapcar #'(lambda (a) (subseq a 1))
          (mapcar #'string (read-from-string
                            (concatenate 'string "(" s ")")))))
(defun glue (w)
  (when w (concatenate 'string (car w) " " (glue (cdr w)))))

(defun trim-words(s)
  (string-right-trim " " (glue (cut s))))

> (trim-words "abc abc abc")
"BC BC BC"
```

Задача 1.11 remove-extra-spaces.lisp

В заданном тексте слова разделены пробелами. Удалить лишние пробелы

так, чтобы между словами осталось только по одному пробелу.

Решение 1.11.1

```
(defun remove-extra-spaces (s)
  (concat (mapcar #'string
                  (read-from-string
                   (concatenate 'string "(" s ")")))))

(defun concat (w &optional (ac ""))
  (cond ((null w) (string-right-trim '(#\space) ac))
        ((concat (cdr w) (concatenate 'string ac (car w) " ")))))

> (remove-extra-spaces "a ab  abc")
"A AB ABC"
```

Задача 1.12 max-lets.lisp

Найти в заданном тексте самое длинное слово.

Решение 1.12.1

```
(defun max-lets (s)
  (caar (sort (mapcar #'(lambda (a) (list a (length a)))
                (mapcar #'string (read-from-string
                                  (concatenate 'string "(" s ")"))))
              #'> :key #'cadr)))

> (max-lets "a ab abc")
"ABC"
```

Комментарии (по функциям):

```
> (concatenate 'string "(" "a ab abc" ")")
"(a ab abc)"
> (read-from-string "(a ab abc)")
(A AB ABC)
10
> (mapcar #'string '(A AB ABC))
("A" "AB" "ABC")
> (mapcar #'(lambda (a) (list a (length a))) '("A" "AB" "ABC"))
(("A" 1) ("AB" 2) ("ABC" 3))
> (sort '(("A" 1) ("AB" 2) ("ABC" 3)) #'> :key #'cadr)
(("ABC" 3) ("AB" 2) ("A" 1))
> (caar '(("ABC" 3) ("AB" 2) ("A" 1)))
"ABC"
```

Задача 1.13 romanp.lisp

Для представления римских цифр используются символы: I - один, V - пять, X -десять, L - пятьдесят. C - сто, D - пятьсот, M - тысяча. Для

изображения числа с помощью римских цифр используются общеизвестные правила; так, например, 482 -CDLXXXII. 1999 - MCMXCIX. Напишите функцию (f R), которая проверяет, правильна ли запись некоторого числа римскими цифрами R. Например, (f IIIC) = nil.

Решение 1.13.1

```
(defun romanp (r &optional (n 3999))
  (cond ((zerop n) nil)
        ((string= (string r) (format nil "~@r" n)) t)
        ((romanp r (1- n)))))

> (romanp 'CDLXXVI)
T
> (romanp 'IIIC)
NIL
```

Решение 1.13.2

```
(defun romanp (r)
  (loop for a from 1 to 3999
        when (string= (string r) (format nil "~@r" a)) collect a))

> (romanp 'CDLXXVI)
(476)
> (romanp 'IIIC)
NIL
```

Решение 1.13.3

```
(defun romanp (r &optional (n 3999))
  (cond ((zerop n) nil)
        ((string= (string r) (format nil "~@r" n)) n)
        ((romanp r (1- n)))))

> (romanp 'CDLXXVI)
476
> (romanp 'IIIC)
NIL
```

Решение 1.13.4

```
(defun romanp (n)
  (member n (mapcar #'roman (loop for i from 1 to 3999 collect i))))

(defun roman (n)
  (read-from-string (format nil "~@r" n)))

> (romanp 'MMXII)
(MMXII MMXIII MMXIV MMXV MMXVI MMXVII MMXVIII MMXIX MMXX MMXXI MMXXII
MMXXIII MMXXIV MMXXV MMXXVI MMXXVII MMXXVIII MMXXIX MMXXX MMXXXI
MMXXXII MMXXXIII MMXXXIV MMXXXV MMXXXVI MMXXXVII MMXXXVIII MMXXXIX
MMXL MMXLI MMXLII MMXLIII MMXLIV MMXLV MMXLVI MMXLVII MMXLVIII MMXLIX
```

```
MML MMLI MMLII MMLIII MMLIV MMLV MMLVI MMLVII MMLVIII MMLIX MMLX MMLXI
MMLXII MMLXIII MMLXIV MMLXV MMLXVI MMLXVII MMLXVIII MMLXIX MMLXX
MMLXXI MMLXXII MMLXXIII MMLXXIV MMLXXV MMLXXVI MMLXXVII MMLXXVIII
MMLXXIX MMLXXX MMLXXXI MMLXXXII MMLXXXIII MMLXXXIV MMLXXXV … )

> (romanp 'IIIC)
NIL
```

Решение 1.13.5 (для XLISP)

```
(defun romanp (n)
  (member n '(I II III IV V VI VII VIII IX X XI XII XIII XIV XV XVI
XVII XVIII XIX XX XXI XXII XXIII XXIV XXV XXVI XXVII XXVIII XXIX XXX
XXXI XXXII XXXIII XXXIV XXXV XXXVI XXXVII XXXVIII XXXIX XL XLI XLII
XLIII XLIV XLV XLVI XLVII XLVIII XLIX L LI LII LIII LIV LV LVI LVII
LVIII LIX LX LXI LXII LXIII LXIV LXV LXVI LXVII LXVIII LXIX LXX LXXI
LXXII LXXIII LXXIV LXXV LXXVI LXXVII LXXVIII LXXIX LXXX … MMMCMLXXI
MMMCMLXXII MMMCMLXXIII MMMCMLXXIV MMMCMLXXV MMMCMLXXVI MMMCMLXXVII
MMMCMLXXVIII MMMCMLXXIX MMMCMLXXX MMMCMLXXXI MMMCMLXXXII MMMCMLXXXIII
MMMCMLXXXIV MMMCMLXXXV MMMCMLXXXVI MMMCMLXXXVII MMMCMLXXXVIII
MMCMLXXXIX MMMCMXC MMMCMXCI MMMCMXCII MMMCMXCIII MMMCMXCIV MMMCMXCV
MMMCMXCVI MMMCMXCVII MMMCMXCVIII MMMCMXCIX)))

> (romanp 'MMXII)
(MMXII MMXIII MMXIV MMXV MMXVI MMXVII MMXVIII MMXIX MMXX MMXXI MMXXII
MMXXIII MMXXIV MMXXV MMXXVI MMXXVII MMXXVIII MMXXIX MMXXX MMXXXI
MMXXXII MMXXXIII MMXXXIV MMXXXV MMXXXVI MMXXXVII MMXXXVIII MMXXXIX
MMXL MMXLI MMXLII MMXLIII MMXLIV MMXLV MMXLVI MMXLVII MMXLVIII MMXLIX
MML MMLI MMLII MMLIII MMLIV MMLV MMLVI MMLVII MMLVIII MMLIX MMLX MMLXI
MMLXII MMLXIII MMLXIV MMLXV MMLXVI MMLXVII MMLXVIII MMLXIX MMLXX
MMLXXI MMLXXII MMLXXIII MMLXXIV MMLXXV MMLXXVI MMLXXVII MMLXXVIII
MMLXXIX MMLXXX MMLXXXI MMLXXXII MMLXXXIII MMLXXXIV MMLXXXV … )

> (romanp 'IIIC)
NIL
```

Задача 1.14 dump-duplicates.lisp

Из набора атомов вычеркнуть дубликаты, т.е. повторяющиеся: (1 2 3 4 5 1 2) > (1 2 3 4 5).

Решение 1.14.1

```
(defun dump-duplicates (w)
  (cond ((null w) nil)
        ((member (car w) (cdr w)) (dump-duplicates (cdr w)))
        (t (cons (car w) (dump-duplicates (cdr w))))))

> (dump-duplicates '(1 2 3 4 5 1 2))
(3 4 5 1 2)
```

Решение 1.14.2

```
(defun dump-duplicates (w)
  (when w (if (member (car w) (cdr w))
              (dump-duplicates (cdr w))
              (cons (car w) (dump-duplicates (cdr w))))))

> (dump-duplicates '(1 2 3 4 5 1 2))
(3 4 5 1 2)
```

Решение 1.14.3

```
(defun dump-duplicates (w &aux (a (car w)) (d (cdr w)))
  (when w (if (member a d)
              (dump-duplicates d)
              (cons a (dump-duplicates d)))))

> (dump-duplicates '(1 2 3 4 5 1 2))
(3 4 5 1 2)
```

Решение 1.14.4

```
(defun dump-duplicates (w)
  (when w ((lambda (a d)
             (if (member a d)
                 (dump-duplicates d)
                 (cons a (dump-duplicates d))))
           (car w) (cdr w))))

> (dump-duplicates '(1 2 3 4 5 1 2))
(3 4 5 1 2)

(dump-duplicates "abcbddefdfcfg")
Evaluation aborted
```

P.S. в Common Lisp - remove-duplicates и delete-duplicates работают с любыми последовательностями, в том числе и строками:

```
(remove-duplicates "abcbddefdfcfg")
"abedcfg"
```

Задача 1.15 chess-prize.lisp

Написать функцию, которая вычисляет, сколько будет зёрен на шахматной доске, если, положив на первую клетку доски одно зерно, класть на каждую следующую клетку вдвое больше зёрен, чем на предыдущую.

Решение 1.15.1

```
(defun chess-prize (number-of-squares)
  (1- (expt 2 number-of-squares)))

> (chess-prize 64)
```

```
18446744073709551615

> (format nil "~r" (chess-prize 64))
"eighteen quintillion, four hundred and forty-six quadrillion, seven
hundred and forty-four trillion, seventy-three billion, seven hundred
and nine million, five hundred and fifty-one thousand, six hundred and
fifteen"

Решение 1.15.2

(defun chess-prize-recursive (number-of-squares &optional (acc 1))
  (cond ((zerop number-of-squares) (1- acc))
        ((chess-prize-recursive (1- number-of-squares) (* acc 2)))))

> (chess-prize-recursive 64)
18446744073709551615
```

Задача 1.16 sort-vowels<.lisp

```
Дан текст. Записать каждое предложение текста в порядке возрастания
количества гласных букв в слове.

Решение 1.16.1

(defun split (s &optional ac (p (position #\Space s)))
  (cond ((zerop (length s)) ac)
        (p (split (subseq s (1+ p)) (cons (subseq s 0 p) ac)))
        ((cons (subseq s 0) ac))))

(defun ws (w)
 (mapcar #'(lambda (a) (coerce a 'list)) w))

(defun cn (w &optional (n 0))
  (cond ((null w) n)
        ((member (car w) '(#\a #\e #\i #\o #\u))
         (cn (cdr w) (1+ n)))
        ((cn (cdr w) n))))

(defun ez (w)
  (mapcar #'(lambda (e) (list e (cn e))) w))

(defun ts (w)
  (sort w #'< :key #'cadr))

(defun join (w)
  (apply #'concatenate 'string (mapcar #'string w)))

(defun dm (w)
  (cond ((null w) nil)
        ((cons (join (caar w)) (dm (cdr w))))))

(defun ju (w)
```

```
  (format nil "~{~A~^ ~}" w))

(defun sort-vowels< (w)
  (ju (dm (ts (ez (ws (split w)))))))

> (sort-vowels< "aaaw aaw aww w")
"w aww aaw aaaw"
```

Задача 1.17 select-words.lisp

В строке оставить только слова, содержащие заданную комбинацию букв.

Решение 1.17.1

```
(defun occur (x s)
  (delete-if-not #'(lambda (a) (search x a)) s))

(defun split (s)
  (loop for i = 0 then (1+ j)
        as j = (position #\Space s :start i)
        collect (subseq s i j)
        while j))

(defun join (w)
  (format nil "~{~A~^ ~}" w))

(defun select-words (c s)
  (join (occur c (split s))))

> (select-words "bc" "abc bcd cde")
"abc bcd"
```

Решение 10.17.2 (без использования циклов - рекурсия или/и функционалы)

```
(defun occur (x s)
  (delete-if-not #'(lambda (a) (search x a)) s))

(defun split (s &optional ac (p (position #\Space s)))
  (cond ((zerop (length s)) ac)
        (p (split (subseq s (1+ p)) (cons (subseq s 0 p) ac)))
        ((cons (subseq s 0) ac))))

(defun join (w)
  (format nil "~{~A~^ ~}" w))

(defun select-words (c s)
  (join (occur c (reverse (split s)))))

> (select-words "bc" "abc bcd cde")
"abc bcd"
```

Задача 1.18 string-upcase.lisp

Написать функцию изменения регистра строки: прописные, строчные или как в предложении.

Решение 1.18.1

```
> (string-upcase "ab cd")
"AB CD"

> (string-downcase "AB CD")
"ab cd"

> (string-capitalize "ab cd" :end 1)
"Ab cd"
```

Задача 1.19 2exptp.lisp

Написать функцию, которая для аргумента-числа проверяет, является ли оно степенью двойки.

Решение 1.19.1

```
(defun 2exptp (n)
  (cond ((= n 2) t)
        ((< n 2) nil)
        ((2exptp (/ n 2)))))

> (2exptp 4)
T
```

Решение 1.19.2

```
(defun 2exptp (n)
  (cond ((> n 2) (2exptp (/ n 2)))
        ((= n 2))))

> (2exptp 4)
T
```

Задача 1.20 calculate.lisp

Написать функцию, которая принимает ввод двух чисел и операции между ними. Выход из функция командой quit.

Решение 1.20.1

```
(defun  calculate (&aux cmd)
  (loop (print "Enter: Number Operator Number")
        (setq cmd (read-from-string
                   (concatenate 'string "(" (read-line) ")")))
```

```
        (when (eql (car cmd) 'quit) (return 'ok))
        (print (eval `(,(cadr cmd) ,(car cmd) ,(caddr cmd))))))

> (calculate)
"Enter: Number Operator Number" 2 + 3
5
"Enter: Number Operator Number" quit
OK
```

Решение 1.20.2

```
(defun  calculate (&aux w)
  (loop (print '>)
        (setq w (read-from-string
                  (concatenate 'string "(" (read-line) ")")))
        (when (eql (car w) 'quit) (return 'ok))
        (print (eval `(,(cadr w) ,(car w) ,(caddr w))))))

> (calculate)
> 2 + 3
5
> quit
OK
```

Задача 1.21 factorial+.lisp

Определить функции, которая вычисляет сумму промежуточных значений нахождения факториала.

Решение 1.21.1

```
(defun factorial (n)
   (if (zerop n) 1 (* (factorial (1- n)) n)))

(defun factorial+ (n)
  (if (zerop n) 0 (+ (factorial+ (1- n)) (factorial n))))

> (factorial+ 3)
9
> (factorial+ 4)
33
```

Решение 1.21.2

```
(defun factorial+ (n)
  (loop for a from 1 to n
        for m = a then (* a m) sum m))

> (factorial+ 3)
9
> (factorial+ 4)
33
```

Задача 1.22 char-sum-code+.lisp

Определить функции, которая вычисляет сумму кодов знаков строки и возвращает значок, кодом которого является найденная сумма.

Решение 1.22.1

```
(defun char-sum-code (s)
  (code-char (reduce #'+ (map 'list #'char-code s))))

> (char-sum-code "p3")
#\POUND_SIGN
> (char-sum-code "p3js3")
#\LATIN_CAPITAL_LETTER_Y_WITH_HOOK
```

Решение 1.22.1

```
(defun char-sum-code (s)
  (code-char (reduce #'+ (map 'list #'char-code (string s)))))

> (char-sum-code '|p3|)
#\POUND_SIGN
> (char-sum-code '|p3Js3|)
#\LATIN_CAPITAL_LETTER_G_WITH_HOOK
```

Задача 1.23 2/1+3/2...+[n+1]/n.lisp

Определить функцию, которая вычисляет сумму ряда 2/1+3/2...+(n+1)/n.

Решение 1.23.1

```
(defun 2/1+3/2...+[n+1]/n (n)
  (if (zerop n) 0 (+ (/ (1+ n) n) (2/1+3/2...+[n+1]/n (1- n)))))

> (2/1+3/2...+[n+1]/n 2)
7/2
```

Решение 1.23.2

```
(defun 2/1+3/2...+[n+1]/n (n)
  (loop for a from 1 to n summing (/ (1+ a) a)))

> (2/1+3/2...+[n+1]/n 2)
7/2
```

Задача 1.24 num-magic-ticket-9999.lisp

Найти число "счастливых" билетов (сумма первых двух цифр равна сумме последних двух цифр) с номерами от 0000 до 9999 включительно.

Решение 1.24.1

```
(defun num-magic-ticket-9999 (n)
  (row-ticket (add-nine n) n))

(defun row-ticket (nn n &aux (m (int-digit nn)) (z (add-zero m n)))
  (if (>= nn 0)
      (if (check-balance z (/ n 2))
          (1+ (row-ticket (1- nn) n))
          (row-ticket (1- nn) n))
      0))

(defun add-nine (n &optional w)
  (if (< (length w) n) (add-nine n (cons 9 w)) (list-num w)))

(defun list-num (w)
  (parse-integer (apply #'concatenate 'string
                        (mapcar #'write-to-string w))))

(defun add-zero (w n)
  (if (< (length w) n) (add-zero (cons 0 w) n) w))

(defun int-digit (n &optional ac)
  (if (zerop n) ac (int-digit (truncate n 10) (cons (rem n 10) ac))))

(defun check-balance (w n)
  (= (apply #'+ (subseq w 0 n)) (apply #'+ (nthcdr n w))))

> (num-magic-ticket-9999 4)
670
```

Задача 1.25 sum-odd-plus.lisp

Определить функцию, которая находит сумму положительных нечётных чисел меньше 100.

Решение 1.25.1

```
(defun sum-odd-plus (n)
  (cond ((zerop n) 0)
        ((oddp n) (+ n (sum-odd-plus (1- n))))
        ((sum-odd-plus (1- n)))))

> (sum-odd-plus 99)
2500
```

Задача 1.26 longest-palindrome.lisp

Составить программу, которая в заданном предложении находит самое длинное симметричное слово.

Решение 1.26.1

```
(defun palindrome (w) (equalp w (reverse w)))

(defun palindromes (w)
  (loop for a in w when (palindrome a) collect a))

(defun cleave (s)
  (mapcar #'string (read-from-string
                    (concatenate 'string "(" s ")"))))

(defun longest-palindrome (s)
  (car (sort (palindromes (cleave s)) #'> :key #'length)))

> (longest-palindrome "anna hannah")
"HANNAH"
```

Решение 1.26.2

```
(defun dromes (w)
  (loop for a in w when (equal a (reverse a))
        collect a))

(defun longest-palindrome (s)
  (car (sort (dromes (mapcar #'string s))
             #'> :key #'length)))

> (longest-palindrome '(anna hannah))
"HANNAH"
```

Решение 1.26.3

```
(defun dromes (w)
  (loop for a in w when (equal a (reverse a))
        collect a))

(defun longest-palindrome (s)
  (read-from-string
   (car (sort (dromes (mapcar #'string s))
              #'> :key #'length))))

> (longest-palindrome '(anna hannah))
HANNAH
6
```

Решение 1.26.4

```
(defun dromes (w)
  (loop for a in w when (equal a (reverse a))
        collect a))

(defun longest (w &optional b &aux (a (car w)))
  (if w
      (if (> (length a) (length b))
```

```
          (longest (cdr w) a)
          (longest (cdr w) b))
      b))

(defun longest-palindrome (s)
  (read-from-string (longest (dromes (mapcar #'string s)))))

> (longest-palindrome '(anna hannah))
HANNAH
6
```

Решение 1.26.5

```
(defun dromes (w)
  (loop for a in w when (equal a (reverse a))
        collect a))

(defun longest (w &optional b &aux (a (car w)))
  (cond ((null w) b)
        ((> (length a) (length b)) (longest (cdr w) a))
        ((longest (cdr w) b))))

(defun longest-palindrome (s)
  (read-from-string (longest (dromes (mapcar #'string s)))))

> (longest-palindrome '(anna hannah))
HANNAH
6
```

Задача 1.27 remove-repetition.lisp

Дана строка, содержащая предложение. В предложении некоторые из слов записаны подряд несколько раз (предложение заканчивается точкой или знаком восклицания). Получить в новой строке отредактированный текст, в котором удалены подряд идущие вхождения слов в предложении.

Решение 1.27.1

```
(defun drop-words (w)
  (cond ((null w) nil)
        ((eq (car w) (cadr w)) (drop-words (cdr w)))
        ((cons (car w) (drop-words (cdr w))))))

(defun glue (w m)
  (cond ((null (cdr w)) (concatenate 'string (car w) m ))
        ((concatenate 'string (car w) " " (glue (cdr w) m)))))

(defun list-word (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun remove-repetition (s &aux
                          (n (1- (length s)))
```

```
                          (z (subseq s 0 n))
                          (m (subseq s n)))
   (glue (mapcar 'string (drop-words (list-word z))) m))

> (remove-repetition "aa aa aa bb bb.")
"AA BB."
```

Решение 1.27.2

```
(defun drop-words (w)
  (cond ((null w) nil)
        ((eq (car w) (cadr w)) (drop-words (cdr w)))
        ((cons (car w) (drop-words (cdr w))))))

(defun glue (w m)
  (if (cdr w)
      (concatenate 'string (car w) " " (glue (cdr w) m))
      (concatenate 'string (car w) m )))

(defun list-word (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun remove-repetition (s &aux
                          (n (1- (length s)))
                          (z (subseq s 0 n))
                          (m (subseq s n)))
   (glue (mapcar 'string (drop-words (list-word z))) m))

> (remove-repetition "aa aa aa bb bb.")
"AA BB."
```

Решение 1.27.3

```
(defun drop-words (w)
  (cond ((null w) nil)
        ((eq (car w) (cadr w)) (drop-words (cdr w)))
        ((cons (car w) (drop-words (cdr w))))))

(defun glue (w)
  (if (cdr w)
      (concatenate 'string (car w) " " (glue (cdr w)))
      (car w)))

(defun list-word (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun remove-repetition (s &aux
                          (n (1- (length s)))
                          (z (subseq s 0 n))
                          (m (subseq s n)))
  (concatenate 'string
               (glue (mapcar 'string
                             (drop-words (list-word z))))
```

```
                m))

> (remove-repetition "aa aa aa bb bb.")
"AA BB."
```

Задача 1.28 set-length.lisp

Подсчитать, сколько различных символов содержатся в строке.

Решение 1.28.1

```
(defun set-length (s)
  (length (remove-duplicates s)))

> (set-length "aabbc")
3
```

Задача 1.29 display.lisp

Средствами текстового редактора подготовить файл, содержащий два столбца числовых значений (значения аргумента и функции), и написать программу, выводящую эти числа на экран.

Решение 1.29.1

```
d:/old.txt:
11 12
13 14

(defun display (path)
  (with-open-file (s path :direction :input)
    (do ((line (read-line s nil :eof)(read-line s nil :eof)))
        ((eql line :eof)) (print line))))

> (display "d:/old.txt")
"11 12"
"13 14"
NIL
```

Задача 1.30 num-substr.lisp

Напишите функцию, которая в заданной строке определяет количество вхождений в нее заданной подстроки (слова).

Решение 1.30.1 *возвращает число связанных (конъюнктных) подстрок

```
(defun num-substr
    (a s &optional (m (length a)) (z 0) &aux (n (search a s)))
  (if n (num-substr a (subseq s (+ n m)) m (1+ z)) z))

> (num-substr "jar" "jar jar")
```

```
2
> (num-substr "ja" "jar jar")
2
> (num-substr "r" "jar jar")
2
```

Решение 1.30.2

```
(defun num-substr (a s &optional (z 0) &aux (n (search a s)))
  (if n (num-substr a (subseq s (1+ n)) (1+ z)) z))

> (num-substr "ja" "jar jar")
2
> (num-substr "aa" "aaa")
2
```

Решение 1.30.1 *возвращает число разобщенных (дизъюнктных) подстрок

```
(defun num-substr
    (a s &optional (m (length a)) (z 0) &aux (n (search a s)))
  (if n (num-substr a (subseq s (+ n m)) m (1+ z)) z))

> (num-substr "jar" "jar jar")
2
> (num-substr "ja" "jar jar")
2
> (num-substr "r" "jar jar")
2
```

Задача 1.31 sum-divisors.lisp

Определить функцию, которая принимает целое положительное число и возвращает сумму делителей этого числа.

Решение 1.31.1

```
(defun sum-divisors (n)
  (loop for a from 1 to (truncate (/ n 2))
        when (zerop (rem n a))
        sum a))

> (sum-divisors 100)
117
> (sum-divisors 99)
57
```

Решение 1.31.2

```
(defun sum-divisors (n &optional (m (truncate (/ n 2))))
  (cond ((zerop m) 0)
        ((zerop (rem n m)) (+ m (sum-divisors n (1- m))))
        ((sum-divisors n (1- m)))))
```

```
> (sum-divisors 100)
117
> (sum-divisors 99)
57
```

Решение 1.31.3

```
(defun sum-divisors (n &optional (m (truncate (/ n 2))))
  (if (zerop m)
      0
      (if (zerop (rem n m))
          (+ m (sum-divisors n (1- m)))
          (sum-divisors n (1- m)))))

> (sum-divisors 100)
117
> (sum-divisors 99)
57
```

Задача 1.32 min-shuffle.lisp

Определить функцию, которая переставляет цифры целого числа так, чтобы получилось минимальное число.

Решение 1.32.1

```
(defun digitize (n &optional m)
  (cond ((zerop n) m)
        (t (multiple-value-bind (q r) (truncate n 10)
             (digitize q (cons r m))))))

(defun undigitize (w &optional (n 1))
  (if w (+ (* (car w) n) (undigitize (cdr w) (* n 10))) 0))

(defun min-shuffle (n)
  (undigitize (reverse (sort (digitize n) #'<))))

> (min-shuffle 34534534)
33344455
```

Решение 1.32.2

```
(defun digitize (n &optional m)
  (if (zerop n)
      m
      (multiple-value-bind (q r) (truncate n 10)
        (digitize q (cons r m)))))

(defun undigitize (w &optional (n 1))
  (if w (+ (* (car w) n) (undigitize (cdr w) (* n 10))) 0))

(defun min-shuffle (n)
```

```
  (undigitize (reverse (sort (digitize n) #'<))))

> (min-shuffle 34534534)
33344455
```

Решение 1.32.3

```
(defun min-shuffle (n)
  (parse-integer (sort (write-to-string n) #'char<)))

> (min-shuffle 34534534)
33344455
8
```

Задача 1.33 capitalize-line.lisp

Создайте с помощью текстового редактора текстовый файл, каждая строка которого представляет одно предложение. Напишите программу, которая заменяет первую строчную букву предложения на заглавную.

Решение 1.33.1

```
old.txt:
abc abc
abc abc

(defun capitalize-line
    (&optional (in "d:/old.txt") (out "d:/new.txt")         )
  (with-open-file (s in :direction :input)
    (with-open-file (r out :direction :output :if-exists :supersede)
      (do ((line (read-line s  nil :eof)(read-line s nil :eof)))
          ((eql line :eof))
        (if (zerop (length line))
            (format r "~%")
            (format r "~a~%"
                    (string-capitalize line :start 0 :end 1)))))))
> (capitalize-line)
NIL

new.txt:
Abc abc
Abc abc
```

Задача 1.34 deci-to-hex.lisp

Определить функцию для перевода числа из десятичной системы в шестнадцатеричную.

Решение 1.34.1

```
(defun deci-to-hex (n) (format nil "~x" n))
```

```
> (deci-to-hex 9)
"9"
> (deci-to-hex 10)
"A"
> (deci-to-hex 11)
"B"
> (deci-to-hex 24)
"18"
> (deci-to-hex 255)
"FF"
```

Решение 1.34.2

```
(defun deci-to-hex (n) (write-to-string n :base 16))
```

```
> (deci-to-hex 9)
"9"
> (deci-to-hex 10)
"A"
> (deci-to-hex 11)
"B"
> (deci-to-hex 24)
"18"
> (deci-to-hex 255)
"FF"
```

Задача 1.35 hex-to-bin.lisp

Определить функцию для перевода чисел из шестнадцатеричной ситемы в двоичную. Ввод и вывод реализовать через файлы.

```
hex.in
A
18
```

Решение 1.35.1

```
(defun hexa-to-bin (&optional (in "d:/hex.in") (out "d:/bin.in"))
  (with-open-file (s in :direction :input)
    (with-open-file (r out :direction :output :if-exists :supersede)
      (do ((line (read-line s  nil :eof)(read-line s nil :eof)))
          ((eql line :eof))
        (if (zerop (length line))
            (format r "~%")
            (format r "~a~%"
                    (write-to-string
                     (parse-integer line :radix 16) :base 2)))))))
```

```
> (hex-to-bin)
NIL
```

Решение 1.35.2

```
(defun hexa-to-bin (&optional (in "d:/hex.in") (out "d:/bin.out"))
  (with-open-file (s in :direction :input)
    (with-open-file (r out :direction :output :if-exists :supersede)
      (do ((line (read-line s  nil :eof)(read-line s nil :eof)))
          ((eql line :eof))
        (if (zerop (length line))
            (format r "~%")
            (format r "~b~%" (parse-integer line :radix 16)))))))
```

```
> (hex-to-bin)
NIL

bin.out
1010
11000
```

Задача 1.36 ration.lisp

Определить рациональное ли число, если да то возвести его в квадрат если нет, то оставить вернуть это число.

Решение 1.36.1

```
(defun ration (n)
  (if (rationalp n) (expt n 2) n))
```

```
> (ration 7/3)
49/9
> (ration 1.7)
1.7
```

Решение 1.36.2

```
(defun ration (n)
  (if (rationalp n) (expt (float n) 2) n))
```

```
> (ration 7/3)
5.444444
> (ration 1.7)
1.7
```

Решение 1.36.3

```
(defun ration (n)
  (if (rationalp n) (float (expt n 2)) n))
```

```
> (ration 7/3)
5.4444447
> (ration 1.7)
1.7
```

Задача 1.37 extract-nums.lisp

Выбрать все числа из текста и записать их в новую строку.

Решение 1.37.1

```
(defun extract-nums (s)
  (string-trim  "#\(#\)"
                (write-to-string
                 (delete-if-not
                  #'numberp
                  (read-from-string
                   (concatenate 'string "(" s ")"))))))

> (extract-nums "a 1 b 2c 3")
"1 3"
```

Задача 1.38 count-divisors.lisp

Определить функцию, которая принимает целое положительное число и возвращает число делителей этого числа.

Решение 1.38.1

```
(defun count-divisors (n)
  (loop for a from 1 to (truncate (/ n 2))
        when (zerop (rem n a))
        count a))

> (count-divisors 100)
8
> (count-divisors 99)
5
```

Решение 1.38.2

```
(defun count-divisors (n &optional (m (truncate (/ n 2))))
  (cond ((zerop m) 0)
        ((zerop (rem n m)) (1+ (count-divisors n (1- m))))
        ((count-divisors n (1- m)))))

> (count-divisors 100)
8
> (count-divisors 99)
5
```

Решение 1.38.3

```
(defun count-divisors (n &optional (m (truncate (/ n 2))))
  (if (plusp m)
      (if (zerop (rem n m))
          (1+ (count-divisors n (1- m)))
```

```
          (count-divisors n (1- m)))
      0))

> (count-divisors 100)
8
> (count-divisors 99)
5
```

Задача 1.39 adjust-subtitles.lisp

```
                          It is not how much you do, but how much love
                          you put in the doing.
                                                                    ~
```

Внести изменения в субтитры фильма "White Men Can't Jump (1992).txt".

```
1
00:00:37,200 --> 00:00:40,100
Presbyterians are called
"God's frozen people."

2
00:00:40,400 --> 00:00:42,400
Wouldn't swing if you hung them.
```

чтобы получилось:

```
11
00:00:44,200 --> 00:00:47,100
Presbyterians are called
"God's frozen people."

12
00:00:47,400 --> 00:00:49,400
Wouldn't swing if you hung them.
```

Решение 1.39.1

```
(defun adjust-subtitles (path s+ &optional (f+ 0) &aux(n 1))
  (with-open-file (s path :direction :input)
    (with-open-file
        (r "c:/nst/new.txt" :direction :output :if-exists :supersede)
      (do ((line (read-line s  nil :eof)(read-line s nil :eof)))
          ((eql line :eof))
        (cond ((zerop (length line)) (format r "~%"))
              ((equal (format nil "~a" n) line)
               (format r "~a~%" (+ n f+)) (incf n))
              ((char= (elt line 0) #\0)
               (format r "~a~%" (at (at line s+ 6) s+ 23)))
              ((format r "~a~%" line)))))))

(defun at (line s+ pos
```

```
          &aux (z (+ (parse-integer
                      line :start pos :end (+ pos 2)) s+)))
  (cond ((> z 59) (re line (mod z 60) pos)
         (at line (floor z 60) (- pos 3)))
        ((re line z pos))) line)

(defun re (line u pos)
  (replace line (format nil "~2,'0d" u) :start1 pos :end1 (+ pos 2)))

> (adjust-subtitles "c:/nst/wmcj.txt" 7 10)

11
00:00:44,200 --> 00:00:47,100
Presbyterians are called
"God's frozen people."

12
00:00:47,400 --> 00:00:49,400
Wouldn't swing if you hung them.
```

Задача 1.40 decode-phone-n.lisp

```
                                The only way to learn a new programming
                                 language is by writing programs in it.

                                                    Kernighan and Ritchie
```

Написать программу декодирования телефонного номера для АОН. По запросу АОНа АТС посылает телефонный номер, используя следующие правила:
- Если цифра повторяется менее 2 раз, то это помеха и она должна быть отброшена;
- Каждая значащая цифра повторяется минимум 2 раза;
- Если в номере идут несколько цифр подряд, то для обозначения «такая же цифра как предыдущая» используется идущий 2 или более подряд раз знак #;
Например, входящая строка 4434###552222311333661 соответствует номеру 4452136.

Решение 1.40.1

```
(defun decode-phone-n (s)
  (glue (squeeze (cleave s))))

(defun cleave (s)
  (map 'list #'string s))

(defun squeeze (w &optional ac q b z &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (nreverse ac))
        ((or (string/= a b) (equal b q))
         (squeeze d ac b a z))
        ((and (equal a b) (equal b "#"))
         (squeeze d (push z ac) b a z))
```

```
        ((squeeze d (push a ac) b a a))))

(defun glue (v)
  (reduce
   #'(lambda (x &optional y)
       (concatenate 'string x y)) v :initial-value ""))

> (decode-phone-n "4434###552222311333661")
"4452136"
> (decode-phone-n "###4434###552222311333661")
"4452136"
> (decode-phone-n "5###4434###552222311333661")
"4452136"
```

Решение 1.40.2 * (decode-phone-n "22122")

```
(defun decode-phone-n (s)
  (glue (squeeze (cleave s))))

(defun cleave (s)
  (map 'list #'string s))

(defun squeeze (w &optional ac q b z &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (nreverse ac))
        ((or (string/= a b) (equal b q) (equal a z))
         (squeeze d ac b a z))
        ((and (equal a b) (equal b "#"))
         (squeeze d (push z ac) b a z))
        ((squeeze d (push a ac) b a a))))

(defun glue (v)
  (reduce
   #'(lambda (x &optional y)
       (concatenate 'string x y)) v :initial-value ""))

> (decode-phone-n "5###4434###552222311333661")
"4452136"
> (decode-phone-n "22122")
"2"
```

Решение 1.40.3

```
(defun decode-phone-n (s)
  (glue (squeeze (cleave s))))

(defun cleave (s)
  (map 'list #'string s))

(defun squeeze (w &optional ac q b z &aux (a (car w)) (d (cdr w))
                (n (or (string/= a b) (equal b q) (equal a z)))
                (m (and (equal a b) (equal b "#"))))
  (if w (squeeze
         d
```

```
         (cond (n ac)
               (m (push z ac))
               (t (push a ac)))
         b
         a
         (if (or n m) z a))
       (nreverse ac)))

(defun glue (v)
  (reduce
   #'(lambda (x &optional y)
       (concatenate 'string x y)) v :initial-value ""))

> (decode-phone-n "5###4434###552222311333661")
"4452136"
> (decode-phone-n "22122")
"2"
```

Задача 1.41 reverse-digits.lisp

Определить функцию, которая выводит цифры числа в обратном порядке

Решение 1.41.1

```
(defun reverse-digits (n &optional (m 0))
  (cond ((zerop n) m)
        (t (multiple-value-bind (q r) (truncate n 10)
             (reverse-digits q (+ (* m 10) r))))))

> (reverse-digits 1234)
4321
```

Решение 1.41.2 (автор Chris Jester-Young codereview.stackexchange.com)

```
(defun reverse-digits (n)
  (labels ((next (n v)
             (if (zerop n) v
                 (multiple-value-bind (q r)
                     (truncate n 10)
                   (next q (+ (* v 10) r))))))
    (next n 0)))

> (reverse-digits 34567)
76543
```

Решение 1.41.3

```
(defun reverse-digits (n)
  (parse-integer (reverse (write-to-string n))))

> (reverse-digits 1234)
4321
```

4

Задача 1.42 remove-extra-spaces.lisp

В заданной строке слова разделены несколькими пробелами. Удалить все лишние пробелы.

Решение 1.42.1

```
(defun remove-extra (s)
  (format nil "~{~s ~}"
          (read-from-string
           (concatenate
            'string "("
            (delete-if-not #'(lambda (x)
                               (or (alpha-char-p x)
                                   (equal x #\space)
                                   (equal x #\-)))
                           s) ")"))))

(defun remove-extra-spaces (s)
  (string-right-trim " " (remove-extra s)))

> (remove-extra-spaces "aa   bb")
"AA BB"
```

Решение 1.42.2

```
(defun remove-extra-spaces (w)
  (loop for c across w
        unless (and n (char= c (car (last n)) #\space))
        collect c into n
        finally (return (string-trim " " (concatenate 'string n)))))

> (remove-extra-spaces "aa   bb")
"AA BB"
```

Задача 1.43 swap-lets.lisp

Написать программу, которая в каждом слове предложения меняет местами первую и последнюю букву.

Решение 1.43.1

```
(defun cut (s)
  (mapcar #'swap
          (mapcar #'string (read-from-string
                            (concatenate 'string "(" s ")")))))

(defun swap (s &aux (v (concatenate 'list s)))
  (if (cdr v) (nconc (last v) (butlast (cdr v)) `(,(car v))) v))
```

```
(defun glue (w)
  (when w (concatenate 'string (car w) " " (glue (cdr w)))))

(defun swap-lets(s)
  (string-right-trim " " (glue (cut s))))

> (swap-lets "a ab abc")
"A BA CBA"
> (swap-lets "a ab #.(print 'security-hole) abc")
SECURITY-HOLE
"A BA EECURITY-HOLS CBA"
```

Решение 1.43.2

```
(defun cut (s)
  (mapcar #'swap (word (map 'list #'string s))))

(defun word (w &optional acc)
  (cond ((null w) (list (apply #'concatenate 'string (reverse acc))))
        ((equal (car w) " ")
         (cons (apply #'concatenate 'string (reverse acc))
               (word (cdr w) nil)))
        ((word (cdr w) (cons (car w) acc)))))

(defun swap (s &aux (v (concatenate 'list s)))
  (if (cdr v) (nconc (last v) (butlast (cdr v)) `(,(car v))) v))

(defun glue (w)
  (when w (concatenate 'string (car w) " " (glue (cdr w)))))

(defun swap-lets(s)
  (string-right-trim " " (glue (cut s))))

> (swap-lets "a ab abc")
"A BA CBA"
> (swap-lets "a ab #.(print 'security-hole) abc")
"a ba t.(prin# )security-hole' cba"
```

Задача 1.44 factorial-of.lisp

Определить, факториал какого числа равен передаваемому значению.

Решение 1.44.1

```
(defun factorial-of (m &optional (n 1) &aux (f (factorial n)))
  (cond ((> f m) nil)
        ((= f m) n)
        (t (factorial-of m (1+ n)))))

(defun factorial (n)
  (if (zerop n) 1 (* n (factorial (- n 1)))))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Решение 1.44.2

```
(defun factorial-of (m &optional (n 1) &aux (f (factorial n)))
  (when (<= f m) (if (= f m) n (factorial-of m (1+ n)))))

(defun factorial (n)
  (if (zerop n) 1 (* n (factorial (- n 1)))))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Решение 1.44.3

```
(defun factorial-of (m &optional (n 2) &aux (z (/ m n)))
  (cond ((= m 1) 1)
        ((= m 2) 2)
        ((= (1+ n) z) z)
        ((< (1+ n) z) (factorial-of Z (1+ n)))
        (t nil)))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Решение 1.44.4

```
(defun factorial-of (m &optional (n 2) &aux (z (/ m n)))
  (cond ((or (= m 2) (= m 1)) m)
        ((= (1+ n) z) z)
        ((< (1+ n) z) (factorial-of Z (1+ n)))
        (t nil)))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Решение 1.44.5

```
(defun factorial-of (m &optional (n 2) &aux (z (/ m n)))
  (cond ((= m 1) m)
        ((= z 1) n)
        ((< n z) (factorial-of Z (1+ n)))
        (nil)))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Решение 1.44.6

```
(defun factorial-of (m &optional (n 2) &aux (z (/ m n)))
  (cond ((= m 1) m)
        ((= z 1) n)
        ((< n z) (factorial-of Z (1+ n)))))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Решение 1.44.7 (автор - Catstail, www.cyberforum.ru)

```
(defun factorial-of (m &optional (z 1) (n 1))
  (cond ((= z m) n)
        ((> z m) nil)
        ((factorial-of m (* z (1+ n)) (1+ n)))))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
```

```
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Решение 1.44.8

```
(defun factorial-of (m &optional (z 1) (n 1))
  (when (<= z m) (if (= z m) n (factorial-of m (* z (1+ n)) (1+ n)))))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Решение 1.44.9

```
(defun factorial-of (m &optional (z 1) (n 1))
  (unless (> z m) (if (= z m)
                      n
                      (factorial-of m (* z (1+ n)) (1+ n)))))
```

```
> (factorial-of 1)
1
> (factorial-of 2)
2
> (factorial-of 479001600)
12
> (factorial-of 479001601)
NIL
```

Структура 2 (функция ATOM) > (СПИСОК)

Задача 2.1 count-word.lisp

Написать программу, которая по заданному тексту строит список пар: (<слово> <частота встречаемости в тексте>).

Решение 2.1.1

```
(defun count-word (pathname)
  (let ((u)) (with-open-file (s pathname :direction :input)
               (do ((c (read-line s nil :eof) (read-line s nil :eof)))
                   ((eql c :eof))
                 (setq u (nconc (en c) u))))
       (qu u)))
```

```
(defun en (s)
  (read-from-string
   (concatenate
    'string "(" (delete-if-not #'(lambda (x)
                                   (or (alpha-char-p x)
                                       (equal x #\space)
                                       (equal x #\-)))
                               s) ")")))

(defun qu (w)
  (when w (let ((a (car w)) (d (cdr w)))
            (cons (cons (string-downcase a) (count a w))
                  (qu (delete a d))))))

> (count-word "test.txt")
(("less" . 5) ("de-facto" . 1))
```

Решение 2.1.2

```
(defun count-word (pathname)
  (let ((u)) (with-open-file (s pathname :direction :input)
               (do ((c (read-line s nil :eof) (read-line s nil :eof)))
                   ((eql c :eof))
                 (setq u (nconc (en c) u))))
       (symb-str u)))

(defun en (s)
  (read-from-string
   (concatenate
    'string "(" (delete-if-not #'(lambda (x)
                                   (or (alpha-char-p x)
                                       (equal x #\space)
                                       (equal x #\-)))
                               s) ")")))

(defun symb-str (w)
  (let ((u)) (dolist(n w)
               (push (string-downcase n) u))
       (count-str (sort u #'string-greaterp))))

(defun count-str (w)
  (let ((u) (e (car w)) (n 1))
    (dolist (m w)
      (cond ((equal m e) (incf n))
            ((push (cons e n) u) (setq n 1 e m))))
    u))

> (count-word "test.txt")
(("along" . 3) ("already" . 3))
```

Решение 2.1.3

```
(defun count-word (pathname)
  (let ((u)) (with-open-file (s pathname :direction :input)
               (do ((c (read-line s nil :eof) (read-line s nil :eof)))
                   ((eql c :eof))
                 (setq u (nconc (list-symb (drop-noise c)) u))))
       (list-pair (list-str u))))

(defun list-symb (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun drop-noise (s)
  (delete-if-not #'(lambda (x) (or
                                (alpha-char-p x)
                                (equal x #\space)
                                (equal x #\-))) s))

(defun list-str (w)
  (sort (mapcar #'(lambda (a) (string-downcase a)) w) #'string-lessp))

(defun list-pair (w &optional (e (car w)) (n 0) u)
  (mapcan #'(lambda (a) (incf n) (when (string-not-equal a e)
                                   (setq u (cons e n) n 0 e a)
                                   (list u)))
          w))

> (count-word "test.txt")
(("balance" . 1) ("based" . 1))
```

Решение 2.1.4*

```
* mapcan-version

(defun count-word (pathname)
  (let ((u)) (with-open-file (s pathname :direction :input)
               (do ((c (read-line s nil :eof) (read-line s nil :eof)))
                   ((eql c :eof))
                 (setq u (nconc (list-symb (drop-noise c)) u))))
       (list-pair (sort u #'string<))))

(defun list-symb (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun drop-noise (s)
  (delete-if-not #'(lambda (x) (or
                                (alpha-char-p x)
                                (equal x #\space)
                                (equal x #\-))) s))

(defun list-pair (w &optional (e (car w)) (n 0) u)
  (mapcan #'(lambda (a) (incf n) (unless (eql a e)
                                   (setq u (cons e n) n 0 e a)
                                   (list u)))
          w))
```

```
> (count-word "test.txt")
(LEGAL . 1) (LEGALLY . 2)
```

Решение 2.1.5**

```
** maphash-version

(defun count-word (pathname)
  (with-open-file (s pathname :direction :input)
    (loop for line = (read-line s nil nil)
          while line
          nconc (list-symb (drop-noise line)) into words
          finally (return (sort (list-pair words) #'> :key #'cdr)))))

(defun list-symb (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun drop-noise (s)
  (delete-if-not #'(lambda (x) (or
                                (alpha-char-p x)
                                (equal x #\space)
                                (equal x #\-))) s))

(defun list-pair (words &aux (hash (make-hash-table)) acc)
  (dolist (word words) (incf (gethash word hash 0)))
  (maphash #'(lambda (e n) (push `(,e . ,n) acc)) hash) acc)

> (count-word "test.txt")
((ANNEXED . 1))
```

Задача 2.2 run-test.lisp

Есть набранная программа, она записана в отдельном файле. К ней есть тесты. Они записаны в другом файле. Как (что надо в программу добавить или может как-нибудь по другому не меняя саму программу) сделать так, чтоб подгрузились тесты из файла и по нажатию ENTER мы увидели на экране интерпретатора их выполнение? Т.е надо сделать так чтоб тесты не вводились в самом интерпретаторе, а брались из текстового файла и по нажатию ENTER они выполнялись.

Решение 2.2.1

test.txt

```
a a b c
1 1 2 3
```

```
(defun run-test (path)
  (with-open-file (s path :direction :input)
    (do ((line (read-line s  nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
```

```
      (format t "~a~%" (rm (sm line))))))

(defun sm (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun rm (v)
  (remove (car v) (cdr v)))

> (run-test "c:/test.txt")
(B C)
(2 3)
```

Решение 2.2.2

test.txt

```
1 2 3 4 kill 1 2
a b c d kill a b
```

```
(defun run-test (path)
  (with-open-file (s path :direction :input)
    (do ((line (read-line s  nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
      (format t "~a~%" (rm (sm line))))))

(defun sm (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun rm (w &aux (p (position 'kill w)))
  (nset-difference (subseq w 0 p) (subseq w (1+ p))))

> (run-test "c:/test.txt")
(3 4)
(C D)
```

Задача 2.3 balanced.lisp

Решить следующую задачу: вводится массив 4-хзначных чисел, программа выводит числа, у которых сумма первых двух элементов числа равна сумме вторых двух элементов числа. То есть первое число будет 0000 последнее число 9999, а выводит числа такие 0000, 0101, 2525, 2552 и т.д.

Решение 2.3.1

```
(defun xd (n p)
  (parse-integer (format nil "~4,'0d" n) :start p :end (1+ p)))

(defun bd (m)
  (loop for n from 0 to m
        when (eq (+ (xd n 0) (xd n 1)) (+ (xd n 2) (xd n 3)))
        collect n))
```

```
(defun balanced (m)
  (mapcar #'(lambda (a) (format nil "~4,'0d" a)) (bd m)))

> (balanced 9999)
("0000" "0101" "0110" "0202" ... "9788" "9797" "9889" "9898" "9999")
```

Решение 2.3.2

```
(defun xd (n p)
  (parse-integer (format nil "~4,'0d" n) :start p :end (1+ p)))

(defun bd (m &optional (n 0))
  (cond ((> n m) nil)
        ((/= (+ (xd n 0) (xd n 1)) (+ (xd n 2) (xd n 3)))
         (bd m (1+ n)))
        ((cons n (bd m (1+ n))))))

(defun balanced (m)
  (mapcar #'(lambda (a) (format nil "~4,'0d" a)) (bd m)))

(balanced 9999)
("0000" "0101" "0110" "0202" ... "9788" "9797" "9889" "9898" "9999")
```

Задача 2.4 int>digits.lisp

Определить функцию для построения списка цифр числа.

Решение 2.4.1

```
(defun int>digits (n)
  (loop for c across (write-to-string n) collect (digit-char-p c)))

> (int>digits 123)
(1 2 3)
```

Решение 2.4.2

```
(defun int>digits (n &optional ac)
  (if (zerop n) ac (int>digits (truncate n 10) (cons (rem n 10) ac))))

> (int>digits 123)
(1 2 3)
```

Решение 2.4.3

```
(defun int>digits (n)
  (mapcar #'digit-char-p (coerce (write-to-string n) 'list)))

> (int>digits 123)
(1 2 3)
```

Решение 2.4.4

```
(defun int>digits (n)
  (mapcar #'digit-char-p (concatenate 'list (write-to-string n))))

> (int>digits 123)
(1 2 3)
```

Решение 2.4.5

```
(defun int>digits (n)
  (mapcar #'digit-char-p (map 'list #'identity (write-to-string n))))

> (int>digits 123)
(1 2 3)
```

Решение 2.4.6

```
(defun int>digits (n)
  (mapcar #'parse-integer (map 'list #'string (write-to-string n))))

> (int>digits 123)
(1 2 3)
```

Задача 2.5 squares.lisp

Написать функцию построения списка квадратов чисел от 1 до n.

Решение 2.5.1

```
(defun squares (n &optional (m 1))
  (when (>= n m) (cons (* m m) (squares n (1+ m)))))

> (squares 10)
(1 4 9 16 25 36 49 64 81 100)
```

Решение 2.5.2

```
(defun squares (n)
  (loop for x from 1 to n collect (* x x)))

> (squares 10)
(1 4 9 16 25 36 49 64 81 100)
```

Решение 2.5.3

```
(defun squares (n)
  (if (> n 0) (nconc (squares (1- n)) (list (* n n)))))

> (squares 10)
(1 4 9 16 25 36 49 64 81 100)
```

Задача 2.6 prin&list.lisp

Определить функцию (f n), которая сначала печатает строку чисел без пробелов n n-1 … 1, а на второй строке выводит значение в виде списка чисел (n n-1 … 1).

Решение 2.6.1

```
(defun prin&list (n &optional ac)
  (cond ((zerop n) (reverse ac))
        (t (prin1 n) (prin&list (- n 1) (cons n ac)))))

> (prin&list 6)
654321
(6 5 4 3 2 1)
```

Решение 2.6.2

```
(defun pa (n ac)
  (cond ((zerop n) (reverse ac))
        (t (prin1 n) (pa (- n 1) (cons n ac)))))

(defun prin&list (n)
  (pa n nil))

> (prin&list 6)
654321
(6 5 4 3 2 1)
```

Решение 2.6.3

```
(defun prin&list (n)
  (labels ((pa (n ac)
             (cond ((zerop n) (reverse ac))
                   (t (prin1 n) (pa (- n 1) (cons n ac))))))
    (pa n nil)))

> (prin&list 6)
654321
(6 5 4 3 2 1)
```

Решение 2.6.4

```
(defun prin&list (n)
  (let ((r))
    (dotimes (i n r)
      (prin1 (- n i))
      (setq r (cons (+ i 1) r)))))

> (prin&list 6)
654321
(6 5 4 3 2 1)
```

Задача 2.7 deci-bin.lisp

Определить функцию для перевода числа из десятичной системы в двоичную.

Решение 2.7.1

```
(defun deci-bin (n &optional ac)
  (if (zerop n) ac (deci-bin (floor (/ n 2)) (cons (mod n 2) ac))))

> (deci-bin 27)
(1 1 0 1 1)
> (deci-bin 10)
(1 0 1 0)
```

Решение 2.7.2

```
(defun deci-bin (n) (format nil "~b" n))

> (deci-bin 27)
"11011"
> (deci-bin 10)
"1010"
```

Задача 2.8 bin-deci.lisp

Определить функцию для перевода числа (записанного в виде списка) из двоичной системы в десятичную.

Решение 2.8.1

```
(defun bin-deci (w &optional (n 0))
  (if (null w) n (bin-deci (cdr w) (+ (car w) n n))))

> (bin-deci '(1 1 0 1 1))
27
> (bin-deci '(1 0 1 0))
10
```

Задача 2.9 drop-word-duplicates.lisp

В файле записаны слова, некоторые из них повторяются. Требуется считывать слова из этого файла и формировать из них список, в котором повторов не будет. Результат записать в файл.

Решение 2.9.1

```
(defun list-string (s)
  (read-from-string
   (concatenate
    'string "(" (delete-if-not #'(lambda (x)
```

```
                                    (or (alpha-char-p x)
                                        (equal x #\space)
                                        (equal x #\-)))
                                s) ")")))

(defun drop-word-duplicates (path-in path-out)
  (let ((u))
    (with-open-file (s path-in :direction :input)
      (with-open-file
          (r path-out :direction :output :if-exists :supersede)
        (do ((line (read-line s nil :eof)(read-line s nil :eof)))
            ((eql line :eof))
          (setq u (nconc (list-string line) u)))
        (format r "~a" (remove-duplicates u))))))

> (drop-word-duplicates "file-in.txt" "file-out.txt")
NIL

file-in.txt
aaa bbb aaa bbb

file-out.txt
(AAA BBB)
```

Задача 2.10 prime2n-1.lisp

Составить программу, вычисляющую 2n-1=N (где n степень числа 2) и проверяющую является ли N простым числом. Число n меняется от 0 до 100. Вывести все n при которых N простое.

Решение 2.10.1

```
(defun primep (n)
  (loop for i from 3 to (isqrt n) by 2 never (zerop (rem n i))))

(defun lucas-lehmer (n)
  (loop with m = (1- (expt 2 n))
        for i from 1 to (1- n)
        for j = 4 then (rem (- (expt j 2) 2) m)
        finally (return (zerop j))))

(defun prime2n-1 (m &optional (v '(2)) (n 2))
  (cond ((= m 2) (reverse v))
        ((and (primep n) (lucas-lehmer n))
         (prime2n-1 (1- m) (cons n v) (1+ n)))
        ((prime2n-1 (1- m) v (1+ n)))))

> (prime2n-1 100)
(2 3 5 7 13 17 19 31 61 89)
```

Задача 2.11 count-letters.lisp

Дан текст из латинских букв и знаков препинания. Составить программу частотного анализа букв этого текста, т.е. напечатать каждую букву с указанием количества ее вхождений.

Решение 2.11.1

```
(defun count-letters (pathname)
  (with-open-file (s pathname :direction :input)
    (loop for line = (read-line s nil nil)
          while line
          nconc (list-symb (drop-noise line)) into words
          finally (return (sort (list-pair words) #'> :key #'cdr)))))

(defun list-symb (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun drop-noise (s)
  (delete-if-not
   #'(lambda (x) (or (alpha-char-p x)
                     (equal x #\space)))
   s))

(defun list-pair (words &aux (hash (make-hash-table)) acc)
  (dolist (word words) (incf (gethash word hash 0)))
  (maphash #'(lambda (e n) (push `(,e . ,n) acc)) hash) acc)

> (count-letters "test.txt")
((A . 2) (E . 1) (V . 1) (S . 1))
```

Задача 2.12 print-linen.lisp

Определите функцию (f n), печатающую n раз квадрат 2 x 2 звездочками (*) с промежутком 2 знака.

Решение 2.12.1

```
(defun print-linen (n)
  (dotimes (i n) (format t "*  *   "))
  (format t "~%")
  (dotimes (i n) (format t "*  *   ")))

> (print-linen 4)
*  *   *  *   *  *   *  *
*  *   *  *   *  *   *  *
```

Задача 2.13 odd50.lisp

Опишите функцию, которая печатает все нечетные числа от 0 до 100 в порядке возрастания, по одному на строчке.

Решение 2.13.1

```
(defun odd50 (&optional (n 1))
  (when (< n 100) (format t "~a~&" n) (odd50 (+ n 2))))

> (odd50)
1
3
...
99
NIL
```

Задача 2.14 f.lisp

Определить функцию без параметров, конструирующую список (A B ((C D) E)) из одноуровневых списков и атомов, которые вводятся с клавиатуры.

Решение 2.14.1

```
(defun f ()
  `(,(read) ,(read) (,(read) ,(read))))

> (f)
1
2
(3 4)
5
(1 2 ((3 4) 5))
```

Решение 2.14.2

```
(defun |[1 2 [[3 4] 5]]| ()
  `(,(read) ,(read) (,(read) ,(read))))

> (|[1 2 [[3 4] 5]]|)
1
2
(3 4)
5
(1 2 ((3 4) 5))
```

Решение 2.14.3

```
(defun [12[[34]5]] ()
  `(,(read) ,(read) (,(read) ,(read))))

> ([12[[34]5]])
1
2
(3 4)
5
```

```
(1 2 ((3 4) 5))
```

Решение 2.14.4

```
(defun [ab[[cd]e]] ()
  `(,(read) ,(read) (,(read) ,(read))))
```

```
> ([ab[[cd]e]])
1
2
(3 4)
5
(1 2 ((3 4) 5))
```

Задача 2.15 magic-ticket-9999.lisp

Найти номера "счастливых" билетов (сумма первых двух цифр равна сумме последних двух цифр) с номерами от 0000 до 9999 включительно.

Решение 2.15.1

```
(defun magic-ticket-9999 (n)
  (row-ticket (add-nine n) n))

(defun row-ticket (nn n &aux (m (int-digit nn)) (z (add-zero m n)))
  (when (>= nn 0) (if (check-balance z (/ n 2))
                      (cons z (row-ticket (1- nn) n))
                      (row-ticket (1- nn) n))))

(defun add-nine (n &optional w)
  (if (< (length w) n) (add-nine n (cons 9 w)) (list-num w)))

(defun list-num (w)
  (parse-integer (apply #'concatenate 'string
                        (mapcar #'write-to-string w))))

(defun add-zero (w n)
  (if (< (length w) n) (add-zero (cons 0 w) n) w))

(defun int-digit (n &optional ac)
  (if (zerop n) ac (int-digit (truncate n 10) (cons (rem n 10) ac))))

(defun check-balance (w n)
  (= (apply #'+ (subseq w 0 n)) (apply #'+ (nthcdr n w))))
```

```
> (time (magic-ticket-9999 4))
Real time: 0.0624001 sec.
Run time: 0.0624004 sec.
Space: 487976 Bytes
GC: 1, GC time: 0.0 sec.
((9 9 9 9) (9 8 9 8) (9 8 8 9)...(0 1 1 0) (0 1 0 1) (0 0 0 0))
```

Задача 2.16 magic-ticket-222222.lisp

Найти число "счастливых" билетов (сумма первых трех цифр равна сумме последних трех цифр) с номерами от 000000 до 222222 включительно.

Решение 2.16.1

```
(defun magic-ticket-222222 (n)
  (row-ticket (add-two n) n))

(defun row-ticket (nn n &aux (m (int-digit nn)) (z (add-zero m n)))
  (when (>= nn 0) (if (check-balance z (/ n 2))
                      (cons z (row-ticket (1- nn) n))
                      (row-ticket (1- nn) n))))

(defun add-two (n &optional w)
  (if (< (length w) n) (add-two n (cons 2 w)) (list-num w)))

(defun list-num (w)
  (parse-integer (apply #'concatenate 'string
                        (mapcar #'write-to-string w))))

(defun add-zero (w n)
  (if (< (length w) n) (add-zero (cons 0 w) n) w))

(defun int-digit (n &optional ac)
  (if (zerop n) ac (int-digit (truncate n 10) (cons (rem n 10) ac))))

(defun check-balance (w n)
  (= (apply #'+ (subseq w 0 n)) (apply #'+ (nthcdr n w))))
```

```
> (magic-ticket-222222 6)
((2 2 2 2 2 2) (2 2 2 2 1 3) (2 2 2 2 0 4) (2 2 2 1 5 0) (2 2 2 1 4 1)
 (2 2 2 1 3 2) (2 2 2 1 2 3) (2 2 2 1 1 4) (2 2 2 1 0 5) (2 2 2 0 6 0)
…
(0 0 3 0 3 0) (0 0 3 0 2 1) (0 0 3 0 1 2) (0 0 3 0 0 3) (0 0 2 2 0 0)
(0 0 2 1 1 0) (0 0 2 1 0 1) (0 0 2 0 2 0) (0 0 2 0 1 1) (0 0 2 0 0 2)
(0 0 1 1 0 0) (0 0 1 0 1 0) (0 0 1 0 0 1) (0 0 0 0 0 0))
```

Задача 2.17 sum-20.lisp

Найти все пары чисел в диапазоне от 1 до 20, сумма которых равна 20.

Решение 2.17.1

```
(defun sum-20 ()
  (loop for a from 1 to 19
        for b from 19 downto 1
        collect `(,a ,b)))
```

```
> (sum-20)
((1 19) (2 18) (3 17) (4 16) (5 15) (6 14) (7 13) (8 12) (9 11) (10
```

```
10) (11 9) (12 8) (13 7) (14 6) (15 5) (16 4) (17 3) (18 2) (19 1))
```

Задача 2.18 sum-n.lisp

Найти все пары чисел в диапазоне от 1 до n, сумма которых равна n.

Решение 2.18.1

```
(defun sum-n (n)
  (loop for a from 1 to (1- n)
        for b from (1- n) downto 1
        collect `(,a ,b)))

> (sum-n 20)
((1 19) (2 18) (3 17) (4 16) (5 15) (6 14) (7 13) (8 12) (9 11) (10
10) (11 9) (12 8) (13 7) (14 6) (15 5) (16 4) (17 3) (18 2) (19 1))
```

Задача 2.19 onion-n.lisp

Определите функцию, которая создает на n - уровне вложенный список, элементом которого на самом нижнем уровне является n.

Решение 2.19.1

```
(defun onion-n (n &optional (m n))
  (if (zerop n) m (list (onion-n (1- n) m))))

> (onion-n 7)
(((((((7)))))))
```

Решение 2.19.2

```
(defun onion-n (n &optional (m n))
  (if (zerop n) m `(,(onion-n (1- n) m))))

> (onion-n 7)
(((((((7)))))))
```

Решение 2.19.3

```
(defun onion-n (n &optional (m n))
  (if (zerop n)
      m
      (cons (onion-n (1- n) m) nil)))

> (onion-n 7)
(((((((7)))))))
```

Задача 2.20 factorials.lisp

Построить список, элементами которого являются значения факториала от

1 до n.

Решение 2.20.1

```
(defun factorial (n)
  (if (= n 0) 1 (* n (factorial (- n 1)))))

(defun factorials (n)
  (loop for a from 1 to n collect (factorial a)))

> (factorials 5)
(1 2 6 24 120)
```

Решение 2.20.2

```
(defun factorial (n)
  (if (= n 0) 1 (* n (factorial (- n 1)))))

(defun factorials (n)
  (when (> n 0) (cons (factorial n) (factorials (1- n)))))

> (factorials 5)
(120 24 6 2 1)
```

Решение 2.20.3

```
(defun factorial (n)
  (if (= n 0) 1 (* n (factorial (- n 1)))))

(defun factorials (n &optional ac)
  (if (> n 0)
      (factorials (1- n) (cons (factorial n) ac))
      ac))

> (factorials 5)
(1 2 6 24 120)
```

Решение 2.20.4

```
(defun factorial (n)
  (if (= n 0) 1 (* n (factorial (- n 1)))))

(defun factorials (n &optional ac)
  (cond ((zerop n) ac)
        ((factorials (1- n) (cons (factorial n) ac)))))

> (factorials 5)
(1 2 6 24 120)
```

Задача 2.21 dice-game.lisp

Написать простой вариант игры в кости, в котором бросаются две прави-

льные кости. Если сумма выпавших очков равна 7 или 11 - выигрыш. Если выпало (1,1) или (6,6) - игрок получает право снова бросить кости , во всех остальных случаях ход переходит ко второму игроку, но запоминается сумма выпавших очков. Если второй игрок не выигрывает абсолютно, то выигрывает тот игрок, у которого больше очков. Результат игры и значения выпавших костей выводить на экран.

Решение 2.21.1

```
(defun dice (&optional (d 0)
             &aux (a (random 6)) (b (random 6)) (s (+ a b)))
  (cond ((or (= a b 1) (= a b 6))
         (print (list a '+ b '= s)) (dice (+ d s)))
        ((or (= s 7) (= s 11)) (print (list a '+ b '= s)) s)
        (t (print (list a '+  b '= (if (zerop d) s (+ d s))))
           (if (zerop d) s (+ d s)))))

(defun dice-game (&aux (n (dice)) (m (dice)))
  (cond ((or (= n 7) (= n 11) (> n m)) 'you-win)
        ((or (= m 7) (= m 11) (< n m)) 'he-wins)
        (t 'even)))

> (dice-game)
(5 + 2 = 7)
(5 + 4 = 9)
YOU-WIN
```

Решение 2.21.2

```
(defun prnt (a b s)
  (print (list a '+ b '= s)))

(defun dice (&optional (d 0)
             &aux (a (random 6)) (b (random 6)) (s (+ a b)))
  (cond ((or (= a b 1) (= a b 6)) (prnt a b s) (dice (+ d s)))
        ((or (= s 7) (= s 11))(prnt a b s) s)
        (t (let ((m (if (zerop d) s (+ d s)))) (prnt a b m) m))))

(defun dice-game (&aux (n (dice)) (m (dice)))
  (cond ((or (= n 7) (= n 11) (> n m)) 'you-win)
        ((or (= m 7) (= m 11) (< n m)) 'he-wins)
        (t 'even)))

> (dice-game)
(1 + 1 = 2)
(5 + 1 = 8)
(2 + 2 = 4)
YOU-WIN
```

Задача 2.22 rand.lisp

Реализовать функцию которая порождает по заданному N список, состоящий

из N случайных натуральных чисел

Решение 2.22.1

```
(defun rand (n)
  (if (zerop n) nil (cons (random 101) (rand (1- n)))))

> (rand 10)
(39 86 7 68 81 6 34 34 44 97)
```

Решение 2.22.2

```
(defun rand (n)
  (loop for i from 1 to n collect (random 101)))

> (rand 10)
(39 86 7 68 81 6 34 34 44 97)
```

Задача 2.23 cumulate.lisp

Написать программу, вводящую данные из файла, разделив содержание нечетных и четных строк.

```
d:/test.txt:
1 1
a a
2 2
b b
```

Решение 2.23.1

```
(defun cumulate (path &aux (n 1) a b)
    (with-open-file (s path :direction :input)
      (do ((line (read-line s  nil :eof)(read-line s nil :eof)))
          ((eql line :eof))
        (if (evenp n)
            (progn (setf n (1+ n)) (setf a (nconc a (sw line))))
            (progn (setf n (1+ n)) (setf b (nconc b (sw line)))))))
    (values a b))

(defun sw (s)
  (read-from-string (concatenate 'string "(" s ")")))

> (cumulate "d:/test.txt")
(1 1 2 2)
(A A B B)
```

Задача 2.24 divisors.lisp

Определить функцию, которая принимает целое положительное число и возвращает список делителей этого числа.

Решение 2.24.1

```
(defun divisors (n)
  (loop for a from 1 to (truncate (/ n 2))
        when (zerop (rem n a))
        collect a))

> (divisors 100)
(1 2 4 5 10 20 25 50)
> (divisors 99)
(1 3 9 11 33)
```

Решение 2.24.2

```
(defun divisors (n &optional (m (truncate (/ n 2))))
  (cond ((zerop m) nil)
        ((zerop (rem n m)) (cons m (divisors n (1- m))))
        ((divisors n (1- m)))))

> (divisors 100)
(50 25 20 10 5 4 2 1)
> (divisors 99)
(33 11 9 3 1)
```

Решение 2.24.3

```
(defun divisors (n &optional (m (truncate (/ n 2))))
  (when (plusp m) (if (zerop (rem n m))
                      (cons m (divisors n (1- m)))
                      (divisors n (1- m)))))

> (divisors 100)
(50 25 20 10 5 4 2 1)
> (divisors 99)
(33 11 9 3 1)
```

Задача 2.25 suffixes.lisp

Определить функцию, которая строит список суффиксов строки.

Решение 2.25.1

```
(defun suffixes (s)
  (mapcar #'(lambda (e) (coerce e 'string))
          (cdr (loop for d on
                     (loop for a across s
                           collect a)
                     by #'cdr collect d))))

> (suffixes "abc")
("bc" "c")
```

Задача 2.26 bin-deci.lisp

Определить функцию для перевода числа из двоичной системы в десятичную.

Решение 2.26.1

```
(defun bin-deci (n)
  (parse-integer (write-to-string n) :radix 2))

> (bin-deci 1010)
10
4
```

*Задача 2.27 fn>*123.lisp*

Определить функцию, результатом которой будет выражение, являющееся факториалом числа, и в котором числа (сомножители) упорядочены в порядке возрастания.

Решение 2.27.1

```
(defun fn>*123 (n &optional ac)
  (if (zerop n) (cons '* ac) (fn>*123 (1- n) (cons n ac))))

> (fn>*123 10)
(* 1 2 3 4 5 6 7 8 9 10)
> (eval (fn>*123 10))
3628800
```

Задача 2.28 res.lisp

Определить функцию, формирующую список студентов – имя, год рождения, адрес, средний балл.

Решение 4.28.1

```
(defun res (n)
  (when (> n 0)
    (cons (list (read) (read) (read) (read))
          (res (1- n)))))

> (res 3)
ivanov
1992
pushkina-10
4
petrov
1992
gogolya-20
5
```

```
semenov
1992
esenina-10
4
((IVANOV 1992 PUSHKINA-10 4) (PETROV 1992 GOGOLYA-20 5)
 (SEMENOV 1992 ESENINA-10 4))
```

Задача 2.29 amicable-nums.lisp

Два натуральных числа называются дружественными, если сумма всех делителей числа первого равна второму, а сумма всех делителей второго числа равна первому числу. Найдите пары дружественных чисел меньших 1000.

Решение 2.29.1

```
(defun sum-divisors (n)
  (loop for a from 1 to (truncate (/ n 2))
        when (zerop (rem n a))
        sum a))

(defun amicable-nums (m)
  (loop for n from 2 to m when (= (sum-divisors (sum-divisors n)) n)
        collect (list n (sum-divisors n))))
```

```
> (amicable-nums 15000)
((6 6) (28 28) (220 284) (284 220) (496 496) (1184 1210) (1210 1184)
(2620 2924) (2924 2620) (5020 5564) (5564 5020) (6232 6368) (6368
6232) (8128 8128) (10744 10856) (10856 10744) (12285 14595) (14595
12285))
```

Решение 2.29.2

```
(defun sum-divisors (n)
  (loop for a from 1 to (truncate (/ n 2))
        when (zerop (rem n a))
        sum a))

(defun amicable-nums (m)
  (clear-amicable
   (loop for n from 2 to m when (= (sum-divisors (sum-divisors n)) n)
         collect (list n (sum-divisors n)))))

(defun clear-amicable (w &optional ac &aux (a (sort (car w) #'<)))
  (cond ((null w) (nreverse ac))
        ((or (= (car a) (cadr a)) (member a ac :test #'equalp))
         (clear-amicable (cdr w) ac))
        ((clear-amicable (cdr w) (cons a ac)))))
```

```
> (amicable-nums 15000)
((220 284) (1184 1210) (2620 2924) (5020 5564) (6232 6368) (10744
10856) (12285 14595))
```

Решение 2.29.3

```
(defun sum-divisors (n)
  (loop for a from 1 to (truncate (/ n 2))
        when (zerop (rem n a))
        sum a))

(defun amicable-nums (m)
  (clear-amicable
   (loop for n from 2 to m when (= (sum-divisors (sum-divisors n)) n)
         collect (list n (sum-divisors n)))))

(defun clear-amicable (w &optional ac &aux (a (sort (car w) #'<)))
  (cond ((null w) (nreverse ac))
        ((= (car a) (cadr a)) (clear-amicable (cdr w) ac))
        ((clear-amicable (cdr w) (pushnew a ac :test #'equalp)))))

> (amicable-nums 15000)
((220 284) (1184 1210) (2620 2924) (5020 5564) (6232 6368) (10744
10856) (12285 14595))
```

Решение 2.29.4

```
(defun sum-divisors (n)
  (loop for a from 1 to (truncate (/ n 2))
        when (zerop (rem n a))
        sum a))

(defun amicable-nums (m)
  (remove-duplicates
   (mapcar #'(lambda (a) (sort a #'<))
           (remove-if #'(lambda (e) (= (car e) (cadr e)))
                      (loop for n from 2 to m
                            when (= (sum-divisors (sum-divisors n)) n)
                            collect (list n (sum-divisors n)))))
   :test #'equalp))

> (amicable-nums 15000)
((220 284) (1184 1210) (2620 2924) (5020 5564) (6232 6368) (10744
10856) (12285 14595))
```

* Пары дружественных чисел, меньших 100 000:

220 и 284 (Пифагор, около 500 до н. э.)
1184 и 1210 (Паганини, 1866)
2620 и 2924 (Эйлер, 1747)
5020 и 5564 (Эйлер, 1747)
6232 и 6368 (Эйлер, 1750)
10744 и 10856 (Эйлер, 1747)
12285 и 14595 (Браун, 1939)
17296 и 18416 (Ибн ал-Банна, около 1300, Фариси, около 1300, Ферма, Пьер, 1636)
63020 и 76084 (Эйлер, 1747)

66928 и 66992 (Эйлер, 1750)
67095 и 71145 (Эйлер, 1747)
69615 и 87633 (Эйлер, 1747)
79750 и 88730 (Рольф (Rolf), 1964)

Задача 2.30 n-234.lisp

Определить функцию, которая для произвольного целого n создает список из трех элементов, а именно: квадрата, куба и четвертой степени числа n.

Решение 2.30.1

```
defun n-234 (n)
  (list (* n n) (* n n n) (* n n n n)))

> (n-234 2)
(4 8 16)
```

Решение 2.30.2

```
(defun n-234 (n)
  (list (expt n 2) (expt n 3) (expt n 4)))

> (n-234 2)
(4 8 16)
```

Решение 2.30.3

```
(defun n-234 (n)
  (mapcar #'(lambda (a) (expt n a)) '(2 3 4)))

> (n-234 2)
(4 8 16)
```

Решение 2.30.4

```
(defun n-234 (n &rest w)
  (mapcar #'(lambda (a) (expt n a)) w))

> (n-234 2 2 3 4)
(4 8 16)
```

Решение 2.30.5

```
(defun n-234 (n)
  (loop for a in '(2 3 4) collect (expt n a)))

> (n-234 2)
(4 8 16)
```

Решение 2.30.6

```
(defun n-234 (n &rest w)
  (loop for a in w collect (expt n a)))

> (n-234 2 2 3 4)
(4 8 16)
```

Решение 2.30.7

```
(defun n-234 (n m)
  (when (> m 1) (nconc (n-234 n (1- m)) (list (expt n m)))))

> (n-234 2 4)
(4 8 16)
```

Решение 2.30.8

```
(defun n-234 (n m)
  (when (> m 1) (cons (expt n m) (n-234 n (1- m)))))

> (n-234 2 4)
(16 8 4)
```

Структура 3 (функция (СПИСОК)) > АТОМ

Задача 3.1 count-elm.lisp

Подсчитать количество елементов в списке.

Решение 3.1.1

```
(defun count-elm (a w &optional (n 0))
  (cond ((null w) n)
        ((equalp (car w) a) (count-elm a (cdr w) (1+ n)))
        (t (count-elm a (cdr w) n))))

> (count-elm 'b '(a b b a b a b))
4
```

Решение 3.1.2

```
(defun count-elm (a w &optional (n 0))
  (if w
      (count-elm a
                 (cdr w)
                 (if (equalp (car w) a)
                     (1+ n)
                     n))
      n))
```

```
> (count-elm 'b '(a b b a b a b))
4

Решение 3.1.3

(defun count-elm (a w)
  (loop for e in w when (equalp e a) count e))

> (count-elm 'b '(a b b a b a b))
4

Решение 3.1.4

(defun count-elm (a w) (count a w))

> (count-elm 'b '(a b b a b a b))
4
```

Задача 3.2 sum137.lisp

```
Дан список. Найти сумму первого, третьего и седьмого элементов списка,
если указанные элементы - числа, иначе - вернуть последний элемент
списка.

Решение 3.2.1

(defun sum137 (w)
  (if (and (numberp (nth 0 w))
           (numberp (nth 2 w))
           (numberp (nth 6 w)))
      (+ (nth 0 w) (nth 2 w) (nth 6 w))
      (car (last w))))

> (sum137 '(1 2 3 4 5 6 7))
11

Комментарии (построчно):
 1. Определяем функцию/defun sum137 с одним параметром w - список;
 2. Если/if одновременно и/and числом/numberp является первый элемент
списка w/(nth 0 w) (нумерация в списке начинается с 0),
 3. Третий элемент списка w/(nth 2 w),
 4. Седьмой элемент спискаw /(nth 6 w),
 5. То возвращается сумма/+ этих трех элементов/(+ (nth 0 w) (nth 2 w)
(nth 6 w));
 6. В случае невыполнения условия в строках 2-4: возвращается первый
элемент/car списка, состоящего из последнего элемента/last списка w:
(car (last w)), то есть (last '(1 2 3 4 5 6 7)) вернет список из одно-
го (последнего) элемента (7), и затем (car '(7)) вернет 7.

Решение 3.2.2

(defun sum137 (w)
```

```
  (if (and (numberp (first w))
           (numberp (third w))
           (numberp (seventh w)))
      (+ (first w) (third w) (seventh w))
      (car (last w))))

> (sum137 '(1 2 3 4 5 6 7))
11
```

Решение 3.2.3

```
(defun sum137 (w)
  (if (and (numberp (car w))
           (numberp (caddr w))
           (numberp (seventh w)))
      (+ (car w) (caddr w) (seventh w))
      (car (last w))))

> (sum137 '(1 2 3 4 5 6 7))
11
```

Решение 3.2.4

```
(defun sum137 (w &aux (v (list (car w) (caddr w) (seventh w))))
  (if (every #'numberp v) (reduce #'+ v) (car (last w))))

> (sum137 '(1 2 3 4 5 6 7))
11
```

Решение 3.2.5

```
(defun sum137 (w)
  (if (and (numberp (elt w 0))
           (numberp (elt w 2))
           (numberp (elt w 6)))
      (+ (elt w 0) (elt w 2) (elt w 6))
      (car (last w))))

> (sum137 '(1 2 3 4 5 6 7))
11
```

Задача 3.3 sum-odd-plus.lisp

Дан список. Составить программу, вычисляющую сумму всех положительных нечетных чисел.

Решение 3.3.1

```
(defun sum-odd-plus (w)
  (cond ((null w) 0)
        ((and (oddp (car w)) (plusp (car w)))
         (+ (car w) (sum-odd-plus (cdr w))))
```

```
        ((sum-odd-plus (cdr w)))))

> (sum-odd-plus '(0 5 6 7 8))
12
```

Задача 3.4 min-char.lisp

В списке из атомов-символов выбрать символ, код которого наименьший.

Решение 3.4.1

```
(defun min-char (w)
  (nth-value 0 (intern
                (string
                 (code-char
                  (car (sort (list-code (list-char w)) #'<)))))))

(defun list-char (w)
  (mapcar #'(lambda (a) (character a)) w))

(defun list-code (w)
  (mapcar #'(lambda (a) (char-code a)) w))

> (min-char '(b c a))
A
```

Решение 3.4.2

```
(defun min-char (w)
  (nth-value 0 (intern (car (sort (list-str w) #'string-lessp)))))

(defun list-str (w)
  (mapcar #'(lambda (a) (string a)) w))

> (min-char '(b c a))
A
```

Решение 3.4.3

```
(defun min-char (w)
  (nth-value 0 (intern (string (car (sort (list-char w) #'char<))))))

(defun list-char (w)
  (mapcar #'(lambda (a) (character a)) w))

> (min-char '(b c a))
A
```

Решение 3.4.4

```
(defun min-char (w)
  (car (sort w #'string<)))
```

```
(min-char '(b c a))
A
```

Задача 3.5 rare-num.lisp

Найти наименее часто встречающееся число в списке.

Решение 3.5.1

```
(defun rare-num (w)
  (when w (rar w (car w) (count (car w) w))))

(defun rar (w n f)
  (if w (let* ((a (car w)) (c (count a w)))
          (if (<= c f)
              (rar (delete a w) a c)
              (rar (delete a w) n f)))
      n))

> (rare-num '(2 1 2))
1
```

Решение 3.5.2

```
(defun rare-num (w &optional n (f (length w)))
  (if w (let* ((a (car w)) (c (count a w)))
          (if (<= c f)
              (rare-num (delete a w) a c)
              (rare-num (delete a w) n f)))
      n))

> (rare-num '(2 1 2))
1
```

Решение 3.5.3

```
(defun rare-num (w)
  (caar (sort (mapcar #'(lambda (a)
                          (cons a (count a w))) w) #'< :key #'cdr)))

> (rare-num '(2 1 2))
1
```

Решение 3.5.4

```
(defun rare-num (w &aux acc)
  (dolist (a w) (push (cons a (count a w)) acc))
  (caar (sort acc #'< :key #'cdr)))

> (rare-num '(2 1 2))
1
```

Решение 3.5.5

```
(defun rare-num (w)
  (caar (sort (loop for a in w collect
                    (cons a (count a w)))#'< :key #'cdr)))

> (rare-num '(2 1 2))
1
```

Задача 3.6 subp.lisp

Написать программу, проверяющую, является ли первое из двух (заданных в виде списков) множеств подмножеством второго.

Решение 3.6.1

```
(defun subp (v w)
  (cond ((null v))
        ((member (car v) w) (subp (cdr v) w))
        (nil)))

> (subp '(0) '(1 2))
NIL
> (subp '(1) '(1 2))
T
```

Функция вернет t - истину, когда будет выполнено условие (null v) - исчерпан первый список, и до этого не войдет в третью строчку и не вернет nil - ложь.

Решение 3.6.2

```
(defun subp (v w)
  (or (null v) (and (member (car v) w) (subp (cdr v) w))))

> (subp '(0) '(1 2))
NIL
> (subp '(1) '(1 2))
T
```

Решение 3.6.3

```
(defun subp (v w)
  (or (null v) (when (-member (car v) w) (subp (cdr v) w))))

(defun -member (a w)
    (cond ((null w) nil)
    ((equalp a (car w)) (cons (car w) (cdr w)))
    ((-member a (cdr w)))))

> (subp '(0) '(1 2))
```

```
NIL
> (subp '(1) '(1 2))
T
```

Решение 3.6.4

```
(defun subp (v w)
  (cond ((null v))
        ((-member (car v) w) (subp (cdr v) w))))

(defun -member (a w)
    (cond ((null w) nil)
    ((equalp a (car w)) (cons (car w) (cdr w)))
    ((-member a (cdr w)))))

> (subp '(0) '(1 2))
NIL
> (subp '(1) '(1 2))
T
```

Задача 3.7 uno-element.lisp

Если для построения списка и его подсписков был использован только 1 элемент, выдать его, иначе выдать "нет". Пример:

```
> (f '(a (a) (a a a) (((a))))
a
```

Решение 3.7.1

```
(defun uno-element (w)
  (cond ((null w) nil)
        ((listp (car w)) (uno-element (car w)))
        ((car (un w (list (car w)))))))

(defun un (w acc)
  (cond ((null w) acc)
        ((atom w) (if (member w acc)
                      (cons w acc)
                      (return-from un nil)))
        ((un (car w) (un (cdr w) acc)))))

> (uno-element '(a (a) (a a a) (((a)))))
A
```

Задача 3.8 leader-atom.lisp

Написать функцию, формальным параметром которой является список. Результат функции - первый атом списка (в учет принимаются списки всех уровней).

Решение 3.8.1

```
(defun leader-atom (w)
  (cond ((null w) nil)
        ((listp (car w)) (or (leader-atom (car w))
                             (leader-atom (cdr w))))
        ((car w))))

> (leader-atom '((()) (()((a)) (b c)) d))
A
```

Решение 3.8.2

```
(defun leader-atom (w &aux (a (car w)))
  (when w (if (listp a)
              (or (leader-atom a) (leader-atom (cdr w)))
              a)))

> (leader-atom '((()) (()((a)) (b c)) d))
A
```

Задача 3.9 num-sym-3rd.lisp

Написать функцию f (x y z) которая проверяет: является ли третий элемент списка целым числом или символом.

Решение 3.9.1

```
(defun num-sym-3rd (w &aux (a (caddr w)))
  (cond ((null a) nil)
        ((symbolp a) (format nil "~a is a ~a" a "symbol"))
        ((integerp a) (format nil "~a is an ~a" a "integer"))
        ((format nil "~a is ~a" a "unknown"))))

> (num-sym-3rd '(a b c))
"C is a symbol"
> (num-sym-3rd '(1 2 3))
"3 is an integer"
```

Задача 3.10 some-odd.lisp

Функция возвращает t, если в одноуровневом списке есть хотя бы одно нечетное число.

Решение 3.10.1

```
(defun some-odd (w)
  (some #'(lambda (a) (and (numberp a) (oddp a))) w))

> (some-odd '(2 3))
T
```

```
> (some-odd '(2 4))
NIL
```

Задача 3.11 last-odd.lisp

Написать рекурсивную функцию, которая возвращает последнее нечетное число из числового списка.

Решение 3.11.1

```
(defun last-odd (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (cond ((null w) n)
        ((oddp a) (last-odd d a))
        ((last-odd d n))))

(defun last-odd (w)
  (find-if #'oddp w :from-end t))

> (last-odd '(1 4 2 6 3 5 6 4 16))
5
```

Задача 3.12 best-marks.lisp

Дана база данных:
((a 5 5 5) (b 5 4 4) (c 4 4 3)), где
(a 5 5 5) - 1 ученик
(b 5 4 4) - 2 ученик
(c 4 4 3) - 3 ученик.
Определить количество отличников.

Решение 3.12.1

```
(defun best-marks (w &optional (n 0) &aux (a (car w)))
  (cond ((null w) n)
        ((= 5 (cadr a) (caddr a) (cadddr a))
         (best-marks (cdr w) (+ n 1)))
        ((best-marks (cdr w) n))))

> (best-marks '((a 5 5 5) (b 5 4 4) (c 4 4 3)))
1
```

Задача 3.13 average-height.lisp

Имеется список студентов группы, в котором указаны фамилия студента, дата рождения, пол, рост и вес. Определить средний рост студентов мужского пола.

Решение 3.13.1

```
(defparameter w '((ivanov 92 m 180 80) (petrova 92 f 160 60) (petrov
92 m 190 90)))
```

```
(defun average-height (w)
  (let ((v (remove-if-not #'(lambda (a) (equal (caddr a) 'm)) w)))
    (/ (reduce #'+ v :key #'cadddr) (length v))))

> (average-height w)
185
```

Задача 3.14 max-height.lisp

Имеется список студентов группы, в котором указаны фамилия студента, дата рождения, пол, рост и вес. Определить фамилию самого высокого юноши.

Решение 3.14.1

```
(defparameter w '((ivanov 92 m 180 80) (petrova 92 f 160 60) (petrov
92 m 190 90)))

(defun max-height (w)
  (caar (sort w #'> :key #'cadddr)))

> (max-height w)
PETROV
```

Задача 3.15 counter.lisp

Определить функцию для подсчета количества вершин бинарного дерева значения которых лежат в определённом диапазоне.

Решение 3.15.1

```
(defun counter (x z w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 0)
        ((listp a) (+ (counter x z a) (counter x z d)))
        ((and a (<= x a z)) (1+ (counter x z d)))
        ((counter x z d))))

> (counter 1 3 '((2 (5 (nil 3 1) 7))))
3
```

Решение 3.15.2

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun counter (x z w)
  (loop for a in (flat w) counting (<= x a z)))
```

```
> (counter 1 3 '((2 (5 (nil 3 1) 7))))
3
```

Решение 3.15.3

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun counter (x z w)
  (count-if #'(lambda (a) (<= x a z)) (flat w)))

> (counter 1 3 '((2 (5 (nil 3 1) 7))))
3
```

Задача 3.16 last-atom.lisp

Написать функцию, которая находит последний атом на верхнем уровне списка.

Решение 3.16.1

```
(defun last-atom (w)
 (find-if #'atom w :from-end t))

> (last-atom '((a b c) d e (f g h)))
E
```

Задача 3.17 number-max.lisp

Определить функцию, вычисляющую максимальный элемент в многоуровневом списке, состоящем из числовых атомов.

Решение 3.17.1

```
(defun maximus (w ac)
  (cond ((null w)  (car ac))
        ((listp (car w))
         (maximus (car w) (list (maximus (cdr w) ac))))
        ((null (car ac)) (maximus (cdr w) (list (car w))))
        ((> (car w) (car ac)) (maximus (cdr w) (list (car w))))
        ((maximus (cdr w) ac))))

(defun number-max (w)
  (maximus w nil))

> (number-max '(1 2 3 (4 5) (((6))) 4 5))
6
```

Задача 3.18 almost-last.lisp

Определить функцию, которая выдает в качестве результата предпоследний элемент списка, если он есть, а иначе nil.

Решение 3.18.1

```
(defun almost-last (w)
  (cond  ((null (cddr w)) (when (cdr w) (car w)))
         ((almost-last (cdr w)))))

> (almost-last '(1 2))
1
> (almost-last '(1))
NIL
```

Решение 3.18.2

```
(defun almost-last (w) (when (cdr w) (car (last w 2))))

> (almost-last '(1 2))
1
> (almost-last '(1))
NIL
```

Решение 3.18.3

```
(defun almost-last (w) (cadr (reverse w)))

> (almost-last '(1 2))
1
> (almost-last '(1))
NIL
```

Задача 3.19 palindrome.lisp

Есть список из атомов, имена которых - буквы. Написать функцию, которая выдает значение истина t, если слово из этих букв является палиндромом и ложь nil в противном случае.

Решение 3.19.1

```
(defun palindrome (w) (equalp w (reverse w)))

> (palindrome '(a b c b a))
T
> (palindrome "abba")
T
```

Задача 3.20 not-equal-last.lisp

Написать программу, которая подсчитывала бы количество элементов в списке отличных от последнего.

Решение 3.20.1

```
(defun not-equal-last (w)
  (cond ((null (cdr w)) 0)
        ((equalp (car w) (car (last w))) (not-equal-last (cdr w)))
        ((1+ (not-equal-last (cdr w))))))

> (not-equal-last '(a b b a))
2
```

Задача 3.21 sum-number.lisp

Дан список произвольной структуры, в который входят как числа, так и символьные атомы. Написать функцию, которая возвращает список из двух чисел, первое из них равно сумме чисел исходного списка, а второе их количеству.

Решение 3.21.1

```
(defun flat (w acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun sum-number (w &aux (v (remove-if-not #'numberp (flat w nil))))
  (values (reduce #'+ v) (length v)))

> (sum-number '(a (2) ((3) b 4) 5))
14
4
```

Задача 3.22 equal-set.lisp

Определите предикат равенство-множеств, проверяющий совпадение двух множеств (независимо от порядка следования элементов).

Решение 3.22.1 (автор – VH, www.cyberforum.ru)

```
(defun equal-set (w v)
  (cond ((null w) (null v))
        ((member (car w) v) (equal-set (cdr w) (remove (car w) v)))))

> (equal-set '(1 2) '(2 1))
T
> (equal-set '(1 2) '(2 2))
NIL
```

Задача 3.23 sum-plusp.lisp

Найти сумму неотрицательных элементов заданного списка целых чисел.

Решение 3.23.1

```
(defun sum-plusp (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 0)
        ((plusp a) (+ a (sum-plusp d)))
        ((sum-plusp d))))

> (sum-plusp '(2 -3 4 -1 -6))
6
```

Задача 3.24 tops-number.lisp

Определить функцию вычисления количества атомов на верхнем уровне многоуровневого списка.

Решение 3.24.1

```
(defun tops-number (w)
  (cond ((null w) 0)
        ((atom (car w)) (1+ (tops-number (cdr w))))
        ((tops-number (cdr w)))))

> (tops-number '(a (a) a (a a) a))
3
```

Решение 3.24.2

```
(defun tops-number (w)
  (if w (1+ (tops-number (cdr w))) 0))

> (tops-number '(a (a) a (a a) a))
5
```

Решение 3.24.3

```
(defun tops-number (w)
 (cond ((null w) 0)
       ((1+ (tops-number (cdr w))))))

> (tops-number '(a (a) a (a a) a))
5
```

Задача 3.25 sum-unique.lisp

Определить функцию, которая выводит сумму разных элементов списка.

Решение 3.25.1

```
(defun sum-unique (w)
  (cond ((null w) 0)
        ((member (car w) (cdr w)) (sum-unique (cdr w)))
        ((+ (car w)(sum-unique (cdr w))))))

> (sum-unique '(1 2 3 1 2 3))
6
```

Решение 3.25.2

```
(defun sum-unique (w)
  (reduce #'+ (remove-duplicates w)))

> (sum-unique '(1 2 3 1 2 3))
6
```

Задача 3.26 our-car-cdaddr.lisp

Составьте композицию из функций CAR и CDR, для которой результатом применения этой композиции к списку (1 (2) ((3 4) 5)) будет 5.

Решение 3.26.1

```
(defun our-car-cdaddr (w)
  (car (cdr (car (cdr (cdr w))))))

> (our-car-cdaddr '(1 (2) ((3 4) 5)))
5
```

Задача 3.27 descend.lisp

Проверять является ли отсортированным в порядке убывания список, состоящий из чисел.

Решение 3.27.1

```
(defun descend (w)
  (cond ((null (cdr w)) t)
        ((< (car w) (cadr w)) nil)
        ((descend (cdr w)))))

> (descend '(3 2 2 1))
T
```

Решение 3.27.2

```
(defun descend (w)
  (cond ((null (cdr w)))
        ((< (car w) (cadr w)) nil)
        ((descend (cdr w)))))
```

```
> (descend '(3 2 2 1))
T
```

Решение 3.27.3

```
(defun descend (w)
  (if (null (cdr w)) t (and (>= (car w) (cadr w)) (descend (cdr w)))))

> (descend '(3 2 2 1))
T
> (descend '(3 2 2 10))
NIL
```

Решение 3.27.4

```
(defun descend (w)
  (if (cdr w) (and (>= (car w) (cadr w)) (descend (cdr w))) t))

> (descend '(3 2 2 1))
T
> (descend '(3 2 2 10))
NIL
```

Решение 3.27.5

```
(defun descend (w)
  (apply #'> w))

> (descend '(3 2 2 1))
T
```

Задача 3.28 our-caddr.lisp

Определите функцию (caddr x) с помощью базовых функций. Используйте имя our-caddr, чтобы не переопределять одноименную встроенную функцию Лиспа.

Решение 3.28.1

```
(defun our-caddr (w)
  (car (cdr (cdr w))))

>  (our-caddr '(1 2 3))
3
```

Задача 3.29 count-low-atoms.lisp

Задан многоуровневый список, найти количество атомов списка, находящихся на нижних уровнях.

Решение 3.29.1

```
(defun del-upper-atoms (w)
  (cond ((null w) nil)
        ((atom (car w)) (del-upper-atoms (cdr w)))
        ((cons (car w) (del-upper-atoms (cdr w))))))

(defun flatten (w &optional ac)
  (cond ((null w) ac)
        ((atom w) (cons w ac))
        ((flatten (car w) (flatten (cdr w) ac)))))

(defun count-atoms (w &optional (n 0))
    (cond ((null w) n)
          ((count-atoms (cdr w) (1+ n)))))

(defun count-low-atoms (w)
  (count-atoms (flatten (del-upper-atoms w))))

> (count-low-atoms '((a)))
1
```

Решение 3.29.2

```
(defun flatten (w &optional ac)
  (cond ((null w) ac)
        ((atom w) (cons w ac))
        ((flat (car w) (flatten (cdr w) ac)))))

(defun count-low-atoms (w)
  (count-if #'atom (flatten (delete-if #'atom w))))

> (count-low-atoms '((a)))
1
```

Задача 3.30 even.lisp

Определять, является ли сумма элементов списка из целых чисел, четным числом.

Решение 3.30.1

```
(defun even (w &optional (ac 0))
  (cond ((null w) (evenp ac))
        ((even (cdr w) (+ ac (car w))))))

> (even '(1 1))
T
```

Задача 3.31 sum-odd.lisp

Подсчитать сумму всех нечётных элементов списка (по месту нахождения).

Решение 3.31.1

```
(defun sum-odd (w &optional (ac 0))
  (cond ((null w) ac)
        ((oddp (car w)) (sum-odd (cdr w) (+ ac (car w))))
        ((sum-odd (cdr w) ac))))

> (sum-odd '(1 2 3 4))
4
```

Задача 3.32 sum-even.lisp

Подсчитать сумму всех чётных элементов списка (по месту нахождения).

Решение 3.32.1

```
(defun sum-even (w &optional (ac 0))
  (cond ((null w) ac)
        ((evenp (car w)) (sum-even (cdr w) (+ ac (car w))))
        ((sum-even (cdr w) ac))))

> (sum-even '(1 2 3 4))
6
```

Задача 3.33 count-zero.lisp

Заданы четыре числа a, b, c, d. Определить количество нулевых значений.

Решение 3.33.1

```
(defun count-zero (a b c d)
  (count-if #'zerop (list a b c d)))

> (count-zero 1 0 1 0)
2
```

Решение 3.33.2

```
(defun count-zero (a b c d)
  (+ (if (zerop a) 1 0)
     (if (zerop b) 1 0)
     (if (zerop c) 1 0)
     (if (zerop d) 1 0)))

> (count-zero 1 0 1 0)
2
```

Решение 3.33.3

```
(defun count-zero (w &optional (n 0))
  (cond ((null w) n)
        ((zerop (car w)) (count-zero (cdr w) (1+ n)))
        ((count-zero (cdr w) n))))

> (count-zero '(1 0 1 0))
2
```

Решение 3.33.4

```
(defun count-x (w &optional (n 0))
  (cond ((null w) n)
        ((zerop (car w)) (count-x (cdr w) (1+ n)))
        ((count-x (cdr w) n))))

(defun count-zero (a b c d)
  (count-x (list a b c d)))

> (count-zero 1 0 1 0)
2
```

Решение 3.33.5

```
(defun count-x (w &optional (n 0))
  (cond ((null w) n)
        ((zerop (car w)) (count-x (cdr w) (1+ n)))
        ((count-x (cdr w) n))))

(defun count-zero (&rest w)
  (count-x w))

> (count-zero 1 0 1 0)
2
```

Решение 3.33.6

```
(defun count-zero (a b c d &optional (w (list a b c d)) (n 0))
  (cond ((null w) n)
        ((zerop (car w)) (count-zero a b c d (cdr w) (1+ n)))
        ((count-zero a b c d (cdr w) n))))

> (count-zero 0 2 3 0)
2
```

Решение 3.33.7

```
(defun count-zero (a b c d)
  (reduce #'+ (mapcar #'(lambda (e) (if (zerop e) 1 0))
                      (list a b c d))))

> (count-zero 0 1 0 2)
2
```

Решение 3.33.8

```
(defun count-zero (a b c d)
  (labels ((counter (v n)
             (cond ((null v) n)
                   ((zerop (car v)) (counter (cdr v) (1+ n)))
                   ((counter (cdr v) n)))))
    (counter (list a b c d) 0)))

> (count-zero 0 1 0 2)
2
```

Решение 3.33.9

```
(defun count-zero (&rest w)
  (labels ((counter (v n)
             (cond ((null v) n)
                   ((zerop (car v)) (counter (cdr v) (1+ n)))
                   ((counter (cdr v) n)))))
    (counter w 0)))

> (count-zero 0 1 0 2)
2
```

Задача 3.34 product-even.lisp

Дан список целых чисел. Составить программу, вычисляющую произведение всех четных элементов списка за исключение нуля.

Решение 3.34.1

```
(defun product-even (w)
  (reduce #'* (remove-if #'zerop (remove-if-not #'evenp w))))

> (product-even '(0 1 2 3 4))
8
```

Задача 3.35 average-cadr.lisp

Напишите функцию для вычисления среднего возраста группы людей, которая описывается ассоциативным списком, например таким — ((mark 40) (kathy 30) (sally 57) (doran 35) (anthony 3)), здесь возраст является вторым элементом подсписка.

Решение 3.35.1

```
defun sm (w m n)
  (cond ((null w) (/ m n))
        ((sm (cdr w)(+ m (cadar w)) (1+ n)))))
```

```
(defun average-cadr (w)
  (sm w 0 0))

> (average-cadr '((mark 40) (kathy 30) (sally 57) (doran 35) (anthony
3)))
33
```

Задача *3.36 binary-arithmetic.lisp*

Написать функцию сложения, вычитания, умножения и деления двоичных чисел.

Решение 3.36.1

```
(defun binary-arithmetic (w)
  (format
   nil "~b"
   (reduce
    (car w)
    (mapcar
     #'(lambda (a)
         (parse-integer (write-to-string a) :radix 2))
     (cdr w)))))

> (binary-arithmetic  '(+ 1011 111001 101))
"1001001"
```

Решение 3.36.2

```
(defun deci-bin (n &optional ac)
  (if (zerop n) ac (deci-bin (floor (/ n 2)) (cons (mod n 2) ac))))

(defun bin-deci (w &optional (n 0))
  (if (null w) n (bin-deci (cdr w) (+ (car w) n n))))

(defun binary+ (&rest w)
  (deci-bin (reduce #'+ (mapcar #'bin-deci w))))

> (binary+  '(1 0 1 1) '(1 1 1 0 0 1) '(1 0 1))
(1 0 0 1 0 0 1)
```

Задача 3.37 min-number.lisp

Задан список произвольных чисел. Написать функцию нахождения минимального элемента.

Решение 3.37.1

```
(defun min-number (w &optional x &aux (a (car w)))
  (cond ((null w) x)
        ((or (not x) (> x a)) (min-number (cdr w) a))
```

```
        ((min-number (cdr w) x))))

> (min-number '(1 2))
1
```

Задача 3.38 count-atoms.lisp

Напишите функцию, считающую полное количество атомов (не равных nil) в многоуровневом списке.

Решение 3.38.1

```
(defun count-atoms (w &optional (n 0))
  (cond ((null w) n)
        ((atom (car w)) (count-atoms (cdr w) (1+ n)))
        ((count-atoms (car w) (count-atoms (cdr w) n)))))

> (count-atoms '(a (b c (d) e)))
5
```

Решение 3.38.2

```
(defun count-atoms (w)
  (cond ((null w) 0)
        ((atom w) 1)
        ((+ (count-atoms (car w)) (count-atoms (cdr w))))))

> (count-atoms '(a (b c (d) e)))
5
```

Решение 3.38.3

```
(defun count-atoms (w)
  (loop for a in w sum (if (atom a) 1 (count-atoms a))))

> (count-atoms '(a (b c (d) e)))
5
```

Задача 3.39 sum-137-last.lisp

Дан список. Найти сумму 1,3 и 7 элемента, если это цифры, иначе возвратить последний элемент списка.

Решение 3.39.1

```
(defun sum-137-last
    (w &aux (a (first w)) (b (third w)) (c (seventh w)))
  (if (every #'numberp (list a b c)) (+ a b c) (car (last w))))

> (sum-137-last '(1 2 3 4 5 6 7))
11
```

Задача 3.40 double-plusp.lisp

Дан список целых чисел. Получить удвоенную сумму всех положительных элементов списка.

Решение 3.40.1

```
(defun double-plusp (w)
  (* (apply #'+ (delete-if-not #'plusp w)) 2))

> (double-plusp '(1 -2 3))
8
```

Задача 3.41 avarage-num.liap

Найти среднее арифметическое списка вещественных чисел.

Решение 3.41.1

```
(defun avarage-num (w)
  (when w (/ (reduce #'+ w) (count-if #'realp w))))

> (avarage-num '(11 12 13))
12
```

Решение 3.41.2

```
(defun avarage-num (w)
  (when w (/ (reduce #'+ w) (length w))))

> (avarage-num '(11 12 13))
12
```

Задача 3.42 more-nums.lisp

Написать функцию, которая принимает значение t, если в списке больше чисел, чем атомов.

Решение 3.42.1

```
(defun more-nums (w &optional (n 0))
  (cond ((null w) (plusp n))
        ((numberp (car w)) (more-nums (cdr w) (1+ n)))
        ((more-nums (cdr w) (1- n)))))

> (more-nums '(1 1 a))
T
```

Задача 3.43 list2-3.lisp

Напишите функцию, которая выдает истину, если ее аргумент удовлетворяет хотя бы одному из следующих условий:
а) является списком из двух элементов;
б) является списком из двух атомов;
в) является списком из трех элементов.

Решение 3.43.1

```
(defun list2-3 (x)
  (and (consp x) (cdr x) (null (cdddr x))))

> (list2-3 '(a))
NIL
> (list2-3 '(a b))
T
> (list2-3 '(a b c))
T
> (list2-3 '(a b c d))
NIL
```

Задача 3.44 matrix-main.lisp

Определить функцию, вычисляющую сумму элементов главной диагонали матрицы.

Решение 3.44.1

```
(defun matrix-main (w &optional (n 0))
  (if (null w) 0 (+ (nth n (car w)) (matrix-main (cdr w) (1+ n)))))

> (setq a '((1 0 0) (0 1 0) (0 0 1)))
((1 0 0) (0 1 0) (0 0 1))
> (matrix-main a)
3
> (setq a '((1 0 0 0) (0 1 0 0) (0 0 1 0) (0 0 0 1)))
((1 0 0 0) (0 1 0 0) (0 0 1 0) (0 0 0 1))
> (matrix-main a)
4
```

Задача 3.45 max-depth.lisp

Определить функцию, вычисляющую максимальный уровень вложенности списка.

Решение 3.45.1

```
(defun max-depth (w)
  (cond ((null w) 0)
        ((atom (car w)) (max-depth (cdr w)))
        ((max (1+ (max-depth (car w))) (max-depth (cdr w))))))
```

```
> (max-depth '(1 2 3))
0
> (max-depth '(1 ((2)) 3))
2
```

Задача 3.46 evenp-sum.lisp

Написать программу, определяющую, является ли сумма элементов многоуровневого списка целых чисел четным числом.

Решение 3.46.1

```
(defun sum (w)
  (cond ((null w) 0)
        ((atom (car w)) (+ (car w) (sum (cdr w))))
        ((+ (sum (car w)) (sum (cdr w))))))

(defun evenp-sum (w) (evenp (sum w)))

> (evenp-sum '(1 (2 (3)) (4)))
T
> (evenp-sum '(0 (2 (3)) (4)))
NIL
```

Решение 3.46.2

```
(defun sum (w)
  (cond ((null w) 0)
        ((atom w) w)
        ((+ (sum (car w)) (sum (cdr w))))))

(defun evenp-sum (w) (evenp (sum w)))

> (evenp-sum '(1 (2 (3)) (4)))
T
> (evenp-sum '(0 (2 (3)) (4)))
NIL
```

Решение 3.46.3

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun evenp-sum (w)
  (evenp (loop for a in (flat w) summing a)))

> (evenp-sum '(1 (2 (3)) (4)))
T
> (evenp-sum '(0 (2 (3)) (4)))
NIL
```

```
Решение 3.46.4

(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun evenp-sum (w)
  (evenp (reduce #'+ (flat w))))

> (evenp-sum '(1 (2 (3)) (4)))
T
> (evenp-sum '(0 (2 (3)) (4)))
NIL
```

Задача 3.47 |parens left|.lisp

```
Определить функцию, проверяющую, что все левые скобки в s-выражении,
представляющем список, являются крайними левыми скобками.

Решение 3.47.1

(defun |parens left| (w)
  (if w
      (and (or (atom (car w)) (|parens left| (car w)))
               (every #'atom (cdr w)))
      t))

> (|parens left| '((((3 4) 5) 6 7) 1))
T
> (|parens left| '((((3 4) 5) (6 7) 1)))
NIL

Решение 3.47.2

(defun |parens left| (w)
  (cond ((null w) t)
        ((every #'atom (cdr w))
         (or (atom (car w)) (|parens left| (car w))))
        (t nil)))

> (|parens left| '((((3 4) 5) 6 7) 1))
T
> (|parens left| '((((3 4) 5) (6 7) 1)))
NIL

Решение 3.47.3

(defun atoms (w)
  (every #'atom w))
```

```
(defun |parens left| (w)
  (cond ((null w) t)
        ((atoms (cdr w)) (or (atom (car w)) (|parens left| (car w))))
        (t nil)))

> (|parens left| '((((3 4) 5) 6 7) 1))
T
> (|parens left| '((((3 4) 5) (6 7) 1)))
NIL
```

Решение 3.47.4

```
(defun atoms (w)
  (cond ((null w) t)
        ((atom (car w)) (atoms (cdr w)))
        (t niL)))

(defun |parens left| (w)
  (cond ((null w) t)
        ((atoms (cdr w)) (or (atom (car w)) (|parens left| (car w))))
        (t nil)))

> (|parens left| '((((3 4) 5) 6 7) 1))
T
> (|parens left| '((((3 4) 5) (6 7) 1)))
NIL
```

Решение 3.47.5

```
(defun |parens left| (w)
  (cond ((cdr w) t)
        ((and (not (atom (car w))) (cdr w)) (|parens left| (car w)))
        (t nil)))

> (|parens left| '((((3 4) 5) 6 7) 1))
T
> (|parens left| '((((3 4) 5) (6 7) 1)))
NIL
```

Решение 3.47.6

```
(defun atoms (w)
  (cond ((null w) t)
        ((atom (car w)) (atoms (cdr w)))
        (t niL)))

(defun |parens left| (w)
  (cond ((cdr w) t)
        ((or (atom (car w)) (null (cdr w))) nil)
        ((|parens left| (car w)))))

> (|parens left| '((((3 4) 5) 6 7) 1))
T
```

```
> (|parens left| '((((3 4) 5) (6 7) 1)))
NIL
```

Задача 3.48 max-occur.lisp

Определить функцию, которая находит элемент с максимальным числом вхождений в список.

Решение 3.48.1

```
(defun max-occur (w &optional (n 0) z &aux (m (count (car w) w)))
  (cond ((null w) z)
        ((> m n) (max-occur (cdr w) m (car w)))
        ((max-occur  (cdr w) n z))))

> (max-occur '(1 1 1 1 2 3 1))
1
```

Решение 3.48.2

```
(defun our-count (a w &optional (p 0))
  (cond ((null w) p)
        ((equal a (car w)) (our-count a (cdr w) (1+ p)))
        ((our-count a (cdr w) p))))

(defun max-occur (w &optional (n 0) z &aux (m (our-count (car w) w)))
  (cond ((null w) z)
        ((> m n) (max-occur (cdr w) m (car w)))
        ((max-occur (cdr w) n z))))

> (max-occur '(1 1 1 1 2 3 1))
1
```

Решение 3.48.3

```
(defun our-count (a w &optional (p 0))
  (cond ((null w) p)
        ((equal a (car w)) (our-count a (cdr w) (1+ p)))
        ((our-count a (cdr w) p))))

(defun max-occur (w &optional (n 0) z)
      (cond ((null w) z)
            ((> (our-count (car w) w) n)
             (max-occur (cdr w) (f (car w) w) (car w)))
            ((max-occur (cdr w) n z))))

> (max-occur '(1 1 1 1 2 3 1))
1
```

Решение 3.48.4 * без использования счетчика

```
(defun drop-one (w &optional acc ac)
```

```
  (cond ((null w) acc)
        ((member (car w) ac) (drop-one (cdr w) (cons (car w) acc) ac))
        ((drop-one (cdr w) acc (cons (car w) ac)))))

(defun max-occur (w)
  (if (cdr w) (max-occur (drop-one w)) (car w)))

> (max-occur '(1 1 1 1 2 3 3 1))
1
```

Задача 3.49 caar-cdaaar.lisp

Используя только car и cdr вытащить aaa и следующего списка: ((((ddd (aaa)) eee) nnn) kkk).

Решение 3.49.1

```
> (car (car (cdr (car (car (car '((((ddd (aaa)) eee) nnn) kkk)))))))
AAA
```

Решение 3.49.2

```
> (caar (cdaaar '((((ddd (aaa)) eee) nnn) kkk)))
AAA
```

Задача 3.50 caddd.lisp

Используя только car и cdr вытащить aaa и следующего списка: ((((ddd (aaa)) eee) nnn) kkk).

Решение 3.50.1

```
> (car (cdr (cdr (cdr '(hhh iii ddd aaa)))))
AAA
```

Решение 3.50.2

```
> (cadddr '(hhh iii ddd aaa))
AAA
```

Задача 3.51 count-till*.lisp

Определить функцию, которая вычисляет количество элементов списка, предшествующих звездочке (*).

Решение 3.51.1

```
(defun count-till* (w)
   (cond ((null w) 0)
         ((equal (car w) '*) 0)
         ((+ 1 (count-till* (cdr w))))))
```

```
> (count-till* '(1 2 3 4 * 6 7))
4
```

Задача 3.52 winner.lisp

Известны результаты бега спортсменов на длинную дистанцию. Время представлено в минутах и секундах. Определить фамилию победителя соревнования.

Решение 3.52.1

```
(defstruct run name mm ss)

(defun res (n)
  (when (> n 0) (cons (make-run :name (read) :mm (read) :ss (read))
                      (res (1- n)))))

(defun name-ss (w)
  (mapcar #'(lambda (a) (list (run-name a)
                              (+ (* (run-mm a) 60) (run-ss a))))
          w))

(defun r-sort (w)
  (sort (name-ss w) #'< :key #'cadr))

(defun winner (w)
  (caar (r-sort w)))

> (setq 3km (res 2))
ivanov
10
54
petrov
9
37
(#S(RUN :NAME IVANOV :MM 10 :SS 54) #S(RUN :NAME PETROV :MM 9 :SS 37))
> (winner 3km)
PETROV
```

Задача 3.53 advantage.lisp

Известны результаты бега спортсменов на длинную дистанцию. Время представлено в минутах и секундах. Определить разницу между лучшим и худшим результатом.

Решение 3.53.1

```
(defstruct run name mm ss)

(defun res (n)
```

```
  (when (> n 0) (cons (make-run :name (read) :mm (read) :ss (read))
                      (res (1- n)))))

(defun name-ss (w)
  (mapcar #'(lambda (a) (list (run-name a)
                              (+ (* (run-mm a) 60) (run-ss a))))
          w))

(defun r-sort (w)
  (sort (name-ss w) #'< :key #'cadr))

(defun advantage (w &aux (v (r-sort w)))
  (- (cadar (last v)) (cadar v)))

> (setq 3km (res 2))
ivanov
10
54
petrov
9
37
(#S(RUN :NAME IVANOV :MM 10 :SS 54) #S(RUN :NAME PETROV :MM 9 :SS 37))
> (advantage 3km)
77
```

Задача 3.54 clear-average.lisp

Используя рекурсивный подход исключить из списка все вхождения элемента nil и вычислить среднее геометрическое элементов списка.

Решение 3.54.1

```
(defun clear-average (w &optional ac (n 0))
  (conD ((null w) (/ (apply #'* ac) n))
        ((null (car w)) (clear-average (cdr w) ac n))
        ((clear-average (cdr w) (cons (car w) ac) (1+ n)))))

> (setf w '(nil 46 45 44 41  nil 44 45 46 43 44 47  nil 46 45 46 46 44
42 41 49 nil))
(NIL 46 45 44 41 NIL 44 45 46 43 44 47 NIL 46 45 46 46 44 42 41 49
NIL)
> (clear-average w)
27324001339715137104165888000
> (setf w '(nil 46 45 44 41  nil 44 45 46 43 44 47  nil 46 45 46 nil))
(NIL 46 45 44 41 NIL 44 45 46 43 44 47 NIL 46 45 46 NIL)
> (clear-average w)
2879897744844185472000/13
```

Задача 3.55 lead-atom.lisp

Определить Функцию, которая вернет первый атом в списке, а если такого

в списке нет, вернет nil.

Решение 3.55.1

```
(defun lead-atom (w)
  (cond ((null w) nil)
        ((atom (car w)) (car w))
        ((lead-atom (cdr w)))))

> (lead-atom '((a) ((b) c) d))
D
> (lead-atom '((a) ((b) c)))
NIL
```

Решение 3.55.2

```
(defun lead-atom (w)
  (loop for a in w when (atom a) return a))

> (lead-atom '((a) ((b) c) d))
D
> (lead-atom '((a) ((b) c)))
NIL
```

Решение 3.55.3

```
(defun lead-atom (w) (find-if #'atom w))

> (lead-atom '((a) ((b) c)))
NIL
> (lead-atom '((a) ((b) c) d))
D
> (lead-atom '((a) ((b) c) d e))
D
```

Задача 3.56 product-even.lisp

Дан список целых чисел. Составить программу, вычисляющую произведение всех четных элементов списка за исключение нуля.

Решение 3.56.1

```
(defun product-even (w &aux (a (car w)))
  (cond ((null w) 1)
        ((or (oddp a) (zerop a)) (product-even (cdr w)))
        ((* a (product-even (cdr w))))))

> (product-even '(1 2 3 4 5 6 7))
48
```

Задача 3.57 our-length.lisp

Определить рекурсивную функцию, возвращающую количество элементов в списке.

Решение 3.57.1

```
(defun our-length (w)
  (if (null w) 0 (1+ (our-length (cdr w)))))

> (our-length '(a b c d))
4
> (our-length '())
0
```

Задача 3.58 sumx.lisp

Найти сумму чисел одноуровневого списка.

Решение 3.58.1

```
(defun sumx (w)
  (cond ((null w) 0)
        ((numberp (car w)) (+ (car w) (sumx (cdr w))))
        ((sumx (cdr w)))))

> (sumx '(54 gh 5 fd 7 9 12 -2 f))
85
```

Решение 3.58.2

```
(defun sumx (w)
  (loop for a in w when (numberp a) sum a))

> (sumx '(54 gh 5 fd 7 9 12 -2 f))
85
```

Задача 3.59 mult.lisp

Найти произведение чисел одноуровневого списка.

Решение 3.59.1

```
(defun mult (w)
  (cond ((null w) 1)
        ((numberp (car w)) (* (car w) (mult (cdr w))))
        ((mult (cdr w)))))

> (mult '(54 gh 5 fd 7 9 12 -2 f))
-408240
```

Решение 3.59.2

```
(defun mult (w)
  (loop for a in w
        when (numberp a) collect a into v
        finally return (reduce #'* v)))

> (mult '(2 gh 2 fd 2 2 2 -2 f))
-64
```

Задача 3.60 longest.lisp

В списке однобуквенных строк S1, S2, ..., ST найти длину самого длинного слова, если разделителем между словами является один или несколько пробелов.

Решение 3.60.1

```
(defun longest (w)
  (apply #'max
         (mapcar #'length
                 (mapcar #'string
                         (read-from-string
                          (concatenate 'string
                                       "("
                                       (apply #'concatenate 'string w)
                                       ")"))))))

> (longest '("t" "e" "s" "t" " " "s" "t" "r" "o" "k" "a" " " "s"))
6
```

Задача 3.61 prod-digits.lisp

Найти произведение цифр в символах одноуровневого списка.

Решение 3.61.1

```
(defun prod-digits (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 1)
        ((symbolp a) (* (apply
                         #'* (mapcar
                              #'digit-char-p
                              (remove-if-not
                               #'digit-char-p
                               (coerce (string a) 'list))))
                        (prod-digits d)))
        ((* (abs a) (prod-digits d)))))

> (prod-digits '(a s ss d 1 -2 h3-4d))
24
```

Решение 3.61.1

```
(defun prod-digits (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 1)
        ((numberp a) (* (abs a) (prod-digits d)))
        ((* (apply #'* (mapcar #'digit-char-p
                               (remove-if-not
                                #'digit-char-p
                                (coerce (string a) 'list))))
            (prod-digits d)))))
```

```
> (prod-digits '(a s ss d 1 -2 h3-4d))
24
```

Задача 3.62 sum-digits.lisp

Найти сумму цифр в символах одноуровневого списка.

Решение 3.62.1

```
(defun sum-digits (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 0)
        ((numberp a) (+ (abs a) (sum-digits d)))
        ((+ (apply #'+ (mapcar #'digit-char-p
                               (remove-if-not
                                #'digit-char-p
                                (coerce (string a) 'list))))
            (sum-digits d)))))
```

```
> (sum-digits '(1a3 e34d w12 r3r1))
18
```

Задача 3.63 magic-square.lisp

Написать программу, которая проверяет, является ли квадратная матрица "магическим" квадратом. "Магическим" квадратом называется матрица, у которой сумма чисел в каждом горизонтальном ряду, в каждом вертикальном и по каждой из диагоналей одна и та же. Пример матрицы 3x3:

```
2 9 4
7 5 3
6 1 8
```

Решение 3.63.1

```
(defun transpose (m) (apply #'mapcar #'list m))

(defun sum-main (w &optional (n 0))
  (if (null w) 0 (+ (nth n (car w)) (sum-main (cdr w) (1+ n)))))

(defun magic-sq (m)
```

```
  (apply #'=
         (sum-main m)
         (sum-main (reverse m))
         (mapcar #'(lambda (a) (reduce #'+ a))
                 (nconc m (transpose m)))))

> (magic-sq '((2 9 4) (7 5 3) (6 1 8)))
T
> (magic-sq '((2 9 4) (7 5 3) (1 6 8)))
NIL
```

Задача 3.64 count-not-numbers.lisp

Посчитать количество нечисловых элементов в списке.

Решение 3.64.1

```
(defun count-not-numbers (w)
  (cond ((null w) 0)
        ((numberp (car w)) (count-not-numbers (cdr w)))
        ((1+ (count-not-numbers (cdr w))))))

> (count-not-numbers '(a a 1))
2
```

Решение 3.64.2

```
(defun count-not-numbers (w)
  (loop for a in w counting (not (numberp a))))

> (count-not-numbers '(a a 1))
2
```

Решение 3.64.3

```
(defun count-not-numbers (w)
  (count-if-not #'numberp w))

> (count-not-numbers '(a a 1))
2
```

Задача 3.65 double-zero.lisp

Определить наличие в числовом списке двух подряд идущих нулевых элементов.

Решение 3.65.1

```
(defun double-zero (w)
  (cond ((null (cdr w)) nil)
        ((= (car w) (cadr w) 0) t)
```

```
        ((double-zero (cdr w)))))

> (double-zero '(0 0))
T
> (double-zero '(0 1))
NIL
```

Решение 3.65.2

```
(defun double-zero (w)
  (mapcan #'(lambda (a b)
              (when (= a b 0) (list 'ok)))
          w (cdr w)))

> (double-zero '(0 1))
NIL
> (double-zero '(0 0))
(OK)
```

Решение 3.65.3

```
(defun double-zero (w)
  (if (loop for a on w never
            (when (cdr a) (= (car a) (cadr w) 0))) nil t))

> (double-zero '(0 0))
T
> (double-zero '(0 1))
NIL
```

Решение 3.65.4

```
(defun double-zero (w)
  (loop for a on w thereis
        (when (cdr a) (= (car a) (cadr w) 0))))

> (double-zero '(0 1))
NIL
> (double-zero '(0 0))
T
```

Решение 3.65.5

```
(defun double-zero (w)
  (loop for a in w for b in (cdr w) thereis (= a b 0)))

> (double-zero '(0 0))
T
> (double-zero '(0 1))
NIL
```

Задача 3.66 max-min.lisp

Дан список. Если максимальный элемент предшествует минимальному, то посчитать их сумму, если нет то просто вывести 0.

Решение 3.66.1

```
(defun max-min (w &aux (x (apply #'max w)) (n (apply #'min w)))
  (if (< (position x w) (position n w)) (+ x n) 0))

> (max-min '(9 7 6 5 4 3 2 1))
10
> (max-min '(7 6 5 4 3 2 1 9))
0
```

Задача 3.67 expt-7-8.lisp

Дан числовой список. Возвести седьмой элемента списка в степень, равную восьмому элементу.

Решение 3.67.1

```
(defun expt-7-8 (w)
  (labels ((_expt (n m)
             (if (= m 1) n (* n (_expt n (1- m))))))
    (_expt (nth 6 w) (nth 7 w))))

> (expt-7-8 '(1 2 3 4 5 6 7 8 9))
5764801
> (expt 7 8)
5764801
```

Задача 3.68 max-deptht.lisp

Написать функцию, определяющую максимальную глубину вложенности подсписков многоуровневого списка.

Решение 3.68.1

```
(defun max-depth (w)
  (cond ((atom w) 0)
        ((max (+ 1 (max-depth (car w))) (max-depth (cdr w))))))

> (max-depth '(a b (c d (e f) g)))
3
```

Решение 3.68.2

```
(defun max-depth (w)
  (if (atom w) 0 (max (1+ (max-depth (car w))) (max-depth (cdr w)))))
```

```
> (max-depth '(a b (c d (e f) g)))
3
```

Задача 3.69 average-wage.lisp

Дан список, содержащий сведения о зарплате рабочих завода. Каждая запись содержит поля - фамилия работника, наименование цеха, размер зарплаты за месяц. Вычислить среднемесячный заработок рабочего этого завода.

Решение 3.69.1

```
(defun average-wage (w)
  (float (/ (sum-wage w) (num-workers w))))

(defun sum-wage (w)
  (if (null w) 0 (+ (caddar w) (sum-wage (cdr w)))))

(defun num-workers (w)
  (if (null w) 0 (1+ (num-workers (cdr w)))))

> (average-wage '((ivanov 2 15000) (petrov 2 12000) (sidorovv 2
16000)))
14333.333
```

Решение 3.69.2

```
(defun average-wage (w)
  (float (/ (loop for a in w sum (caddr a)) (length w))))

> (average-wage '((ivanov 2 15000) (petrov 2 12000) (sidorovv 2
16000)))
14333.333
```

Задача 3.70 min-plusp.lisp

Написать программу поиска минимального положительного элемента.

Решение 3.70.1

```
(defun min-plusp (w)
  (cond ((null w) nil)
        ((plusp (car w)) (mp (cdr w) (car w)))
        ((min-plusp (cdr w)))))

(defun mp (w n)
  (cond ((null w) n)
        ((and (plusp (car w)) (< (car w) n)) (mp (cdr w) (car w)))
        ((mp (cdr w) n))))

> (min-plusp '(-1 0 1 2))
```

```
1
```

```
Решение 3.70.2
```

```
(defun min-plusp (w)
  (reduce #'min (remove-if-not #'plusp w)))

> (min-plusp '(-1 0 1 2))
1
```

Задача 3.71 max-line.lisp

```
Определить номер строки матрицы, сумма элементов которой наибольшая.
```

```
Решение 3.71.1
```

```
(defun max-line (w &aux (v (mapcar #'(lambda (a) (apply #'+ a)) w)))
  (position (reduce #'max v) v))

> (max-line '((1 2 3) (1 2 4) (1 2 3)))
1
```

Задача 3.72 sum-eventh.lisp

```
Дан числовой список. Определить функцию, вычисляющую сумму чисел,
стоящих на четных местах.
```

```
Решение 3.72.1
```

```
(defun sum-eventh (w)
  (if (cdr w) (+ (cadr w) (sum-eventh (cddr w))) 0))

> (sum-eventh '(0 1 0 1 1))
2
```

Задача 3.73 compare-first-last.lisp

```
В списке, состоящем из подсписков типа ((a b) (s h) (h a)) проверить,
совпадают ли первый и последний символы.
```

```
Решение 3.73.1
```

```
(defun deep-last (w &aux (a (car w)))
  (cond ((null (cdr w)) (if (atom a) a (deep-last a)))
        ((deep-last (cdr w)))))

(defun compare-first-last (w)
  (eq (caar w) (deep-last w)))

> (compare-first-last '((1 2 3) (4 5 6) (7 8 9)))
NIL
```

```
> (compare-first-last '((1 2 3) (4 5 6) (7 8 1)))
T
```

Решение 3.73.2

```
(defun deep-last (w &aux (a (car w)))
  (if (cdr w) (deep-last (cdr w)) (if (atom a) a (deep-last a))))

(defun compare-first-last (w)
  (eq (caar w) (deep-last w)))

> (compare-first-last '((1 2 3) (4 5 6) (7 8 1)))
T
> (compare-first-last '((1 2 3) (4 5 6) (7 8 9)))
NIL
```

Задача 3.74 non-number.lisp

Написать функцию, возвращающую первый нечисловой элемент заданного списка.

Решение 3.74.1

```
(defun non-number (w)
  (loop for a in w unless (numberp a) return a))

> (non-number '(1 2 a b 3))
A
```

Задача 3.75 expp.lisp

Написать функцию, проверяющую корректность произвольного выражения в форме списка - например, a*b2*5+a*x.

Решение 3.75.1

```
(defun divide (w &aux (n (truncate (/ (length w) 2))))
  (mapcar #'list (subseq w 0 (1+ n)) (reverse (subseq w (1- n)))))

(defun collect (w)
  (when w (append (car w) (collect (cddr w)))))

(defun check-s (w &aux (a (car w)))
  (cond ((null w) t)
        ((or (eq a '+) (eq a '*)) nil)
        ((check-s (cdr w)))))

(defun check-m (w &aux (a (car w)))
  (cond ((null w) t)
        ((or (eq a '+) (eq a '*)) (check-m (cdr w)))
        (t nil)))
```

```
(defun expp (w &aux (v (divide w)))
  (and (check-s (collect v)) (check-m (collect (cdr v)))))

> (expp '(a * b2 * 5 + a * x))
T
> (expp '(a * b2 * 5 + a *))
NIL
```

Решение 3.75.2

```
(defun expp (w &optional f &aux (a (car w)))
  (cond ((null w) f)
        ((and f (or (eq a '+) (eq a '*))) (expp (cdr w) nil))
        ((expp (cdr w) t))))

> (expp '(a * b2 * 5 + a * x))
T
> (expp '(a * b2 * 5 + a *))
NIL
```

Решение 3.75.3

```
(defun expp (w &optional f)
  (cond ((null w) f)
        ((and f (or (eq (car w) '+) (eq (car w) '*)))
         (expp (cdr w) nil))
        ((expp (cdr w) t))))

> (expp '(a * b2 * 5 + a * x))
T
> (expp '(a * b2 * 5 + a *))
NIL
```

Решение 3.75.4

```
(defun expp (w &optional f &aux (a (car w)))
  (if w (expp (cdr w) (not (and f (or (eq a '+) (eq a '*))))) f))

> (expp '(a * b2 * 5 + a * x))
T
> (expp '(a * b2 * 5 + a *))
NIL
```

Задача 3.76 min-number.lisp

Дан список, элементами которого являются списки, состоящие из чисел. Определить номер элемента списка, содержащего наименьшее число. Например, в случае `'((1 3) (0) (6 1)), ответ: n = 2. Не рассматривать случай, когда такой элемент не единственный.

Решение 3.76.1

```
defun min-number (w &aux (v (mapcar
                              #'(lambda (a)
                                  (reduce #'min a)) w)))
  (position (reduce #'min v) v))

> (min-number '((1 1) (1 0) (1 1)))
1
```

Задача 3.77 sum-elms.lisp

Определить функцию, которая считает сумму элементов многоуровневого списка.

Решение 3.77.1

```
(defun sum-elms (w)
  (cond ((null w) 0)
        ((atom w) w)
        ((+ (sum-elms (car w)) (sum-elms (cdr w))))))

> (sum-elms '(1 ((1) (1)) (1) 1))
5
```

Задача 3.78 deep-product.lisp

Определить функцию, которая считает произведение элементов многоуровневого списка.

Решение 3.78.1

```
(defun deep-product (w)
  (cond ((null w) 1)
        ((atom (car w)) (* (car w) (deep-product (cdr w))))
        (t (* (deep-product (car w)) (deep-product (cdr w))))))

> (deep-product '(1 2 3 (3 6) 2 1 (2 4) 2))
3456
```

Решение 3.78.2 (автор – helter, www.cyberforum.ru)

```
(defun deep-product (w)
  (if (listp w)
      (reduce #'* w :key #'deep-product)
      w))

> (deep-product '(1 2 3 (3 6) 2 1 (2 4) 2))
3456
```

Решение 3.78.3

```
(defun deep-product (w)
  (if (atom w) w (reduce #'* w :key #'deep-product)))

> (deep-product '(1 2 3 (3 6) 2 1 (2 4) 2))
3456
```

Задача 3.79 sum-10+.lisp

Определить функцию, которая считает сумму чисел больше 10 в одноуровневом числовом списке.

Решение 3.79.1

```
(defun sum-10+ (w)
  (loop for a in w when (> a 10) sum a))

> (sum-10+ '(8 9 10 11 12))
23
```

Задача 3.80 sum-twinslisp

Определить функцию, которая находит сумму элементов списка, встречающихся ровно дважды.

Решение 3.80.1

```
(defun sum-twins (w &optional (v w))
  (cond ((null w) 0)
        ((= (count (car w) v) 2) (+ (car w) (sum-twins (cdr w) v)))
        ((sum-twins (cdr w) v))))

> (sum-twins '(1 2 3 4 5 2 3))
10
```

Решение 3.80.2

```
(defun sum-twins (w &optional (v w))
  (if w
      (if (= (count (car w) v) 2)
          (+ (car w) (sum-twins (cdr w) v))
          (sum-twins (cdr w) v))
      0))

> (sum-twins '(1 2 3 4 5 2 3))
10
```

Решение 3.80.3

```
(defun sum-twins (w &optional (v w))
  (if w
      (+ (if (= (count (car w) v) 2)
```

```
            (car w)
            0)
         (sum-twins (cdr w) v))
      0))

> (sum-twins '(1 2 3 4 5 2 3))
10
```

Решение 3.80.4

```
(defun sum-twins (w &aux (v w))
  (loop for a in w when (= (count a v) 2) sum a))

> (sum-twins '(1 2 3 4 5 2 3))
10
```

Задача 3.81 count-atoms.isp

Дан многоуровневый список. Определить функцию, возвращающую число атомов на всех уровнях списка.

Решение 3.81.1

```
(defun count-atoms (w)
  (cond ((null w) 0)
        ((atom (car w)) (1+ (count-atoms (cdr w))))
        (t (+ (count-atoms (car w)) (count-atoms (cdr w))))))

> (count-atoms '((a (b) ((c)) d) e))
5
```

Решение 3.81.2

```
(defun count-atoms (w)
  (cond ((null w) 0)
        ((atom w) 1)
        ((+ (count-atoms (car w)) (count-atoms (cdr w))))))

> (count-atoms '((a (b) ((c)) d) e))
5
```

Решение 3.81.3

```
(defun count-atoms (w)
  (if w
      (if (atom w)
          1
          (+ (count-atoms (car w)) (count-atoms (cdr w))))
      0))

> (count-atoms '((a (b) ((c)) d) e))
5
```

Решение 3.81.4 (автор – korvin, www.cyberforum.ru)

```
(defun count-atoms (w)
  (if (atom w) 1 (reduce #'+ w :key #'count-atoms)))

> (count-atoms '((a (b) ((c)) d) e))
5
```

Задача 3.82 glue-numbers.lisp

Определить функцию, преобразующую список (123 34 1234 45) в число 12334123445.

Решение 3.82.1 (автор – helter, www.cyberforum.ru)

```
(defun num-dgt (n)
  (do ((n n (floor n 10))
       (k 0 (1+ k)))
      ((< n 10) (1+ k))))

(defun glue-numbers (w)
  (reduce #'(lambda (a n)
              (+ (* a (expt 10 (num-dgt n)))
                 n))
          w
          :initial-value 0))

> (glue-numbers '(123 34 1234 45))
12334123445
> (glue-numbers #(123 34 1234 45))
12334123445
> (glue-numbers #(123 0 1234 45))
1230123445
> (glue-numbers #(1234 45 0))
1234450
> (glue-numbers #(0 1234 45))
123445
```

Решение 3.82.2

```
(defun digits (n) (if (< n 10) 1 (1+ (digits (floor n 10)))))

(defun glue (w &optional (n 0) &aux (a (car w)))
  (if w (+ (* a (expt 10 n)) (glue (cdr w) (+ n (digits a)))) 0))

(defun glue-numbers (w) (glue (reverse w)))

> (glue-numbers '(123 34 1234 45))
12334123445
> (glue-numbers '(123 0 1234 45))
1230123445
```

```
> (glue-numbers '(0 1234 45))
123445
> (glue-numbers '(10 0))
100
```

Решение 3.82.3

```
(defun glue-numbers (w)
  (parse-integer (apply #'concatenate
                        'string (mapcar #'write-to-string w))))
```

```
> (glue-numbers '(123 34 1234 45))
12334123445
11
> (glue-numbers '(0 123 34 0 1234 45 0))
1233401234450
14
```

Задача 3.83 prns.lisp

Определить функцию, которая считает произведение элементов многоуровневого списка, при этом проверять - если элемент не число, то пропускать его.

Решение 3.83.1

```
(defun prns (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 1)
        ((atom a) (if (numberp a)
                      (* a (prns d))
                      (prns d)))
        ((* (prns a) (prns d)))))
```

```
> (prns '(1 2 a 3 (3 b 6) 2 1 (2 4 c) 2 d))
3456
```

Решение 3.83.2

```
(defun prns (w &aux (a (car w)) (d (cdr w)))
  (if w (* (if (atom a) (if (numberp a) a 1) (prns a)) (prns d)) 1))
```

```
> (prns '(1 2 a 3 (3 b 6) 2 1 (2 4 c) 2 d))
3456
```

Решение 3.83.3

```
(defun prns (w)
  (if (atom w) (if (numberp w) w 1) (reduce #'* w :key #'prns)))
```

```
> (prns '(1 2 a 3 (3 b 6) 2 1 (2 4 c) 2 d))
3456
```

Задача 3.84 sum-ns.lisp

Дан список положительных целых чисел. Составить программу, вычисляющую сумму всех чисел, начинающихся на 1.

Решение 3.84.1

```
(defun sum-ns (w)
  (loop for a in w
        when (equal (subseq (write-to-string a) 0 1) "1")
          sum a))

> (sum-ns '(12 22 13))
25
```

Решение 3.84.2

```
(defun sum-ns (w)
  (loop for a in w when (eq (ns a) 1) sum a))

(defun ns (n)
  (if (< n 10) n (ns (truncate (/ n 10)))))

> (sum-ns '(12 22 13))
25
```

Решение 3.84.3

```
(defun sum-ns (w)
  (cond ((null w) 0)
        ((eq (ns (car w)) 1) (+ (car w) (sum-ns (cdr w))))
        ((sum-ns (cdr w)))))

(defun ns (n)
  (if (< n 10) n (ns (truncate (/ n 10)))))

> (sum-ns '(12 22 13))
25
```

Решение 3.84.4

```
(defun sum-ns (w)
  (cond ((null w) 0)
        ((> (ns (car w)) 1) (sum-ns (cdr w)))
        ((+ (car w) (sum-ns (cdr w))))))

(defun ns (n)
  (if (< n 10) n (ns (truncate (/ n 10)))))

> (sum-ns '(12 22 13))
25
```

Решение 3.84.5

```
(defun sum-ns (w)
  (apply #'+ (remove-if #'(lambda (a) (> (ns a) 1)) w)))

(defun ns (n)
  (if (< n 10) n (ns (truncate (/ n 10)))))

> (sum-ns '(12 22 13))
25
```

Задача 3.85 escalade.lisp

Для числового списка определить количество отрезков, на которых элементы строго возрастают (отрезок возрастания имеет минимальную длину 2).

Решение 3.85.1

```
(defun escalade (w &optional (n 0) f)
  (cond ((null (cdr w)) n)
        ((< (car w) (cadr w)) (escalade (cdr w) (if f n (1+ n)) t))
        ((escalade (cdr w) n nil))))

> (escalade '(1 2 3 2 3 4 1))
2
> (escalade '(1 2 3 2 3 4 1 1 7 7))
3
> (escalade '(1 2 3 2 3 4 1 1 1 7))
3
```

Решение 3.85.2

```
(defun escalade (w &optional (n 0) f)
  (if (cdr w)
      (if (< (car w) (cadr w))
          (escalade (cdr w) (if f n (1+ n)) t)
          (escalade (cdr w) n nil))
      n))

> (escalade '(1 2 3 2 3 4 1))
2
> (escalade '(1 2 3 2 3 4 1 1 7 7))
3
> (escalade '(1 2 3 2 3 4 1 1 1 7))
3
```

Задача 3.86 sum-int-10+.lisp

Дан числовой список. Найти сумму целых чисел больше 10.

```
Решение 3.86.1

(defun sum-int-10+ (w)
  (cond ((null w) 0)
        ((and (integerp (car w)) (> (car w) 10))
         (+ (car w) (sum-int-10+ (cdr w))))
        ((sum-int-10+ (cdr w)))))

> (sum-int-10+ '(20 1 1 1 1 1 1 1 20 20.5))
40

Решение 3.86.2

(defun sum-int-10+ (w)
  (loop for a in w when (and (integerp a) (> a 10)) sum a))

> (sum-int-10+ '(20 1 1 1 1 1 1 1 20 20.5))
40
```

Задача 3.87 list-inside.lisp

```
Составить функцию, которая определяет, есть ли список внутри списка.

Решение 3.87.1

(defun list-inside (w) (some #'listp w))

> (list-inside '(1 2 3 (4 5) 6))
T

Решение 3.87.2

(defun list-inside (w)
  (cond ((null w) nil)
        ((atom (car w)) (list-inside (cdr w)))
        (t t)))

> (list-inside '(1 2 3 (4 5) 6))
T

Решение 3.87.3

(defun list-inside (w)
  (when w (if (atom (car w)) (list-inside (cdr w)) t)))

> (list-inside '(1 2 3 (4 5) 6))
T

Решение 3.87.4

(defun list-inside (w)
  (when w (or (listp (car w)) (list-inside (cdr w)))))
```

```
> (list-inside '(1 2 3 (4 5) 6))
T
```

Задача 3.88 1+z+z^2/2+z^3/3.lisp

Определить функцию, считающую сумму ряда 1 + z + z^2/2 + z^3/3 + z^4/4.

Решение 3.88.1

```
(defun 1+z+z^2/2+z^3/3 (z n)
  (1+ (loop for n from 1 to n sum (/ (expt z n) n))))

> (1+z+z^2/2+z^3/3 -0.475 4)
0.61481524
```

Задача 3.89 over-n.lisp

Определить функцию, которая возвращает количество атомов-чисел из одноуровневого числового списка w больше, чем n.

Решение 3.89.1

```
(defun over-n (n w)
  (cond ((null w) 0)
        ((> (car w) n) (1+ (over-n n (cdr w))))
        (t (over-n n (cdr w)))))

> (over-n 3 '(1 2 3 4 5 6 7))
4
```

Решение 3.89.2

```
(defun over-n (n w)
  (if w
      (if (> (car w) n)
          (1+ (over-n n (cdr w)))
          (over-n n (cdr w)))
      0))

> (over-n 3 '(1 2 3 4 5 6 7))
4
```

Решение 3.89.3

```
(defun over-n (n w)
  (loop for a in w when (> a n) count a))

> (over-n 3 '(1 2 3 4 5 6 7))
4
```

Решение 3.89.4

```
(defun over-n (n w)
  (count-if #'(lambda (a) (> a n)) w))

> (over-n 3 '(1 2 3 4 5 6 7))
4
```

Задача 3.90 8th.lisp

Дан список, состоящий не менее чем из восьми элементов. Определить функцию, которая возвращает восьмой лемент списка.

Решение 3.90.1

```
(defun 8th (w) (car (cdr (cdr (cdr (cdr (cdr (cdr (cdr w)))))))))

> (8th '(1 2 3 4 5 6 7 8 9))
8
```

Решение 3.90.2

```
(defun 8th (w) (car (cdr (cdr (cdr (cdr (cdr (cdr (cdr w)))))))))

> (8th '(1 2 3 4 5 6 7 8 9))
8
```

Решение 3.90.3

```
(defun 8th (w) (nth 7 w))

> (8th '(1 2 3 4 5 6 7 8 9))
8
```

Решение 3.90.4

```
(defun 8th (w) (elt w 7))

> (8th '(1 2 3 4 5 6 7 8 9))
8
```

Решение 3.90.5

```
(defun 8th (w) (eighth w))

> (8th '(1 2 3 4 5 6 7 8 9))
8
```

Задача 3.91 count-aas.lisp

Определить функцию, которая возвращает количество уровней сложного списка на которых есть два не числовых атома.

Решение 3.91.1

```
(defun count-aas (w)
  (if w
    (+
     (if (> (count-if
              #'(lambda (a) (and (atom a) (not (numberp a)))) w) 1)
         1
         0)
     (count-if #'(lambda (a) (and (listp a) (count-aas a))) w))
    0))

> (count-aas '((a (2 b c 6) 7) d e ((9 f g) 12 (h 14) 15)))
3
```

Решение 3.91.2

```
(defun count-aas (w)
  (cond ((null w) 0)
        (t (+ (if (> (count-if
                       #'(lambda (a) (and (atom a)
                                          (not (numberp a)))) w) 1)
                  1
                  0)
              (count-if #'(lambda (a) (and (listp a)
                                           (count-aas a))) w)))))

> (count-aas '((a (2 b c 6) 7) d e ((9 f g) 12 (h 14) 15)))
3
```

Решение 3.91.3

```
(defun count-aas (w)
  (cond ((null w) 0)
        ((+ (if (> (count-if
                     #'(lambda (a) (and (atom a)
                                        (not (numberp a)))) w) 1)
                1
                0)
            (count-if #'(lambda (a) (and (listp a)
                                         (count-aas a))) w)))))

> (count-aas '((a (2 b c 6) 7) d e ((9 f g) 12 (h 14) 15)))
3
```

Решение 3.91.4

```
(defun count-aas (w)
  (cond (w (+ (if (> (count-if #'(lambda (a) (and (atom a)
                                                  (not (numberp a)))) w) 1)
```

```
                1
                0)
            (count-if #'(lambda (a) (and (listp a)
                                         (count-aas a))) w)))
        (0)))

> (count-aas '((a (2 b c 6) 7) d e ((9 f g) 12 (h 14) 15)))
3
```

Задача 3.92 lastt.lisp

Определить функцию, которая возвращает предпоследний элемент списка.

Решение 3.92.1

```
(defun llast (w) (car (last w 2)))

> (llast '(a b c))
B
```

Задача 3.93 unique.lisp

Имеется список, например (1 2 3 4 5 2 4 1). Нужно чтобы функция проверяла имеются ли повторяющиеся символы и писала в ответе или nil или t.

Решение 3.93.1

```
(defun unique (w)
  (cond ((null w) t)
        ((member (car w) (cdr w)) nil)
        ((unique (cdr w)))))

> (unique '(1 2 3))
T
> (unique '(1 2 2))
NIL
```

1. Если (null w), то достигли конца списка, вернули t - истина;
2. Если (member (car w) (cdr w)), то есть (car w) - голова списка входит в (cdr w) - хвост того же списка, то вернули nil - ложь, элемент встречается более одного раза;
3. Если не выполнилось ни одно из условий, то возвращаем (unique (cdr w)) - применяем функцию к хвосту списка (t - истина, в третьей строке - условие, которое всегда выполняется, если не выполнены условия в первой или во второй строке).

Решение 3.93.2

```
(defun unique (w)
  (or (null w) (and (not (member (car w) (cdr w))) (unique (cdr w)))))
```

```
> (unique '(1 2 3))
T
> (unique '(1 2 2))
NIL
```

Решение 3.93.3

```
(defun unique (w)
  (or (null w) (when (not (member (car w) (cdr w)))
                 (unique (cdr w)))))
```

```
> (unique '(1 2 3))
T
> (unique '(1 2 2))
NIL
```

Решение 3.93.4

```
(defun unique (w)
  (or (null w) (unless (member (car w) (cdr w)) (unique (cdr w)))))
```

```
> (unique '(1 2 3))
T
> (unique '(1 2 2))
NIL
```

Решение 3.93.5

```
(defun unique (w &optional acc ac (a (car w))(d (cdr w)))
  (cond ((null w) (nreverse acc))
        ((or (member a ac) (member a d))
         (unique d (cons nil acc) (cons a ac)))
        ((unique d (cons t acc) (cons a ac)))))
```

```
> (unique '(1 2 3 4 5 2 4 1))
(NIL NIL T NIL T NIL NIL NIL)
```

Решение 3.93.6

```
(defun unique (w &optional acc)
  (maplist #'(lambda (d) (let ((a (car d)))
                           (prog1 (and (not (member a acc))
                                       (not (member a (cdr d))))
                             (push a acc))))
           w))
```

```
> (unique '(1 2 3 4 5 2 4 1))
(NIL NIL T NIL T NIL NIL NIL)
```

Решение 3.93.7 *

```
* А может в качестве ответа быть так:
> (unique '(1 2 3 4 5 2 4 1))
```

```
Nill Nill T Nill T T T T

(defun unique (w)
  (when w (cond ((member (car w) (cdr w)) (cons nil (unique (cdr w))))
                ((cons t (unique (cdr w)))))))

> (unique '(1 2 3 4 5 2 4 1))
(NIL NIL T NIL T T T T)

Решение 3.93.8 *

(defun unique (w)
  (when w (cond ((member (car w) (cdr w))
                 (cons nil (unique (cdr w))))
                ((cons t (unique (cdr w)))))))

> (unique '(1 2 3 4 5 2 4 1))
(NIL NIL T NIL T T T T)
```

Задача 3.94 count-min.lisp

Определите функцию, возвращающую количество элементов, равных минимальному элементу числового списка.

```
Решение 3.94.1

(defun count-min (w &aux (n (reduce #'min w)))
  (count n w))

> (count-min '(1 2 3 1))
2
```

Задача 3.95 count-odd.lisp

Определите функцию, возвращающую количество нечетных чисел в списке.

```
Решение 3.95.1

(defun count-odd (w)
  (cond ((null w) 0)
        ((oddp (car w)) (1+ (count-odd (cdr w))))
        ((count-odd (cdr w)))))

> (count-odd '(1 2 3 4 5 6 7))
4

Решение 3.95.1

(defun count-odd (w)
  (count-if #'oddp w))
```

```
> (count-odd '(1 2 3 4 5 6 7))
4
```

Задача 3.96 (lambda (w) (if…).lisp

Известно, что один из четырех элементов списка (a1 a2 a3 a4) отличен от других.Получить номер этого элемента. Составить нерекурсивную программу в форме лямбда-вызова, используя функцию if.

Решение 3.96.1

```
> ((lambda (w) (if (and (equal (car w) (cadr w))
                        (equal (cadr w) (caddr w)))
                   4
                   (if (and (equal (car w) (cadr w))
                            (equal (cadr w) (cadddr w)))
                       3
                       (if (and (equal (car w) (caddr w))
                                (equal (caddr w) (cadddr w)))
                           2
                           1)))) '(1 0 0 0))
1
> ((lambda (w) (if (and (equal (car w) (cadr w))
                        (equal (cadr w) (caddr w)))
                   4
                   (if (and (equal (car w) (cadr w))
                            (equal (cadr w) (cadddr w)))
                       3
                       (if (and (equal (car w) (caddr w))
                                (equal (caddr w) (cadddr w)))
                           2
                           1)))) '(0 1 0 0))
2
> ((lambda (w) (if (and (equal (car w) (cadr w))
                        (equal (cadr w) (caddr w)))
                   4
                   (if (and (equal (car w) (cadr w))
                            (equal (cadr w) (cadddr w)))
                       3
                       (if (and (equal (car w) (caddr w))
                                (equal (caddr w) (cadddr w)))
                           2
                           1)))) '(0 0 1 0))
3
> ((lambda (w) (if (and (equal (car w) (cadr w))
                        (equal (cadr w) (caddr w)))
                   4
                   (if (and (equal (car w) (cadr w))
                            (equal (cadr w) (cadddr w)))
                       3
                       (if (and (equal (car w) (caddr w))
                                (equal (caddr w) (cadddr w)))
```

```
                          2
                          1)))) '(0 0 0 1))
4
```

Задача 3.97 sum-oddp.lisp

```
Вычислить сумму нечетных элементов списка.

Решение 3.97.1

(defun sum-oddp (w)
  (cond ((null w) 0)
        ((oddp (car w)) (+ (car w) (sum-oddp (cdr w))))
        (t (sum-oddp (cdr w)))))

> (sum-oddp '(1 2 3 4 5))
9

Решение 3.97.2

(defun sum-oddp (w)
  (if w (+ (if (oddp (car w)) (car w) 0) (sum-oddp (cdr w))) 0))

> (sum-oddp '(1 2 3 4 5))
9

Решение 3.97.3

(defun sum-oddp (w)
  (loop for a in w when (oddp a) sum a))

> (sum-oddp '(1 2 3 4 5))
9

Решение 3.97.4

(defun sum-oddp (w)
  (reduce #'+ (remove-if-not #'oddp w)))

> (sum-oddp '(1 2 3 4 5))
9

Решение 3.97.5

(defun sum-oddp (w)
  (loop for a in w sum (if (oddp a) a 0)))

> (sum-oddp '(1 2 3 4 5))
9

Решение 3.97.6
```

```
(defun sum-oddp (w)
  (reduce #'+ w :key #'(lambda (a) (if (oddp a) a 0))))

> (sum-oddp '(1 2 3 4 5))
9
```

Задача 3.98 monotonic.lisp

Определить предикат, который проверяет, является ли простой список чисел монотонной последовательностью.

Решение 3.98.1

```
(defun monotonic (w)
  (or (mono w #'<) (mono w #'>)))

(defun mono (w p)
  (cond ((null (cdr w)))
        ((funcall p (car w) (cadr w)) nil)
        ((mono (cdr w) p))))

> (monotonic '(3 2 2 1))
T
> (monotonic '(1 2 2 3))
T
> (monotonic '(10 2 2 3))
NIL
```

Решение 3.98.2

```
(defun monotonic (w)
  (or (mono w #'<=) (mono w #'>=)))

(defun mono (w p)
  (if (cdr w)
      (and (funcall p (car w) (cadr w))
           (mono (cdr w) p))
      t))

> (monotonic '(3 2 2 1))
T
> (monotonic '(1 2 2 3))
T
> (monotonic '(10 2 2 3))
NIL
```

Решение 3.98.3

```
(defun monotonic (w)
  (or (mono w #'<=) (mono w #'>=)))

(defun mono (w p)
```

```
  (loop for (a b) on w
        while b always (funcall p a b)))

> (monotonic '(3 2 2 1))
T
> (monotonic '(1 2 2 3))
T
> (monotonic '(10 2 2 3))
NIL
```

Решение 3.98.4

```
(defun monotonic (w)
  (or (apply #'>= w) (apply #'<= w)))

> (monotonic '(3 2 2 1))
T
> (monotonic '(1 2 2 3))
T
> (monotonic '(10 2 2 3))
NIL
```

Задача 3.99 count-minus.lisp

Определить функцию, вычисляющую количество отрицательных элементов в списке.

Решение 3.99.1

```
(defun count-minus (w)
  (cond ((null w) 0)
        ((minusp (car w)) (1+ (count-minus (cdr w))))
        ((count-minus (cdr w)))))

> (count-minus '(-3 -2 -1 0))
3
```

Решение 3.99.2

```
(defun count-minus (w)
  (if w (+ (if (minusp (car w)) 1 0) (count-minus (cdr w))) 0))

> (count-minus '(-3 -2 -1 0))
3
```

Решение 3.99.3

```
(defun count-minus (w)
  (loop for a in w count (minusp a)))

> (count-minus '(-3 -2 -1 0))
3
```

Решение 3.99.4

```
(defun count-minus (w)
  (count-if #'minusp w))

> (count-minus '(-3 -2 -1 0))
3
```

Задача 3.100 check-<.lisp

Определить, является ли массив упорядоченным по возрастанию. В случае отрицательного ответа, определить номер первого элемента, нарушившего такую упорядоченность.

Решение 3.100.1

```
(defun check-< (w &optional (n 1))
  (cond ((or (null w) (null (cdr w))))
        ((< (car w) (cadr w)) (check-< (cdr w) (1+ n)))
        (n)))

> (check-< '(1 2 3 4 5))
T
> (check-< '(1 2 3 1 5))
3
```

Задача 3.101 atom-n.lisp

Написать функцию, которая проверяет является ли агрумент списком из n атомов.

Решение 3.101.1

```
(defun atom-n (w n)
  (and (eq (length w) n) (every #'atom w)))

> (atom-n '(1 2 3) 3)
T
> (atom-n '(1 2) 3)
NIL
> (atom-n '(1 2 nil) 3)
T
> (atom-n '(1 2 '(a)) 3)
NIL
```

Задача 3.102 count-all.lisp

Дан список целых чисел. Требуется дать описание фрагмента программы, в котором определяется количество нечетных элементов, количество отрицательных четных и количество прочих элементов списка.

Решение 3.102.1

```
(defun count-all (w)
  (values (count-if #'oddp w)
          (count-if #'(lambda (a) (and (minusp a) (evenp a))) w)
          (count-if-not #'(lambda (a)
                            (or (oddp a)
                                (and (minusp a) (evenp a))))
                        w)))
```

```
> (count-all '(11 -63 51 -32 -6 44 -59 55 -67 -24 0 -85 29))
8
3
2
```

Решение 3.102.2

```
(defun count-all (w &aux
                  (a (count-if #'oddp w))
                  (b (count-if #'(lambda (a)
                                   (and (minusp a)
                                        (evenp a)))
                               w)))
  (values a b (- (length w) a b)))
```

```
> (count-all '(11 -63 51 -32 -6 44 -59 55 -67 -24 0 -85 29))
8
3
2
```

Решение 3.102.3

```
(defun count-all (w)
  (loop for a in w
        if (oddp a) count a into o
        else
        if (and (minusp a) (evenp a))
        count a into me
        else count a into r
        finally (return (values o me r))))
```

```
> (count-all '(11 -63 51 -32 -6 44 -59 55 -67 -24 0 -85 29))
8
3
2
```

Решение 3.102.4

```
(defun count-all (w &optional (o 0) (me 0) (r 0)
                  &aux (a (car w)) (d (cdr w)))
  (cond ((null w) `(,o ,me ,r))
        ((oddp a) (count-all d (1+ o) me r))
        ((and (minusp a) (evenp a)) (count-all d o (1+ me) r))
```

```
        ((count-all d o me (1+ r)))))

> (count-all '(11 -63 51 -32 -6 44 -59 55 -67 -24 0 -85 29))
(8 3 2)
```

Решение 3.102.5

```
(defun count-all (w)
  (loop for a in w
        if (oddp a)
        collect a into o
        else
        if (and (minusp a) (evenp a))
        collect a into me
        else collect a into r
        finally (return (values (length o)
                                (length me)
                                (length r)
                                o
                                me
                                r))))

> (count-all '(11 -63 51 -32 -6 44 -59 55 -67 -24 0 -85 29))
8
3
2
(11 -63 51 -59 55 -67 -85 29)
(-32 -6 -24)
(44 0)
```

Решение 3.102.6

```
(defun count-all (w &optional o me r &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (values (length o) (length me) (length r) o me r))
        ((oddp a) (count-all d (cons a o) me r))
        ((and (minusp a) (evenp a)) (count-all d o (cons a me) r))
        (t (count-all d o me (cons a r)))))

> (count-all '(11 -63 51 -32 -6 44 -59 55 -67 -24 0 -85 29))
8
3
2
(29 -85 -67 55 -59 51 -63 11)
(-24 -6 -32)
(0 44)
```

Структура 4 (функция (СПИСОК)) > (СПИСОК)

Задача 4.1 odd-even.lisp

Определить функцию перестановки местами соседних чётных и нечётных элементов в заданном списке.

Решение 4.1.1

```
(defun odd-even (w)
  (if (cdr w) (cons (cadr w) (cons (car w) (odd-even (cddr w)))) w))

> (odd-even '(a 1 a 1))
(1 A 1 A)
> (odd-even '(a 1 a 1 a))
(1 A 1 A A)
```

До тех пор - if, пока список w состоит хотя бы из двух элементов (cadr w) включаем - cons второй элемент (cadr w) списка w в начало списка, полученного в результате включения - cons первого элемента (car w) списка w в начало списка, полученного в результате вызова определяемой функции odd-even с хвостом хвоста (cddr w) списка w. Если условие if не выполнено, то есть в списке w меньше двух элементов, то возвращаем сам список w (или что от него осталось).

Задача 4.2 age.lisp

Написать функцию, которая принимает в качестве параметров: фамильное имя, год рождения и год смерти исторического деятеля и возвращает список из фамильного имени исторического деятеля и прожитых им лет.

Решение 4.2.1 *

```
(defun age (n b d)
  (list n (- d b)))

> (age 'bismarck 1815 1898)
(BISMARCK 83)
```

* Поскольку в условии не заданы даты рождения и смерти, то в отношении количества полных прожитых лет решение может содержать ошибку в один год.

Задача 4.3 ages.lisp

Написать функцию, которая принимает в качестве параметров список, состоящий из подсписков, содержащих фамильные имена, годы рождения и годы смерти исторических деятелей и возвращает список подсписков с фамильными именами исторических деятелей и числом прожитых ими лет.

Решение 4.3.1

```
(defun age (n b d)
  (list n (- d b)))
```

```
(defun ages (w)
  (mapcar #'(lambda (a) (apply #'age a)) w))

> (ages '((lincoln 1809 1865) (bismark 1815 1898)))
((LINCOLN 56) (BISMARK 83))

> (ages '((cromwell 1599 1658) (napoleon 1769 1821)))
((CROMWELL 59) (NAPOLEON 52))
```

Задача 4.4 w021.lisp

В числовом списке каждый элемент равен 0, 1 или 2. Определите функцию, переставляющую элементы списка так, чтобы сначала располагались все нули, затем все двойки и, наконец, все единицы.

Решение 4.4.1

```
(defun w021 (w &optional a0 a2 a1 &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (nconc a0 a2 a1))
        ((zerop a) (w021 d (cons a a0) a2 a1))
        ((= a 1) (w021 d a0 a2 (cons a a1)))
        ((w021 d a0 (cons a a2) a1))))

> (w021 '(1 0 2 1 2 0))
(0 0 2 2 1 1)
```

Комментарии (построчно):

1. Определяем функцию (**defun** от define/determine the function) с именем **w021**, в которую передается список **w** с необязательными (**&optional**) параметрами: переменными-аккумуляторами **a0**, **a2**, **a1** и вспомогательными (**&aux** - от auxiliary - вспомогательный) параметрами: переменными - голова **a** (функция **car**) и хвост **d** (функция **cdr**) списка **w**;
2. При условии **cond** (от слова condition - условие) если список **w** пустой, то возвращаем объединение **nconc** (можно вместо **nconc** применить **append**/дополнить) списков **a0 a2 a1**;
3. Функция **zerop** (от zero - ноль) проверяет, если первый элемент **a** списка **w** равен нулю, то вызываем функцию **w021** с хвостом **d**, списком из первого элемента **a**, присоединенного к списку-аккумулятору **a0** (функция **cons** включает новый элемент **a** в начало списка **a0**), и аккумуляторами **a2**, **a1**;
4. Функция **=** проверяет если первый элемент **a** списка **w** равен единице, то вызываем функцию **w021** с хвостом **d**, аккумуляторами **a0 a2** и списка из первого элемента **a**, присоединенного к списку-аккумулятору **a1** (функция **cons** включает новый элемент **a** в начало списка **a1**);
5. Если не выполнилось ни одно из трех вышеназванных условий, то вызывается функция **w021** с хвостом **d**, аккумулятором **a0**, списком из первого элемента **a**, присоединенного к списку-аккумулятору **a2** (функция **cons** включает новый элемент **a** в начало списка **a2**), и аккумулятором **a1**.

Задача 4.5 list-min.lisp

Написать функцию, ищущую минимум в заданном списке списков. Например: вход ((34 73 9 6) (5) (67 8 4 0)), выход (6 5 0). Использовать mapcar и lambda.

Решение 4.5.1

```
(defun list-min (w)
  (mapcar #'(lambda (a) (reduce #'min a)) w))

> (list-min '((34 73 9 6) (5) (67 8 4 0)))
(6 5 0)
```

Задача 4.6 bubble-sort.lisp

Реализовать сортировку методом пузырька.

Решение 4.6.1

```
(defun bubble-sort (w)
  (do ((i (- (length w) 1) (- i 1)))
      ((= i 0))
    (do ((j 0 (+ j 1)))
        ((= j i))
      (when (< (nth i w) (nth j w))
        (let ((tmp (nth i w)))
          (setf (nth i w) (nth j w) (nth j w) tmp)))))
  w)

> (bubble-sort '(3 2 1))
(1 2 3)
```

Решение 4.6.2

```
(defun bubble-sort (w)
  (do ((i (- (length w) 1) (- i 1)))
      ((= i 0))
    (do ((j 0 (+ j 1)))
        ((= j i))
      (when (< (nth i w) (nth j w))
        (rotatef (nth i w) (nth j w)))))
  w)

> (bubble-sort '(3 2 1))
(1 2 3)
```

Задача 4.7 del-third.lisp

Написать функцию, которая удаляет из списка w каждый третий элемент (3,6,9,12...и т.д.).

Решение 4.7.1

```
(defun del-third (w &aux (n 0))
  (mapcan #'(lambda (a) (incf n) (cond ((= n 3) (setq n 0) nil)
                                       ((list a))))
          w))

> (del-third '(1 2 3 4 5 6))
(1 2 4 5)
```

Решение 4.7.2

```
(defun del-third (w)
  (cond ((< (length w) 3) w)
        ((nconc (list (car w) (cadr w)) (del-third (nthcdr 3 w))))))

> (del-third '(1 2 3 4 5 6))
(1 2 4 5)
```

Решение 4.7.3

```
(defun del-third (w)
  (if (cddr  w)
      (nconc (list (car w) (cadr w)) (del-third (nthcdr 3 w)))
      w))

> (del-third '(1 2 3 4 5 6))
(1 2 4 5)
```

Решение 4.7.4

```
(defun del-third (w)
  (cond ((< (length w) 3) w)
        ((list* (car w) (cadr w) (del-third (nthcdr 3 w))))))

> (del-third '(1 2 3 4 5 6))
(1 2 4 5)
```

Решение 4.7.5

```
(defun del-third (w)
  (cond ((< (length w) 3) w)
        ((nconc (subseq w 0 2) (del-third (subseq w 3))))))

> (del-third '(1 2 3 4 5 6))
(1 2 4 5)
```

Решение 4.7.6

```
(defun del-third (w &aux (n 0))
  (delete-if #'(lambda (a) (incf n) (zerop (mod n 3))) w))

> (del-third '(1 2 3 4 5 6))
(1 2 4 5)
```

Решение 4.7.7

```
(defun del-third (w &aux (n 0))
  (delete 0 w :key #'(lambda (a) (incf n) (mod n 3))))

> (del-third '(1 2 3 4 5 6))
(1 2 4 5)
```

Решение 4.7.8

```
(defun del-third (w)
  (cond ((null w) nil)
        ((null (cdr w)) (cons (car w) nil))
        ((cons (car w) (cons (cadr w) (del-third (cdddr w)))))))

> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Решение 4.7.9

```
(defun del-third (w)
  (if (cddr w)
      (nconc (list (car w) (cadr w)) (del-third (nthcdr 3 w)))
      w))

> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Решение 4.7.10

```
(defun del-third (w)
  (if (cddr w) (list* (car w) (cadr w) (del-third (nthcdr 3 w))) w))

> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Решение 4.7.11

```
(defun del-third (w)
  (if (cddr w) (nconc (subseq w 0 2) (del-third (subseq w 3))) w))

> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Решение 4.7.12

```
(defun del-third (w)
  (reduce #'nconc
          (loop for a on w by #'cdddr
                if (cdr a) collect (list (car a) (cadr a))
                else collect (list (car a)))))
```

```
> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Решение 4.7.13

```
(defun del-third (w)
  (reduce #'nconc
          (loop for a on w by #'cdddr
                if (cdr a) collect `(,(car w) ,(cadr a))
                else collect `(,(car a)))))
```

```
> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Решение 4.7.14 (автор – nullxdth, cyberforum.ru)

```
(defun del-third (w)
  (remove-if (constantly t) w :start 3 :count 1))
```

```
> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Решение 4.7.15

```
(defun del-third (w)
  (loop for a on w by #'cdddr
        if (cdr a) nconc (list (car a) (cadr a))
        else nconc (list (car a))))
```

```
> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Решение 4.7.16

```
(defun del-third (w)
  (loop for a on w by #'cdddr
        if (cdr a) nconc `(,(car w) ,(cadr a))
        else nconc `(,(car a))))
```

```
> (del-third '(1 2 3 4 5 6 7))
(1 2 4 5 7)
```

Задача 4.8 order-elms.lisp

Проверить список, если в нем есть одинаковые элементы, то выстроить их подряд.

Решение 4.8.1

```
(defun order-elms (w)
  (cond ((= (length  (delete-duplicates w)) (length w)) w)
        ((numberp (car w))(sort w #'<))
```

```
        ((sort w #'string<))))

> (order-elms '(1 2 1))
(1 1 2)
> (order-elms '(a b a))
(A A B)
```

Задача 4.9 del-last3.lisp

Удалить из списка три последних элемента. Опишите функцию на примере списка (1 2 3 4 5).

Решение 4.9.1

```
(defun del-last3 (w)
  (delete-if #'atom w :count 3 :from-end t))

> (del-last3 '(0 1 2 3))
(0)
```

Решение 4.9.2

```
(defun del-last3 (w)
  (delete-if (constantly t) w :count 3 :from-end t))

> (del-last3 '(0 1 2 3))
(0)
```

Решение 4.9.3

```
(defun del-last3 (w)
  (delete-if #'identity w :count 3 :from-end t))

> (del-last3 '(0 1 2 3))
(0)
```

Решение 4.9.4

```
(defun del-last3 (w)
  (ldiff w (last w 3)))

> (del-last3 '(0 1 2 3))
(0)
```

Решение 4.9.5

```
(defun del-last3 (w)
  (nbutlast w 3))

> (del '(0 1 2 3))
(0)
```

Задача 4.10 reverse-els.lisp

Написать функцию, примером работы которой будет:

```
> (f '(a b c))
(c b a)
```

Решение 4.10.1

```
(defun reverse-els (w &optional acc)
  (cond ((null w) acc)
        ((reverse-els (cdr w) (cons (car w) acc)))))

> (reverse-els '(a b c))
(C B A)
```

&optional - указывает на то, что далее следует необязательный (опциональный) параметр
acc - переменная, накапливает результат (аккумулятор) и возвращает результат из второй строки

P.S. (без спецификатора &optional пришлось бы явно задавать второй параметр - nil)

Решение 4.10.2

```
(defun reverse-els (w acc)
  (cond ((null w) acc)
        ((reverse-els (cdr w) (cons (car w) acc)))))

> (reverse-els '(a b c) nil)
(C B A)
```

Задача 4.11 maxub.lisp

Построить функцию, которая заданному списку выдает подсписок, в котором больше всего элементов, пример: (d ’(a (b) (c (d)))) > (c (d)).

Решение 4.11.1

```
(defun maxub (w &optional m)
  (cond ((null w) m)
        ((and (listp (car w))
              (> (length (car w))
                 (length m))
              (maxub (cdr w) (car w))))
        ((maxub (cdr w) m))))

> (maxub '(a (b) (c (d))))
(C (D))
```

Решение 4.11.2

```
(defun maxub (w &optional m)
  (if w ((lambda (a)
           (if (and (listp a) (> (length a) (length m)))
               (maxub (cdr w) a)
               (maxub (cdr w) m)))
         (car w))
      m))

> (maxub '(a (b) (c (d))))
(C (D))
```

Задача 4.12 group3.lisp

Написать функцию преобразования списка: (a b c d e f) ->((a b c)(d e f)) с учетом того что количество троек элементов может быть любым. Элементы списков могут быть любыми объектами.

Решение 4.12.1

```
(defun group3 (w)
  (cond ((caddr w) (cons (list (car w) (cadr w) (caddr w))
                         (group3 (cdddr w))))
        (w (cons w nil))))

> (group3 '(a b c d e f))
((A B C) (D E F))
```

Решение 4.12.2

```
(defun group3 (w)
  (loop for v on w by #'cdddr collect
        (if (caddr v) (subseq v 0 3) (subseq v 0))))

> (group3 '(a b c d e f))
((A B C) (D E F))
```

Решение 4.12.3*

* Нет ни у кого мысли как сделать более высокоскоростной вариант без рекурсии?

```
(defun group3 (w &aux acv acw)
  (dolist (a w)
    (push a acv)
    (when (caddr acv)
      (push (nreverse acv) acw)
      (setq acv nil)))
  (when (> (length acv) 0) (push (nreverse acv) acw))
  (nreverse acw))

> (group3 '(a b c d e f))
```

```
((A B C) (D E F))
```

Решение 4.12.4*

```
(defparameter *a* nil)

(defun pack ()
  (dotimes (i 20000)
    (push "a" *a*)))

(defmacro while (test &rest body)
  `(do ()
    ((not ,test))
    ,@body))

(defun group3 (w acc)
  (while (caddr w)
    (setq acc (append acc (list (list (car w) (cadr w) (caddr w)))))
    (setq w (cdddr w)))
  (if (car w) (setq acc (append acc (list w))))
  acc)

> (pack)
NIL
> (time (group3 *a* nil))
Real time: 2.7221558 sec.
Run time: 2.6988173 sec.
Space: 177982208 Bytes
GC: 339, GC time: 1.8876121 sec.
(("a" "a" "a") ("a" "a" "a") ("a" "a" "a") …)
```

Решение 4.12.5*

```
(defparameter *a* nil)

(defun pack ()
  (dotimes (i 20000)
    (push "a" *a*)))

(defmacro while (test &rest body)
  `(do ()
    ((not ,test))
    ,@body))

(defun group3 (w acc)
  (while (caddr w)
    (setq acc (nconc acc (list (list (car w) (cadr w) (caddr w)))))
    (setq w (cdddr w)))
  (if (car w) (setq acc (nconc acc (list w))))
  acc)

> (pack)
NIL
```

```
> (time (group3 *a* nil))
Real time: 0.157009 sec.
Run time: 0.156001 sec.
Space: 213320 Bytes
(("a" "a" "a") ("a" "a" "a") ("a" "a" "a") …)
```

Решение 4.12.6

```
(defun group3 (w)
  (cond ((caddr w) (cons `(,(car w) ,(cadr w) ,(caddr w))
                         (group3 (cdddr w))))
        (w (cons w nil))))

> (group3 '(a b c d e f))
((A B C) (D E F))
```

Задача 4.13 list-company.lisp

Прайс компьютерного магазина описывается списком (наименование фирма параметры цена): ((Терминал Acer 15 4500) (Терминал Sumsung 17 7000)(Матплата Asus 238 2500) (Процессор Intel 2.5 3000)). Определить функцию возвращающую список фирм.

Решение 4.13.1

```
(defun list-company (w)
  (remove-duplicates (mapcar #'cadr w)))

> (list-company '((terminal acer 15 4500) (terminal samsung 17 7000)
(motherboard intel 238 2500) (processor intel 2.5 3000)))
(ACER SAMSUNG INTEL)
```

Задача 4.14 twice-atom.lisp

Поставлена задача удвоения всех атомов в подсписках любого уровня, т.е. из такого (2(3 5(7 8(2 6)))) получить (22(33 55(77 88(22 66)))).

Решение 4.14.1

```
(defun twice-atom (w &aux (a (car w))(d (cdr w)))
  (cond ((null w) nil)
        ((atom a) (cons a (cons a (twice-atom d))))
        ((cons (twice-atom a) (twice-atom d)))))

> (twice-atom '(2 (3 5 (7 8 (2 6)))))
(2 2 (3 3 5 5 (7 7 8 8 (2 2 6 6))))
```

Решение 4.14.2

```
(defun twice-atom (w &aux (a (car w))(d (cdr w)))
  (when w (if (atom a)
```

```
              (cons a (cons a (twice-atom d)))
              (cons (twins a) (twice-atom d)))))

> (twice-atom '(2 (3 5 (7 8 (2 6)))))
(2 2 (3 3 5 5 (7 7 8 8 (2 2 6 6))))
```

Задача 4.15 vowels-del.lisp

Задан список символов произвольного уровня вложенности, следует сформировать из него список, состоящий только из гласных букв. Вычислить число удаленных символов.

Решение 4.15.1

```
(defun vp (s)
  (find s '(a e i o u)))

(defun vs (w &optional acc (n 0) &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (list (reverse acc) n))
        ((vp a) (vs d (push a acc) n))
        ((vs d acc (1+ n)))))

(defun ft (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((ft (car w) (ft (cdr w) acc)))))

(defun vowels-del (w)
  (vs (ft w)))

> (vowels-del '((a c e) u))
((A E U) 1)
```

```
* Что делает каждая функция?

vp - проверяет, является ли символ гласной (vp - vowel p).
vs - возвращает список гласных символов и число удаленных символов (vs
- vowel symbol).
ft - выравнивает список в одноуровневый, плоский (ft - flat).
vowels-del - главная, вызывает ft и vs.

** Что означают слова &optional, acc, &aux, push, acc?

&optional - в списке параметров предваряет необязательный (опциональ-
ный) параметр.
&aux - в списке параметров предваряет дополнительные (вспомогательные)
переменные.
push - функция, заталкивает значение переменной a в acc - (push a
acc).
acc - переменная - список, накапливает результат (аккумулятор).
```

Задача 4.16 lets&nums.lisp

Определить функцию, которая по списку из букв и цифр (a b 1 2 c) строит два списка, один из букв, другой из цифр, отсортированных по возрастанию): (A B C) (1 2).

Решение 4.16.1

```
(defun lets&nums (w &optional ac acn &aux (a (car w)))
  (cond ((null w) (list (sort ac #'string<) (sort acn #'<)))
        ((numberp a) (lets&nums (cdr w) ac (push a acn)))
        ((lets&nums (cdr w) (push a ac) acn))))

> (lets&nums '(a b 1 2 c))
((A B C) (1 2))
```

Решение 4.16.2*

* При попытке выполнить функцию с заданными параметрами, у меня XLISP ругается: "error: unbound function - PUSH".

```
(defun lets&nums (w &optional ac acn &aux (a (car w)))
  (cond ((null w) (list (sort ac #'string<) (sort acn #'<)))
        ((numberp a) (lets&nums (cdr w) ac (push a acn)))
        ((lets&nums (cdr w) (push a ac) acn))))

(defun push (a w)
  (setf w (cons a w)))

> (lets&nums '(a b 1 2 c))
((A B C) (1 2))
```

Задача 4.17 nums-to-words.lisp

Написать функцию, которая переводит список цифр от 0 до 9 в список соответствующих им названий (строк).

Решение 4.17.1

```
(defun nums-to-words (w)
  (mapcar
   #'(lambda (a)
       (case a
         (0 "zero")
         (1 "one")
         (2 "two")
         (3 "three")
         (4 "four")
         (5 "five")
         (6 "six")
         (7 "seven")
         (8 "eight")
```

```
          (9 "nine")
          ("unknown")))
    w))

> (nums-to-words '(1 2 3))
("one" "two" "three")
```

Решение 4.17.2

```
(defun nums-to-words (w)
  (mapcar #'(lambda (a) (format nil "~r" a)) w))

> (nums-to-words '(1 2 3))
("one" "two" "three")
```

Решение 4.17.3

```
(defun nums-to-words (w)
  (loop for a in w collect (format nil "~r" a)))

> (nums-to-words '(1 2 3))
("one" "two" "three")
```

Задача 4.18 sec-blast-num.lisp

Если второй и предпоследний элементы списка - числовые атомы, то функция возвращает список, состоящий из этих элементов.

Решение 4.18.1

```
(defun sec-blast-num (w)
  (when (every #'numberp (list (cadr w) (car (last w 2))))
    (list (cadr w) (car (last w 2)))))

> (sec-blast-num '(a 1 b 2 c))
(1 2)
> (sec-blast-num '(a 1 b 2))
NIL
```

Задача 4.19 car-encl-list.lisp

Если первый элемент списка содержит списки, то функция возвращает этот элемент.

Решение 4.19.1

```
(defun car-encl-list (w)
  (when (and (listp (car w)) (some #'listp (car w))) (car w)))

> (car-encl-list '(((a) (b)) c))
((A) (B))
```

```
> (car-encl-list '((a) (b) c))
NIL
```

Задача 4.20 arithmetic-to-text.lisp

Преобразовать математические термины, содержащиеся в произвольном тексте в символьное выражение. В качестве результата выдать весь текст с формулами. Текст может содержать следующие мат. термины: "существует", "для любого", "сумма", "разность", "произведение", латинские буквы X, Y, Z. Заменить все эти слова на соответствующие знаки + - * итд.

Решение 4.20.1

```
(defun arithmetic-to-text (w)
  (mapcar
   #'(lambda (a)
       (case a
         (sum '+)
         (product '*)
         (x 'x)
         (z 'z)
         ("unknown")))
   w))

> (tr '(product x z))
(* X Z)
```

Решение 4.20.2*

```
* Возникло пару вопросов... что означает #' и если встречается слово
не входящее в "словарь" как сделать чтобы оно оставалось на месте?
например:
(arithmetic-to-text `(x sum z ravno z product x))
(x + z ravno z * x)

#' является сокращенной записью function (как ' - сокращенной записью
quote)

HyperSpec: 2.4.8.2 Sharpsign Single-Quote

Any expression preceded by #' (sharpsign followed by single-quote), as
in #'expression, is treated by the Lisp reader as an abbreviation for
and parsed identically to the expression (function expression). See
function. For example,
(apply #'+ l) == (apply (function +) l)

(defun arithmetic-to-text (w)
  (mapcar #'(lambda (a) (case a (sum '+) (product '*) (t a))) w))

> (arithmetic-to-text `(x sum z ravno z product x))
(X + Z RAVNO Z * X)
```

Задача 4.21 parse-pascal.lisp

Написать программу, которая преобразует цикл FOR произвольной программы на языке Паскаль в соответствие инструкции DO языка Лисп (считать что в теле цикла один оператор, выполняющий арифметическое действие, сложение или вычитание)

Решение 4.21.1

```
(defun parse-pascal
    (w &aux (a (cadr w)) (j (eq (nth 4 w) 'to)) (f (nth 7 w)))
  `(do ((,a ,(nth 3 w) (,(if j '1+ '1-) ,a)))
    ((,(if j '> '<) ,a ,(nth 5 w)) ,f)
    (setq ,f (,(nth 10 w) ,(nth 9 w) ,(nth 11 w)))))

> (parse-pascal '(for i = 1 to 10 do k := k + i))
(DO ((I 1 (1+ I))) ((> I 10) K) (SETQ K (+ K I)))
```

Функция pars-pascal заполняет do-форму из for-списка. Пример того, как работают в связке `- backquote и , - comma:

```
>`((+ 2 3) ,(+ 2 3))
((+ 2 3) 5)
```

1. Commas должны быть вложены в backquoted-список, поэтому в Вашем примере >(`(+ 2 3) ,(+ 2 3)) выдаст ошибку: READ: comma is illegal outside of backquote [Condition of type SYSTEM::SIMPLE-READER-ERROR];
2. Например: `(do ((,a ,(nth 3 w) ... , где do - необрабатываемый символ, не имя функции / ,a - код, переменная - второй элемент списка 'w'/ ,(nth 3 w) - код, функция - подставляет четвертый элемент списка 'w'; для backquote и comma можно применить метафору обратного переключателя или кнопки: backquote - положение 'вверх' (выключено – кнопка отжата) - режим данных *, comma - положение 'вниз' - (включено – кнопка утоплена) - режим кода.

_____

* Conrad Barski, M.D. - автор идеи 'data mode' и 'code mode': www.lisperati.com/syntax.html

Задача 4.22 arabic-to-roman.lisp

Написать простой транслятор способный переводить арабские цифры в римские в диапазоне от 0 до 100.

Решение 4.22.1

```
(defun arabic-to-roman (w)
  (mapcar #'(lambda (a) (format nil "~@r" a)) w))

(arabic-to-roman '(123 456 789))
("CXXIII" "CDLVI" "DCCLXXXIX")
```

```
Решение 4.22.2

defun arabic-to-roman (w)
  (mapcar #'(lambda (a) (intern (format nil "~@r" a))) w))

(arabic-to-roman '(123 456 789))
(CXXIII CDLVI DCCLXXXIX)
```

Задача 4.23 reverse-list.lisp

```
Написать функцию, которая обращает список. В качестве последнего эле-
мента списка аргумента допускается использовать NTL. Пример: (a b c) >
(c b a), (a(b)c(d)) > ((d)c(b)a).

Решение 4.23.1

(defun reverse-list (w)
  (reverse w))

(reverse-list '(a(b)c(d)))
((D) C (B) A)

... представить аналогичную рекурсивную функцию.

Решение 4.23.2

(defun reverse-list (w &optional ac)
  (cond ((null w) ac)
        ((reverse-list (cdr w) (cons (car w) ac)))))

(reverse-list '(a b c))
(C B A)
```

Задача 4.24 liat.lisp

```
Определите функцию, которая для заданного списка lst формирует список-
результат путем объединения результата реверсирования lst, результата
реверсирования хвоста lst, результата реверсирования хвоста хвоста lst
и так далее. Пример: для списка (1 2 3 4 5 6) результатом будет: (6 5
4 3 2 1 6 5 4 3 2 6 5 4 3 6 5 4 6 5 6).

Решение 4.24.1

(defun liat (w)
  (cond ((null w) nil)
        ((append (reverse w) (liat (cdr w))))))

> (liat '(1 2 3 4 5 6))
(6 5 4 3 2 1 6 5 4 3 2 6 5 4 3 6 5 4 6 5 6)

Решение 4.24.2
```

```
(defun liat (w)
  (cond ((null w) nil)
        ((nconc (reverse w) (liat (cdr w))))))

> (liat '(1 2 3 4 5 6))
(6 5 4 3 2 1 6 5 4 3 2 6 5 4 3 6 5 4 6 5 6)
```

Решение 4.24.3

```
(defun liat (w)
  (when w (nconc (reverse w) (liat (cdr w)))))

> (liat '(1 2 3 4 5 6))
(6 5 4 3 2 1 6 5 4 3 2 6 5 4 3 6 5 4 6 5 6)
```

Решение 4.24.4

```
(defun liat (w)
  (mapcon #'reverse w))

> (liat '(1 2 3 4 5 6))
(6 5 4 3 2 1 6 5 4 3 2 6 5 4 3 6 5 4 6 5 6)
```

Задача 4.25 reverse-subs.lisp

Реализовать функцию, которая в начальном списке изменяет все элементы-списки результатами их реверсирования. Реверсирование делать на всех уровнях вложения. Пример: Для списка (1((2 3)4) 5 6) результатом будет (1(4(3 2))5 6).

Решение 4.25.1

```
(defun reverse-subs (w)
  (cond ((null w) nil)
        ((atom (car w)) (cons (car w) (reverse-subs (cdr w))))
        ((cons (reverse-subs (reverse (car w)))
               (reverse-subs (cdr w))))))

> (reverse-subs '(1((2 3)4) 5 6))
(1 (4 (3 2)) 5 6)
```

Решение 4.25.2

```
(defun reverse-subs (w)
  (when w (if (atom (car w))
              (cons (car w) (reverse-subs (cdr w)))
              (cons (reverse-subs (reverse (car w)))
                    (reverse-subs (cdr w))))))

> (reverse-subs '(1 ((2 3) 4) 5 6))
(1 (4 (3 2)) 5 6)
```

```
Решение 4.25.3

(defun reverse-subs (w)
  (loop for a in w
        collect (if (atom a) a (reverse-subs (reverse a)))))

> (reverse-subs '(1((2 3)4) 5 6))
(1 (4 (3 2)) 5 6)

Решение 4.25.4

(defun reverse-subs (w)
  (mapcar #'(lambda (a)
              (if (atom a) a (reverse-subs (reverse a)))) w))

> (reverse-subs '(1((2 3)4) 5 6))
(1 (4 (3 2)) 5 6)
```

Задача 4.26 num-list-atom.lisp

```
На входе список. Функция сортирует список на три подсписка:
1. Числа.
2. Списки.
3. Атомы.

Решение 4.26.1.

(defun num-list-atom (w &optional n c s &aux (a (car w)))
  (cond ((null w) (list n c s))
        ((consp a) (num-list-atom (cdr w) n (push a c) s))
        ((numberp a) (num-list-atom (cdr w) (push a n) c s))
        ((num-list-atom (cdr w) n c (push a s)))))

> (num-list-atom '(a 1 (b c) 2 d))
((2 1) ((B C)) (D A))
```

Задача 4.27 del-multy.lisp

```
Создайте функцию, удаляющую в исходном списке комбинации из двух и бо-
лее повторяющихся элементов.

Решение 4.27.1

(defun del-multy (w &optional b)
  (cond ((null w) nil)
        ((or (equal (car w) b) (equal (car w) (cadr w)))
         (del-multy (cdr w) (car w)))
        ((cons (car w) (del-multy (cdr w) (car w))))))

> (del-multy '(a b b c b b b c c c d d))
```

```
(A C)
```

Задача 4.28 even-2-3.lisp

Дан список чисел. Написать функцию, возвращающую в случае первого четного элемента исходный список, в котором первые три числа возведены в квадрат, иначе - исходный список, в котором первые три числа возведены в куб.

Решение 4.28.1

```
(defun even-2-3 (w)
  (if (evenp (car w))
      (cons (expt (car w) 2)
            (cons (expt (cadr w) 2)
                  (cons (expt (caddr w) 2) (cdddr w))))
      (cons (expt (car w) 3)
            (cons (expt (cadr w) 3)
                  (cons (expt (caddr w) 3) (cdddr w))))))

> (even-2-3 '(4 5 6 7 8 9))
(16 25 36 7 8 9)
```

Задача 4.29 atbash.lisp

Имеется некоторая последовательность символов, зашифровать ее шифром atbash.

Решение 4.29.1

```
(defun atbash (w)
  (mapcar
   #'(lambda (az)
       (case az
         (a z)
         (b y)
         (c x)
         (d w)
         (e v)
         (f u)
         (g t)
         (h s)
         (i r)
         (j q)
         (k p)
         (l o)
         (m n)
         (n m)
         (o l)
         (p k)
         (q j)
```

```
         (r i)
         (s h)
         (t g)
         (u f)
         (v e)
         (w d)
         (x c)
         (y b)
         (z a)))
   w))

> (atbash '(e x x e g o e x s r g i))
(V C C V T L V C H I T R)

Решение 4.29.2

(defun atbash
    (w &aux
     (x '(a b c d e f g h i j k l m n o p q r s t u v w x y z))
     (z '(z y x w v u t s r q p o n m l k j i h g f e d c b a)))
  (when w (cons (nth (position (car w) x) z) (atbash (cdr w)))))

> (atbash '(e x x e g o e x s r g i))
(V C C V T L V C H I T R)

Решение 4.29.3

(defun atbash
    (w &aux
     (x '(a b c d e f g h i j k l m n o p q r s t u v w x y z))
     (z '(z y x w v u t s r q p o n m l k j i h g f e d c b a)))
  (loop for a in w collect (nth (position a x) z)))

> (atbash '(e x x e g o e x s r g i))
(V C C V T L V C H I T R)

Решение 4.29.4

(defun atbash
    (w &aux
     (x '(a b c d e f g h i j k l m n o p q r s t u v w x y z))
     (z '(z y x w v u t s r q p o n m l k j i h g f e d c b a)))
  (mapcar #'(lambda (a) (nth (position a x) z)) w))

> (atbash '(e x x e g o e x s r g i))
(V C C V T L V C H I T R)
```

Задача 4.30 sine-twins.lisp

Определить функцию, которая удаляет из многоуровневого списка повторяющиеся элементы на верхнем уровне.

Решение 4.30.1

```
;-member (повторяет функцию Лиспа member) проверяет наличие элемента в
;списке, но возвращает не сам элемент, не позицию, а ту часть списка,
;которая начинается с этого элемента, то есть положительное значение,
;поскольку все, что не nil (ложь, пустой список) - все t (истина).

(defun -member (a w)
    (cond ((null w) nil)
    ((equalp a (car w)) (cons (car w) (cdr w)))
    ((-member a (cdr w)))))

(defun sine-twins (w)
  (cond ((null w) nil)
        ((-member (car w) (cdr w)) (sine-twins (cdr w)))
        ((cons (car w) (sine-twins (cdr w))))))

> (sine-twins '((a b) c d (a b) c e))
(D (A B) C E)
```

Решение 4.30.2

```
(defun sine-twins (w)
  (when w (if (member (car w) (cdr w) :test #'equalp)
              (sine-twins (cdr w))
              (cons (car w) (sine-twins (cdr w))))))

> (sine-twins '((a b) c d (a b) c e))
(D (A B) C E)
```

Решение 4.30.3

```
(defun sine-twins (w &optional acc)
  (cond ((null w) (reverse acc))
        ((member (car w) acc :test #'equalp)
         (sine-twins (cdr w) acc))
        ((sine-twins (cdr w) (cons (car w) acc)))))

> (sine-twins '((a b) c d (a b) c e))
((A B) C D E)
```

Решение 4.30.4

```
(defun sine-twins (w)
  (remove-duplicates w :test #'equalp))

> (sine-twins '((a b) c d (a b) c e))
(D (A B) C E)
```

Задача 4.31 conc-asterisc.lisp

Дан чиловой список. Определите функцию, присоединяющую к началу каждо-

го отрицательного элемента символ *.

Решение 4.31.1

```
(defun conc (n)
  (intern (concatenate 'string "*" (write-to-string n))))

(defun conc-asterisc (w)
  (cond ((null w) nil)
        ((minusp (car w))
         (cons (conc (car w)) (conc-asterisc (cdr w))))
        ((cons (car w) (conc-asterisc (cdr w))))))

> (conc-asterisc '(-1 0 1))
(*-1 0 1)
```

Задача 4.32 wordy.lisp

Дана строка символов. Определить, есть ли повторение слов в этой строке. На экран вывести те слова, которые встречаются в строке больше одного раза.

Решение 4.32.1

```
(defun string-list (s)
  (read-from-string
   (concatenate
    'string "(" (delete-if-not #'(lambda (x)
                                   (or (alpha-char-p x)
                                       (equal x #\space)
                                       (equal x #\-)))
                               s) ")")))

(defun word-amount (w)
  (when w (cons (list (string-downcase (car w)) (count (car w) w))
                (word-amount (delete (car w) (cdr w))))))

(defun wordy (s)
  (mapcar #'car (remove-if
                 #'(lambda (a) (eq a 1))
                 (word-amount (string-list s)) :key #'cadr)))

> (wordy  "aaa aaa  bbb")
("aaa")
```

Задача 4.33 sum-matrixx.lisp

Определите функцию, которая вычисляет сумму элементов каждой строки матрицы, где матрица представлена в виде списка, каждая строка матрицы – подсписок.

Решение 4.33.1

```
(defun sum-matrixx (w)
  (when w (cons (apply #'+ (car w)) (sum-matrixx (cdr w)))))

>  (sum-matrixx '((1 3) (3 7)))
(4 10)
```

Решение 4.33.2

```
(defun sum-matrixx (w)
  (mapcar #'(lambda (a) (apply #'+ a)) w))

>  (sum-matrixx '((1 3) (3 7)))
(4 10)
```

Решение 4.33.3

```
(defun sum-matrix (w)
  (loop for a in w collect (apply #'+ a)))

>  (sum-matrixx '((1 3) (3 7)))
(4 10)
```

Задача 4.34 area.lisp

Определите функцию вычисления площади геометрических фигур и поверхностей.

Решение 4.34.1

```
(defconstant square 1)
(defconstant hexagon (* (sqrt 3) 1.5))
(defconstant circle pi)
(defconstant sphere (* 4 pi))
(defconstant torus (* 4 pi pi))

(defun area (shape r)
  (* shape r r))

> (area square 5)
25
> (area hexagon 5)
64.951904
> (area circle 5)
78.53981633974483096L0
> (area sphere 5)
314.15926535897932384L0
> (area torus 5)
986.96044010893586196L0
```

Задача 4.35 clear-twins.lisp

Заданы три списка. Удалить из первого списка все элементы остальных списков.

Решение 4.35.1

```
(defun clear-twins (w x z &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((or (member a x) (member a z)) (clear-twins d x z))
        ((cons a (clear-twins d x z)))))

> (clear-twins '(a b c) '(b) '(c))
(A)
```

Решение 4.35.2

```
(defun clear-twins (w x z)
  (set-difference (set-difference w x) z))

> (clear-twins '(a b c) '(b) '(c))
(A)
```

Решение 4.35.3

```
(defun clear-twins (w x z)
  (set-difference w (union x z)))

> (clear-twins '(a b c) '(b) '(c))
(A)
```

Задача 4.36 fold-n-m.lisp

Дан числовой одноуровневый список. Определить функцию, которая возвращает числа кратные двум заданным.

Решение 4.36.1

```
(defun fold-n-m (w n m)
  (cond ((null w) nil)
        ((= 0 (mod (car w) n) (mod (car w) m))
         (cons (car w) (fold-n-m (cdr w) n m)))
        ((fold-n-m (cdr w) n m))))

> (fold-n-m '(1 2 3 4 5 6 7 8 9 10 11 12) 2 3)
(6 12)
```

Задача 4.37 swing.lisp

Сгруппировать элементы списка в подсписки следуя закономерности по 1, 2, 3, 2 элемента.

Решение 4.37.1

```
(defun swing (w m &optional (n 1) (up t))
  (cond ((null (nthcdr n w)) (cons w nil))
        ((and (< n m) up) (cons (subseq w 0 n)
                                (swing (nthcdr n w) m (1+ n) t)))
        ((= n m) (cons (subseq w 0 n)
                       (swing (nthcdr n w) m (1- n) nil)))
        ((= n 1) (cons (subseq w 0 n)
                       (swing (nthcdr n w) m (1+ n) t)))
        ((cons (subseq w 0 n) (swing (nthcdr n w) m (1- n) nil)))))

> (swing '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16) 3)
((1) (2 3) (4 5 6) (7 8) (9) (10 11) (12 13 14) (15 16))
```

Задача 4.38 map-sum.lisp

Дан список многоуровневых подсписков. Необходимо построить список сумм каждого подсписка верхнего уровня.

Решение 4.38.1

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun map-sum (w)
  (mapcar #'(lambda (a) (apply #'+ a)) (mapcar #'flat w)))

> (map-sum '(((1 2 3)(4 5 6))((7 8 9))))
(21 24)
```

Решение 4.38.2

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun map-sum (w)
  (cond ((null w) nil)
        ((cons (apply #'+ (flat (car w))) (map-sum (cdr w))))))

> (map-sum '(((1 2 3)(4 5 6))((7 8 9))))
(21 24)
```

Решение 4.38.3

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
```

```
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun map-sum (w)
  (if (null w) nil (cons (apply #'+ (flat (car w)))
                         (map-sum (cdr w)))))

> (map-sum '(((1 2 3)(4 5 6))((7 8 9))))
(21 24)
```

Решение 4.38.4

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun map-sum (w)
  (when w (cons (apply #'+ (flat (car w))) (map-sum (cdr w)))))

> (map-sum '(((1 2 3)(4 5 6))((7 8 9))))
(21 24)
```

Решение 4.38.5

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun add-numbers (w)
  (cond ((null w) 0)
        ((+ (car w) (add-numbers (cdr w))))))

(defun map-sum (w)
  (cond ((null w) nil)
        ((cons (add-numbers (flat (car w))) (map-sum (cdr w))))))

> (map-sum '(((1 2 3)(4 5 6))((7 8 9))))
(21 24)
```

Решение 4.38.6

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun add-numbers (w)
  (if (null w) 0 (+ (car w) (add-numbers (cdr w)))))

(defun map-sum (w)
  (when w (cons (add-numbers (flat (car w)))
```

```
                (map-sum (cdr w)))))

> (map-sum '(((1 2 3)(4 5 6))((7 8 9))))
(21 24)
```

Задача 4.39 best-grades.lisp

Имеется список студентов с информацией об итогах сессии, в котором указаны фамилия, номер группы, оценки по трем предметам. Определить название предмета, который был сдан лучше всего.

Решение 4.39.1

```
(defun sum-progres (w)
  (list (caar w) (apply #'+ (mapcar #'cadr w))))

(defun best-course (w &aux (v (mapcar #'cddr w)))
  (list (mapcar #'car v) (mapcar #'cadr v) (mapcar #'caddr v)))

(defun best-grades (w)
  (caar (sort (mapcar #'sum-progres (best-course w))
              #'> :key #'cadr)))

> (best-grades '((cameron 111 (java 5) (ruby 4) (lisp 3)) (haberman
222 (java 3) (ruby 5) (lisp 3))))
RUBY
```

Задача 4.40 unique-subs.lisp

Написать программу определения и вывода подсписков, которые не входят ни в какие другие подсписки. Вхождение определяется в теоретико-множественном смысле.

Решение 4.40.1

```
(defun check-sub (a w)
  (cond ((null w) nil)
        ((subsetp a (car w)) t)
        ((check-sub a (cdr w)))))

(defun unique-subs (w &aux (v (cdr w)))
  (cond ((null w) nil)
        ((check-sub (car w) v) (unique-subs (cdr w)))
        ((cons (car w) (unique-subs (cdr w))))))

> (unique-subs '((a b) (a b c) (d e) (f g h)))
((A B C) (D E) (F G H))
```

Решение 4.40.2

```
(defun check-sub (a w)
```

```
  (when w (if (subsetp a (car w)) t (check-sub a (cdr w)))))

(defun unique-subs (w &aux (v (cdr w)))
  (when w (if (check-sub (car w) v)
              (unique-subs (cdr w))
              (cons (car w) (unique-subs (cdr w))))))

> (unique-subs '((a b) (a b c) (d e) (f g h)))
((A B C) (D E) (F G H))
```

Задача 4.41 permutate.lisp

Для заданного списка получить множество всех возможных перестановок его элементов.

Решение 4.41.1

```
(defun permutate (w)
  (cond ((null w) nil)
        ((null (cdr w)) (list w))
        ((loop for a in w
               nconc (mapcar #'(lambda (e) (cons a e))
                             (permutate (remove a w)))))))

> (permutate '(1 2 3))
((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2) (3 2 1))
```

Решение 4.41.2

```
(defun permutate (w)
  (when w (if (cdr w)
              (loop for a in w
                    nconc (mapcar #'(lambda (e) (cons a e))
                                  (permutate (remove a w))))
              (list w))))

> (permutate '(1 2 3))
((1 2 3) (1 3 2) (2 1 3) (2 3 1) (3 1 2) (3 2 1))
```

Задача 4.42 sine-penultimate.lisp

Дан список. Удалить предпоследний элемент.

Решение 4.42.1

```
(defun sine-penultimate (w)
  (if (cddr w)
      (cons (car w) (sine-penultimate (cdr w)))
      (when (cdr w) (list (cadr w)))))

> (sine-penultimate '(1 2 3 4))
```

```
(1 2 4)
```

Решение 4.42.2

```
(defun sine-penultimate (w)
  (if (cddr w)
      (cons (car w) (sine-penultimate (cdr w)))
      (when (cdr w) `(,(cadr w)))))

> (sine-penultimate '(1 2 3 4))
(1 2 4)
```

Задача 4.43 check-prime-numbers.lisp

Составить программу для проверки утверждения: «Результатами вычислений по формуле x * x +x + 17 при 0 ≤ x ≤ 15 являются простые числа».

Решение 4.43.1

```
(defun primep (n)
  (loop for i from 3 to (1- n) never (zerop (rem n i))))

(defun prime-numbers (n m)
  (loop for a from n to m collect (+ (* a a) a 17)))

(defun check-prime-numbers (n m)
  (every #'primep (prime-numbers n m)))

> (prime-numbers 0 15)
(17 19 23 29 37 47 59 73 89 107 127 149 173 199 227 257)
> (check-prime-numbers 0 15)
T
> (prime-numbers 0 16)
(17 19 23 29 37 47 59 73 89 107 127 149 173 199 227 257 289)
> (check-prime-numbers 0 16)
NIL
```

Задача 4.44 same-celsius.lisp

Составить программу, которая группирует дни месяца, когда температура воздуха была одинаковой.

Решение 4.44.1

```
(defun same-celsius (w)
 (mapcar
  #'(lambda (z) (mapcar #'car z))
  (remove-if-not
   #'cdr (mapcar
          #'(lambda (a) (remove-if-not
                         #'(lambda (e) (eq a e))
```

```
                        w :key #'cadr))
          (remove-duplicates (mapcar #'cadr w))))))


> (same-celsius '((1 15) (2 17) (3 15) (4 21) (5 19) (6 17) (7 17)))
((1 3) (2 6 7))
```

Задача 4.45 sum-lists.lisp

Дан список, который составлен из подсписков. Необходимо построить список сумм каждого подсписка.

Решение 4.45.1

```
(defun sum-lists (w)
  (when w (cons (apply #'+ (car w)) (sum-lists (cdr w)))))

> (sum-lists '((1 1) (1 1)))
(2 2)
```

Задача 4.46 minus>0.lisp

Дан числовой многоуровневый список. Определить функцию, возвращает список, состоящий из элементов исходного списка, отрицательные числа в котором заменены 0.

Решение 4.46.1

```
(defun minus>0 (w)
  (cond ((null w) nil)
        ((listp (car w)) (cons (minus>0 (car w)) (minus>0 (cdr w))))
        ((minusp (car w)) (cons 0 (minus>0 (cdr w))))
        ((cons (car w) (minus>0 (cdr w))))))

> (minus>0 '(1 ((-2 3) -4) 5 (6)))
(1 ((0 3) 0) 5 (6))
```

Задача 4.47 unique-atoms.lisp

Дан многоуровневый список. Определить функцию, которая возвращает список атомов, встречающихся в исходном списке ровно один раз.

Решение 4.47.1

```
(defun flat (w acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun unique (w v)
  (cond ((null w) nil)
```

```
        ((> (count (car w) v) 1) (unique (cdr w) v))
        ((cons (car w) (unique (cdr w) v)))))

(defun unique-atoms (w)
  (let ((v (flat w nil)))
    (unique v v)))

> (unique-atoms '(1 2 (((2 3 4))) 7 (8 (9 1))))
(3 4 7 8 9)
```

Решение 4.47.2

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun unique (w &optional (v w))
  (cond ((null w) nil)
        ((> (count (car w) v) 1) (unique (cdr w) v))
        ((cons (car w) (unique (cdr w) v)))))

(defun unique-atoms (w)
  (unique (flat w)))

> (unique-atoms '(1 2 (((2 3 4))) 7 (8 (9 1))))
(3 4 7 8 9)
```

Решение 4.47.3

```
(defun unique-atoms (w &optional acc ac
                     &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (set-difference acc ac))
        ((listp a)
         (unique-atoms a (unique-atoms d acc ac) ac))
        ((member a acc) (unique-atoms d acc (cons a ac)))
        ((unique-atoms d (cons a acc) ac))))

> (unique-atoms '(1 2 (((2 3 4))) 7 (8 (9 1))))
(4 3 9 8 7)
```

Решение 4.47.4

```
(defun unique-atoms (w &optional acc ac
                     &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (set-difference acc ac))
        ((listp a)
         (unique-atoms d (unique-atoms a acc ac) ac))
        ((member a acc) (unique-atoms d acc (cons a ac)))
        ((unique-atoms d (cons a acc) ac))))

> (unique-atoms '(1 2 (((2 3 4))) 7 (8 (9 1))))
(9 8 7 4 3)
```

Задача 4.48 intercom-section-marks.lisp

Дан список вещественных чисел (a1 b1 a2 b2....an bn), ai < bi. Рассматривая ai и bi, как левые и правые концы отрезков на одной и той же прямой, определить функцию, выдающую концы отрезка, являющегося пересечением всех этих отрезков. Если такого отрезка нет, то выдать nil.

Решение 4.48.1

```
(defun intercom (w)
  (when w (cons (loop for e from (car w) to (cadr w) collect e)
                (intercom (cddr w)))))

(defun intercom-section (w)
  (reduce #'intersection (intercom w)))

(defun intercom-section-marks (w &aux (v (intercom-section w)))
  (cons (car v) (last v)))

> (intercom-section-marks '(2 15 10 20 12 25))
(12 15)
```

Задача 4.49 append-reversed.lisp

Определить функцию, которая добавляет в конец списка все его элементы располагая их в обратном порядке.

Решение 4.49.1

```
defun slow-reverse (w)
  (cond ((null w) nil)
        ((-append (slow-reverse (cdr w)) (list (car w))))))

(defun -append (w v)
  (cond ((null w) v)
        ((cons (car w) (-append (cdr w) v)))))

(defun append-reversed (w)
  (-append w (slow-reverse w)))

> (append-reversed '(1 2 3))
(1 2 3 3 2 1)
```

Решение 4.49.2

```
(defun fast-reverse (w &optional ac)
  (if (null w) ac (fast-reverse (cdr w) (cons (car w) ac))))

(defun -append (w v)
  (cond ((null w) v)
        ((cons (car w) (-append (cdr w) v)))))
```

```
(defun append-reversed (w)
  (-append w (fast-reverse w)))

> (append-reversed '(1 2 3))
(1 2 3 3 2 1)
```

Решение 4.49.3

```
(defun -reverse (w)
  (when w (-append (-reverse (cdr w)) (list (car w)))))

(defun -append (w v)
  (if (null w) v (cons (car w) (-append (cdr w) v))))

(defun append-reversed (w)
  (-append w (-reverse w)))


> (append-reversed '(1 2 3))
(1 2 3 3 2 1)
```

Решение 4.49.4

```
(defun -append (w v)
  (if (null w) v (cons (car w) (-append (cdr w) v))))

(defun append-reversed (w &optional (v w) acc)
  (cond ((null w) (-append v acc))
        ((append-reversed (cdr w) v (cons (car w) acc)))))

> (append-reversed '(1 2 3))
(1 2 3 3 2 1)
```

Решение 4.49.5

```
(defun -append (w v)
  (if (null w) v (cons (car w) (-append (cdr w) v))))

(defun append-reversed (w &optional (v w) acc)
  (if (null w)
      (-append v acc)
      (append-reversed (cdr w) v (cons (car w) acc))))

> (append-reversed '(1 2 3))
(1 2 3 3 2 1)
```

Задача 4.50 cadrs.lisp

Определить функцию, которая строит список элементов, стоящих в исходном списке на четных местах.

Решение 4.50.1

```
(defun cadrs (w)
  (cond ((null (cdr w)) nil)
        ((cons (cadr w) (cadrs (cddr w))))))

> (cadrs '(1 2 3 4 5 6 7))
(2 4 6)
```

Решение 4.50.2

```
(defun cadrs (w)
  (loop for a on w by #'cddr when (cadr a) collect (cadr a)))

> (cadrs '(1 2 3 4 5 6 7))
(2 4 6)
```

Задача 4.51 rotate-last-two.lisp

Определить функцию, которая меняет местами последний и предпоследний элементы списка, если они есть.

Решение 4.51.1 (автор – VH, www.cyberforum.ru)

```
(defun rotate-last-two (w)
  (nconc (butlast w 2) (reverse (last w 2))))

> (rotate-last-two '(1 2 3 4 5 6 7))
(1 2 3 4 5 7 6)
> (rotate-last-two '(1 2))
(2 1)
> (rotate-last-two '(1))
(1)
> (rotate-last-two '())
NIL
```

Задача 4.52 atom-level.lisp

Определить функцию, которая определяет уровень атомов для заданного списка. Для списка (a (b (c)) d) a, d – атомы нулевого уровня; b – атом первого уровня; c – атом второго уровня. В списке имена атомов не могут повторяться

Решение 4.52.1

```
(defun atom-level (w &optional (n 0))
  (cond ((null w)  nil)
        ((atom (car w)) (cons (list (car w) n)
                              (atom-level (cdr w) n)))
        ((nconc (atom-level (car w) (+ n 1))
                (atom-level (cdr w) n)))))
```

```
> (atom-level '(a b (((c d)) e f)))
((A 0) (B 0) (C 3) (D 3) (E 1) (F 1))
```

Решение 4.52.2

```
(defun atom-level (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (cond ((null w)  nil)
        ((atom a) (cons (list a n) (atom-level d n)))
        ((nconc (atom-level a (1+ n)) (atom-level d n)))))
```

```
> (atom-level '(a b (((c d)) e f)))
((A 0) (B 0) (C 3) (D 3) (E 1) (F 1))
```

Решение 4.52.3

```
(defun atom-level (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (when w (nconc (if (atom a) (list (list a n)) (atom-level a (1+ n)))
                 (atom-level d n))))
```

```
> (atom-level '(a b (((c d)) e f)))
((A 0) (B 0) (C 3) (D 3) (E 1) (F 1))
```

Решение 4.52.4

```
(defun atom-level (w &optional (n 0) &aux (a (car w)))
  (when w (nconc (if (atom a) `((,a ,n)) (atom-level a (1+ n)))
                 (atom-level (cdr w) n))))
```

```
> (atom-level '(a b (((c d)) e f)))
((A 0) (B 0) (C 3) (D 3) (E 1) (F 1))
```

Задача 4.53 ascending-row.lisp

Дана целочисленная прямоугольная матрица размера m*n. Определить в каких строках элементы расположены в возрастающем порядке.

Решение 4.53.1

```
(defun ascending-row (w)
  (remove-if-not #'(lambda (a) (apply #'< a)) w))

(ascending-row '((1 2 3) (2 3 1) (1 2 3)))
((1 2 3) (1 2 3))
```

Решение 4.53.2

```
(defun ascending-row (w)
  (mapcan #'(lambda (a) (when (apply #'< a) (list a))) w))

(ascending-row '((1 2 3) (2 3 1) (1 2 3)))
((1 2 3) (1 2 3))
```

Решение 4.53.3

```
(defun ascending-row (w)
  (mapcar #'(lambda (a) (when (apply #'< a) a)) w))

> (ascending-row '((1 2 3) (2 3 1) (1 2 3)))
((1 2 3) NIL (1 2 3))
```

Задача 4.54 cadr-car.lisp

Написать функцию перестановки местами соседних элементов, стоящих на четных и нечетных местах в исходном списке.

Решение 4.54.1

```
(defun cadr-car (w)
  (cond ((null w) nil)
        ((null (cdr w)) w)
        ((cons (cadr w) (cons (car w) (cadr-car (cddr w)))))))

> (cadrcar '(1 2 3 4 5 6))
(2 1 4 3 6 5)
> (cadrcar '(1 2 3 4 5))
(2 1 4 3 5)
```

Задача 4.55 transpose-matrix.lisp

Определить функцию транспонирования матрицы. Транспонированная матрица – матрица, полученная из исходной матрицы заменой строк на столбцы.

Решение 4.55.1

```
(defun transpose-matrix (m) (apply #'mapcar #'list m))

> (transpose-matrix '((1 2 3) (4 5 6) (7 8 9)))
((1 4 7) (2 5 8) (3 6 9))
```

Решение 4.55.2

```
(defun cr (v c)
  (when v (cons (funcall c v) (cr (cdr v) c))))

(defun transpose-matrix (v)
  (when (car v) (cons (cr v #'caar)
                      (transpose-matrix (cr v #'cdar)))))

> (transpose-matrix '((1 2 3) (4 5 6) (7 8 9)))
((1 4 7) (2 5 8) (3 6 9))
```

Задача 4.56 delete-twins.lisp

Дан список. Удалить из него списки размерностью 2.

Решение 4.56.1

```
(defun delete-twins (w)
  (remove-if #'(lambda (a) (and (listp a) (= (length a) 2))) w))

> (delete-twins '(0 (1) (2 2)))
(0 (1))
```

Решение 4.56.2

```
(defun delete-twins (w)
  (remove-if #'(lambda (a)
                 (and (listp a) (cadr a) (null (cddr a))))
             w))

> (delete-twins '(0 (1) (2 2)))
(0 (1))
```

Решение 4.56.3

```
(defun delete-twins (w)
  (cond ((null w) nil)
        ((and (listp (car w)) (= (length (car w)) 2))
         (delete-twins (cdr w)))
        ((cons (car w) (delete-twins (cdr w))))))

> (delete-twins '(0 (1) (2 2)))
(0 (1))
```

Задача 4.57 allocate-cards.lisp

Двн список из 36 игральных карт: ((6 x) (6 y) (6 u) (6 z) (7 x) (7 y) (7 u) (7 z) (8 x) (8 y) (8 u) (8 z) (9 x) (9 y) (9 u) (9 z) (10 x) (10 y) (10 u) (10 z) (11 x) (11 y) (11 u) (11 z) (12 x) (12 y) (12 u) (12 z) (13 x) (13 y) (13 u) (13 z) (14 x) (14 y) (14 u) (14 z)). Необходимо размешать и разделить карты между игроками. Количество игроков может быть от 2 до 4.

Решение 4.57.1

```
(defun oc (w n &optional (m 0) acc ac &aux (a (car w)) (d (cdr w)))
   (cond ((null w) (cons ac acc))
         ((= m n) (oc d n 1 (cons ac acc) (list a)))
         ((oc d n (1+ m) acc (cons a ac)))))

(defun drop (n w)
  (remove n w :test #'equalp))

(defun shuffle (w &aux (n (nth (random 35) w)))
  (when w (if n (cons n (shuffle (drop n w))) (shuffle (drop n w)))))
```

```
(defun allocate-cards (w n &aux (z (length w)))
  (oc (shuffle w) (floor (/ z n))))

> (allocate-cards ' ((6 x) (6 y) (6 u) (6 z) (7 x) (7 y) (7 u) (7 z)
(8 x) (8 y) (8 u) (8 z) (9 x) (9 y) (9 u) (9 z) (10 x) (10 y) (10 u)
(10 z) (11 x) (11 y) (11 u) (11 z) (12 x) (12 y) (12 u) (12 z) (13 x)
(13 y) (13 u) (13 z) (14 x) (14 y) (14 u) (14 z)) 4)
(((7 Z) (11 Y) (14 X) (10 Y) (8 U) (9 Y) (6 U) (12 X) (14 U))
 ((7 Y) (8 Z) (8 X) (7 X) (6 X) (10 U) (10 X) (14 Z) (9 X))
 ((9 Z) (11 X) (11 Z) (13 U) (7 U) (13 X) (9 U) (12 U) (13 Y))
 ((6 Y) (10 Z) (6 Z) (12 Y) (8 Y) (12 Z) (11 U) (14 Y) (13 Z)))
```

Решение 4.57.2

```
(defun oc (w n m acc ac)
   (cond ((null w) (cons ac acc))
         ((= m n) (oc (cdr w) n 1 (cons ac acc) (list (car w))))
         ((oc (cdr w) n (1+ m) acc (cons (car w) ac)))))

(defun drop (n w)
  (remove n w :test #'equalp))

(defun shuffle (w)
  ((lambda (w n)
     (when w
       (if n (cons n (shuffle (drop n w))) (shuffle (drop n w)))))
   w (nth (random 35) w)))

(defun allocate-cards (w n)
  (oc (shuffle w) (floor (/ (length w) n)) 0 nil nil))

> (allocate-cards ' ((6 x) (6 y) (6 u) (6 z) (7 x) (7 y) (7 u) (7 z)
(8 x) (8 y) (8 u) (8 z) (9 x) (9 y) (9 u) (9 z) (10 x) (10 y) (10 u)
(10 z) (11 x) (11 y) (11 u) (11 z) (12 x) (12 y) (12 u) (12 z) (13 x)
(13 y) (13 u) (13 z) (14 x) (14 y) (14 u) (14 z)) 4)
(((8 U) (6 Y) (10 U) (13 U) (14 U) (12 X) (9 U) (10 Y) (13 X))
 ((11 Z) (8 Z) (9 Y) (8 Y) (11 X) (11 Y) (7 X) (10 X) (13 Z))
 ((6 U) (6 X) (9 X) (7 Y) (10 Z) (14 Y) (12 U) (6 Z) (12 Y))
 ((8 X) (14 X) (11 U) (7 Z) (13 Y) (14 Z) (9 Z) (7 U) (12 Z)))
```

Решение 4.57.3

```
(defun drop (n w)
  (remove n w :test #'equalp))

(defun shuffle (w)
  ((lambda (w n)
     (when w
       (if n (cons n (shuffle (drop n w))) (shuffle (drop n w)))))
   w (nth (random 35) w)))

(defun 2-3-4 (w n)
```

```
  (cond ((= n 2) (list (subseq w 0 18) (subseq w 18)))
        ((= n 3) (list (subseq w 0 12)
                       (subseq w 12 24)
                       (subseq w 24)))
        ((list (subseq w 0 9)
               (subseq w 9 18)
               (subseq w 18 27)
               (subseq w 27)))))

(defun allocate-cards (w n)
  (2-3-4 (shuffle w) n))

> (allocate-cards ' ((6 x) (6 y) (6 u) (6 z) (7 x) (7 y) (7 u) (7 z)
(8 x) (8 y) (8 u) (8 z) (9 x) (9 y) (9 u) (9 z) (10 x) (10 y) (10 u)
(10 z) (11 x) (11 y) (11 u) (11 z) (12 x) (12 y) (12 u) (12 z) (13 x)
(13 y) (13 u) (13 z) (14 x) (14 y) (14 u) (14 z)) 4)
(((7 Y) (12 X) (7 X) (12 Z) (8 X) (13 Z) (13 Y) (11 Y) (7 Z))
 ((9 X) (14 Z) (10 U) (7 U) (14 X) (6 U) (6 Y) (11 X) (11 U))
 ((10 Z) (6 X) (9 Y) (12 U) (10 Y) (14 Y) (8 U) (14 U) (9 Z))
 ((13 U) (11 Z) (9 U) (8 Y) (10 X) (6 Z) (12 Y) (8 Z) (13 X)))
```

Решение 4.57.4

```
(defun drop (n w) ; удаляет карту из списка
  (remove n w :test #'equalp))

(defun shuffle (w) ; тасует колоду
  (let ((n (nth (random 35) w)))
     (when w
       (if n (cons n (shuffle (drop n w))) (shuffle (drop n w))))))

(defun s (&rest m) (apply #'subseq m)) ; раздача одному игроку

(defun 2-3-4 (w n) ; раздает 2, 3 или 4 игрокам
  (cond ((= n 2) (list (s w 0 18) (s w 18)))
        ((= n 3) (list (s w 0 12) (s w 12 24) (s w 24)))
        ((list (s w 0 9) (s w 9 18) (s w 18 27) (s w 27)))))

(defun allocate-cards (w n) ; тасует и раздает 36 карт
  (2-3-4 (shuffle w) n))

> (allocate-cards ' ((6 x) (6 y) (6 u) (6 z) (7 x) (7 y) (7 u) (7 z)
(8 x) (8 y) (8 u) (8 z) (9 x) (9 y) (9 u) (9 z) (10 x) (10 y) (10 u)
(10 z) (11 x) (11 y) (11 u) (11 z) (12 x) (12 y) (12 u) (12 z) (13 x)
(13 y) (13 u) (13 z) (14 x) (14 y) (14 u) (14 z)) 4)
(((12 Z) (7 U) (9 Z) (14 Z) (13 Y) (7 Z) (11 U) (14 X) (8 X))
 ((12 Y) (6 Z) (12 U) (14 Y) (10 Z) (7 Y) (9 X) (6 X) (6 U))
 ((13 Z) (10 X) (7 X) (11 Y) (11 X) (8 Y) (9 Y) (8 Z) (11 Z))
 ((13 X) (10 Y) (9 U) (12 X) (14 U) (13 U) (10 U) (6 Y) (8 U)))
```

Задача 4.58 average-sine.lisp

На вход подается список чисел A. Вычислить новый список чисел B, где Bi равно среднему арифметическому всего списка A, кроме Ai. То есть, если список A = (1 2 3), то B1 = (2+3)/2; B2 = (1+3)/2; B3 = (1+2)/2.

Решение 4.58.1

```
(defun average-sine (w &aux (n (1- (length w))) (m (apply #'+ w)))
  (when (> n 1) (mapcar #'(lambda (a) (/ (- m a) n)) w)))

> (average-sine '(2 3 5 6))
(14/3 13/3 11/3 10/3)
```

Решение 4.58.2

```
(defun average-sine (w &aux (n (1- (length w))) (m (apply #'+ w)))
  (when (> n 1) (mapcar #'(lambda (a) (float (/ (- m a) n))) w)))

> (average-sine '(2 3 5 6))
(4.6666665 4.3333335 3.6666667 3.3333333)
```

Решение 4.58.3

```
(defun average-sine (w)
  (loop for a in w collect (/ (- (apply #'+ w) a) (1- (length w)))))

> (average-sine '(2 3 5 6))
(14/3 13/3 11/3 10/3)

> (average-sine '(2 3 5 6))
(14/3 13/3 11/3 10/3)
```

Решение 4.58.4

```
(defun average-sine (w &aux (n (1- (length w))) (m (apply #'+ w)))
  (loop for a in w collect (/ (- m a) n)))

> (average-sine '(2 3 5 6))
(14/3 13/3 11/3 10/3)
> (average-sine '(2 3 5 6))
(14/3 13/3 11/3 10/3)
```

Решение 4.58.5

```
(defun average-sine (w)
  (let ((n (1- (length w))) (m (apply #'+ w)))
        (loop for a in w collect (/ (- m a) n))))

> (average-sine '(2 3 5 6))
(14/3 13/3 11/3 10/3)
```

Решение 4.58.6 * при помощи labels - обернуть во вложенную функцию average с параметрами n = (1- (length w)) и m = (apply #'+ w):

```
(defun average-sine (w)
  (labels ((average (n m)
             (loop for a in w collect (/ (- m a) n))))
    (average (1- (length w)) (apply #'+ w))))

> (average-sine '(2 3 5 6))
(14/3 13/3 11/3 10/3)
```

Решение 4.58.7 * используя lambda - обернуть во вложенную безымянную функцию с параметрами n = (1- (length w)) и m = (apply #'+ w):

```
(defun average-sine (w)
  ((lambda (n m)
     (loop for a in w collect (/ (- m a) n)))
   (1- (length w)) (apply #'+ w)))

> (average-sine '(2 3 5 6))
(14/3 13/3 11/3 10/3)
```

Задача 4.59 not-zero.lisp

Определить в списке ненулевые элементы и создать из них список.

Решение 4.59.1

```
(defun not-zero (w)
  (cond ((null w) nil)
        ((zerop (car w)) (not-zero (cdr w)))
        (t (cons (car w) (not-zero (cdr w))))))

> (not-zero '(0 1))
(1)
```

Решение 4.59.2

```
(defun not-zero (w)
  (when w (if (zerop (car w))
              (not-zero (cdr w))
              (cons (car w) (not-zero (cdr w))))))

> (not-zero '(0 1))
(1)
```

Решение 4.59.3

```
(defun not-zero (w)
  (loop for a in w when (not (zerop a)) collect a))

> (not-zero '(0 1))
(1)
```

Решение 4.59.4

```
(defun not-zero (w)
  (remove-if #'zerop w))

> (not-zero '(0 1))
(1)
```

Решение 4.59.5

```
(defun not-zero (w)
  (remove 0 w))

> (not-zero '(0 1))
(1)
```

Задача 4.60 1..4>zero.lisp

Элементы списка w, значения которых лежат в интервале [0,4], заменить на 0.

Решение 4.60.1

```
(defun 1..4>zero (w)
  (cond  ((null w) nil)
         ((listp (car w)) (cons (1..4>zero (car w))
                                (1..4>zero (cdr w))))
         ((<= 1 (car w) 4) (cons 0 (1..4>zero (cdr w))))
         ((cons (car w) (1..4>zero (cdr w))))))

> (1..4>zero '(((-2) -1) 0 1 (2) (3 (4)) 5 6 7))
(((-2) -1) 0 0 (0) (0 (0)) 5 6 7)
```

Решение 4.60.2

```
(defun 1..4>zero (w)
  (mapcar #'(lambda (a) (if (atom a)
                            (if (<= 1 a 4) 0 a)
                            (1..4>zero a))) w))

> (1..4>zero '(((-2) -1) 0 1 (2) (3 (4)) 5 6 7))
(((-2) -1) 0 0 (0) (0 (0)) 5 6 7)
```

Решение 4.60.3

```
(defun 1..4>zero (w)
  (loop for a in w collect (if (atom a)
                               (if (<= 1 a 4) 0 a)
                               (1..4>zero a))))

> (1..4>zero '(((-2) -1) 0 1 (2) (3 (4)) 5 6 7))
(((-2) -1) 0 0 (0) (0 (0)) 5 6 7)
```

Задача 4.61 odd/even.lisp

Дан список. Вывести в виде отдельных списков его нечетные и четные элементы.

Решение 4.61.1

```
(defun odd/even (w)
  (labels ((oddeven (v odds evens)
             (cond ((null v) (list (reverse odds) (reverse evens)))
                   ((oddp (car v)) (oddeven (cdr v)
                                            (cons (car v) odds)
                                            evens))
                   ((oddeven (cdr v) odds (cons (car v) evens))))))
    (oddeven w nil nil)))

> (odd/even '(1 2 3 4 5 6))
((1 3 5) (2 4 6))
```

Решение 4.61.2

```
(defun odd/even (w)
  (loop for i in w
        if (oddp i)
        collect i into odds
        else collect i into evens
        finally (return (values odds evens))))

> (odd/even '(1 2 3 4 5 6))
(1 3 5)
(2 4 6)
```

Задача 4.62 remove-twins.lisp

Создать функцию, которая должна выполнять преобразование сложного многоуровневого списка таким образом, чтобы в результате его элементы, которые в списке повторяются, возможно и неоднократно, остались в единственных экземплярах. Причем, так должно происходить и в первом уровне списка, и независимым образом, в подсписках любого уровня. То есть, после преобразования список может содержать по несколько одинаковых элементов, но в разных подсписках; в одном же подсписке повторы должны быть исключены.

Решение 4.62.1

```
(defun remove-twins (w)
  (cond ((null w) nil)
        ((listp (car w)) (cons (remove-twins (car w))
                               (remove-twins (cdr w))))
        ((member (car w) (cdr w)) (remove-twins (cdr w)))
        ((cons (car w) (remove-twins (cdr w))))))
```

```
> (remove-twins '(1 (1 1) a b (a a (b b))))
(1 (1) A B (A (B)))
```

Решение 4.62.2

```
(defun remove-twins (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((listp a) (cons (remove-twins a) (remove-twins d)))
        ((member a d) (remove-twins d))
        ((cons a (remove-twins d)))))

> (remove-twins '(1 (1 1) a b (a a (b b))))
(1 (1) A B (A (B)))
```

Решение 4.62.3 * удалять последующие вхождения повторяющихся элементов

```
(defun remove-twins (w)
  (cond ((null w) nil)
        ((listp (car w)) (cons (remove-twins (car w))
                               (remove-twins (cdr w))))
        ((cons (car w)
               (remove-twins (remove (car w) (cdr w)))))))

> (remove-twins '(0 1 (2 3 4 (5 1 5) 5) 4 2))
(0 1 (2 3 4 (5 1) 5) 4 2)
```

Решение 4.62.4

```
(defun remove-twins (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((listp a) (cons (remove-twins a) (remove-twins d)))
        ((member a d) (remove-twins d))
        ((cons a (remove-twins d)))))

> (remove-twins '(1 (1 1) a b (a a (b b))))
(1 (1) A B (A (B)))
```

Задача 4.63 pair-elms.lisp

Составить функцию, группирующую попарно смежные элементы списка в подсписки. Если выходной список состоит из нечетного числа элементов, то последний элемент отбрасывается.

Решение 4.63.1

```
(defun pair-elms (w)
  (when (cdr w) (cons (list (car w) (cadr w)) (pair-elms (cddr w)))))

> (pair-elms '(1 a 2 b))
((1 A) (2 B))
> (pair-elms '(1 a 2))
((1 A))
```

Решение 4.63.2

```
(defun pair-elms (w)
  (when (cdr w) (cons `(,(car w) ,(cadr w)) (pair-elms (cddr w)))))

> (pair-elms '(a a a a a a a a))
((A A) (A A) (A A) (A A))
> (pair-elms '(a a a a a a a))
((A A) (A A) (A A))
```

Задача 4.64 reverse-mxx.lisp

Превратить матрицу, переписав в списках, являющихся элементами данной матрицы, элементе упомянутых списков в обратном порядке (только в верхнем уровне этих списков). Понятно, что списки, о которых идет речь, должны быть под списками уже третьего уровня в исходном списке котором представлена сама матрица.

Решение 4.64.1

```
(defun reverse-mx (w)
  (cond ((null w) nil)
        ((cons (reverse (car w)) (reverse-mx (cdr w))))))

(defun reverse-mxx (w)
  (cond ((null w) nil)
        ((cons (reverse-mx (car w)) (reverse-mxx (cdr w))))))

> (reverse-mxx '(((a b) (c d) (e f)) ((a b) (c d) (e f)))
(((B A) (D C) (F E)) ((B A) (D C) (F E)))
```

Решение 4.64.2

```
(defun reverse-mx (w)
  (if (null w) nil (cons (reverse (car w)) (reverse-mx (cdr w)))))

(defun reverse-mxx (w)
  (if (null w) nil (cons (reverse-mx (car w)) (reverse-mxx (cdr w)))))

> (reverse-mxx '(((a b) (c d) (e f)) ((a b) (c d) (e f)))
(((B A) (D C) (F E)) ((B A) (D C) (F E)))
```

Решение 4.64.3

```
(defun reverse-mx (w)
  (when w (cons (reverse (car w)) (reverse-mx (cdr w)))))

(defun reverse-mxx (w)
  (when w (cons (reverse-mx (car w)) (reverse-mxx (cdr w)))))

> (reverse-mxx '(((a b) (c d) (e f)) ((a b) (c d) (e f)))
```

```
(((B A) (D C) (F E)) ((B A) (D C) (F E)))
```

Решение 4.64.4

```
(defun reverse-mxx (w)
  (loop for v in w collect (loop for a in v collect (reverse a))))

> (reverse-mxx '(((a b) (c d) (e f)) ((a b) (c d) (e f)))
(((B A) (D C) (F E)) ((B A) (D C) (F E)))
```

Решение 4.64.5

```
(defun reverse-mx (w)
  (mapcar #'reverse w))

(defun reverse-mxx (w)
  (mapcar #'reverse-mx w))

> (reverse-mxx '(((a b) (c d) (e f)) ((a b) (c d) (e f)))
(((B A) (D C) (F E)) ((B A) (D C) (F E)))
```

Решение 4.64.6

```
(defun reverse-mxx (w)
  (cond ((null w) nil)
        ((atom (car w)) (reverse w))
        ((cons (reverse-mxx (car w)) (reverse-mxx (cdr w))))))

> (reverse-mxx '(((a b) (c d) (e f)) ((a b) (c d) (e f))))
(((B A) (D C) (F E)) ((B A) (D C) (F E)))
```

Решение 4.64.7

```
(defun reverse-mx (w)
  (loop for (a b) in w collect (list b a)))

(defun reverse-mxx (w)
  (mapcar #'reverse-mx w))

> (reverse-mxx '(((a b) (c d) (e f)) ((a b) (c d) (e f))))
(((B A) (D C) (F E)) ((B A) (D C) (F E)))
```

Решение 4.64.8

```
(defun reverse-mx (w)
  (loop for (a b) in w collect `(,b ,a)))

(defun reverse-mxx (w)
  (mapcar #'reverse-mx w))

> (reverse-mxx '(((1 2) (3 4) (5 6)) ((1 2) (3 4) (5 6))))
(((2 1) (4 3) (6 5)) ((2 1) (4 3) (6 5)))
```

Решение 4.64.9

```
(defun reverse-mxx (w)
  (loop for ((a b) (c d) (e f)) in w
        collect `((,b ,a) (, d ,c) (,f ,e))))

> (reverse-mxx '(((1 2) (3 4) (5 6)) ((1 2) (3 4) (5 6))))
(((2 1) (4 3) (6 5)) ((2 1) (4 3) (6 5)))
```

Решение 4.64.10

```
(defun reverse-mxx (w)
  (mapcar #'(lambda (a) (mapcar #'reverse a)) w))

> (reverse-mxx '(((1 2) (3 4) (5 6)) ((1 2) (3 4) (5 6))))
(((2 1) (4 3) (6 5)) ((2 1) (4 3) (6 5)))
```

Задача 4.65 annex.lisp

Определить функцию для преобразования списка по принципу : (a b c) -> (((a) b) c).

Решение 4.65.1

```
(defun annex (w)
  (reduce #'list (cons (list (car w)) (cdr w))))

> (annex '(a b c d e))
(((((A) B) C) D) E)
```

Решение 4.65.2

```
(defun annex (w &optional acc)
  (if (null w)
      (car acc)
      (annex (cdr w) (list (reverse (cons (car w) acc))))))

> (annex '(a b c d e))
(((((A) B) C) D) E)
```

Решение 4.65.3

```
(defun annex (w &optional acc)
  (if w
      (annex (cdr w) (list (reverse (cons (car w) acc))))
      (car acc)))

> (annex '(a b c d e))
(((((A) B) C) D) E)
```

Решение 4.65.4

```
(defun annex (w &optional acc)
  (cond ((null w) (car acc))
        ((annex (cdr w) (list (reverse (cons (car w) acc)))))))

> (annex '(a b c d e))
(((((A) B) C) D) E)
```

Решение 4.65.5

```
(defun annex (w)
  (let ((r (list (car w))))
    (dolist (i (cdr w) r)
      (setq r (cons r (list i))))))

> (annex '(a b c d e))
(((((A) B) C) D) E)
```

Задача 4.66 remove-lists.lisp

Определить макрос, который удаляет списки из списка.

Решение 4.66.1

```
(defmacro remove-lists (w)
  `(cond ((null ,w) nil)
        ((atom (car ,w)) (cons (car ,w) (remove-lists (cdr ,w))))
        ((remove-lists (cdr ,w)))))

> (remove-lists '(1 (2)))
(1)
```

Решение 4.66.2

```
(defmacro remove-lists (w)
  `(remove-if #'listp ,w))

> (remove-lists '(1 (2)))
(1)
```

Решение 4.66.3

```
(defun remove-lists (w)
  `(loop for a in ,w when (atom a) collect a))

> (remove-lists '(1 (2)))
(LOOP FOR A IN (1 (2)) WHEN (ATOM A) COLLECT A)

(defmacro remove-lists (w)
  `(loop for a in ,w when (atom a) collect a))

> (remove-lists '(1 (2)))
(1)
```

Задача 4.67 read&change.lisp

Есть список (x1 x2 x3 ... xn). Необходимо списку присвоить значения, которые вводятся с клавиатуры. Оканчивать ввод если либо список кончился, либо пользователь ввел nil.

Решение 4.67.1

```
(defun read&change (lst)
    (when lst
      (let ((e (read)))
        (if e (cons e (read&change (cdr lst))) lst))))

> (read&change '(a a a))
3
4
5
(3 4 5)
```

Задача 4.68 chess-board.lisp

Имеется шахматная доска произвольного размера, клетки которой по горизонтали пронумерованы буквами, а по вертикали - числами. Постройте список пар координат всех клеток доски, в соответствии с двумя заданными списками букв и чисел нумерации доски.

Решение 4.68.1

```
(defun row (a w)
  (when w (cons (list a (car w)) (row a (cdr w)))))

(defun chess-board (v w)
  (when v (nconc (row (car v) w) (chess-board (cdr v) w))))

> (chess-board '(a b c) '(1 2))
((A 1) (A 2) (B 1) (B 2) (C 1) (C 2))
```

Задача 4.69 high-landers.lisp

Из заданного списка получить список, элементами которого будут являться только атомы верхнего уровня.

Решение 4.69.1

```
(defun high-landers (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((atom a) (cons a (high-landers d)))
        ((high-landers d))))

> (high-landers '(2 (a e) 4 ((5) y) c))
(2 4 C)
```

Задача 4.70 matrix-column-average.lisp

Дана вещественная матрица. Определить средние арифметические значения элементов столбцов.

Решение 4.70.1

```
(defun matrix-column-average (w)
  (mapcar #'(lambda (a) (/ (reduce #'+ a) (length a)))
          (apply #'mapcar #'list w)))

> (matrix-column-average '((1 2.5 3) (4.5 5 6) (7 8 9.5)))
(4.1666665 5.1666665 6.1666665)
```

Задача 4.71 minus>a.lisp

Определить функцию, которая заменяет все отрицательные элементы списка на a.

Решение 4.71.1

```
(defun minus>a (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((minusp a) (cons 'a (minus>a d)))
        ((cons a (minus>a d)))))

> (minus>a '(-1 1))
(A 1)
```

Решение 4.71.2

```
(defun minus>a (x)
  (cond ((null x) nil)
        ((minusp (car x)) (cons 'a (minus>a (cdr x))))
        ((cons (car x) (minus>a (cdr x))))))

> (minus>a '(-1 1))
(A 1)
```

Решение 4.71.3

```
(defun minus>a (w a)
  (cond ((null w) nil)
        ((minusp (car w)) (cons a (minus>a (cdr w) a)))
        ((cons (car w) (minus>a (cdr w) a)))))

> (minus>a '(-1 1) 10)
(10 1)
```

Решение 4.71.4

```
(defun minus>a (w b &aux (a (car w)))
```

```
  (when w (cons (if (minusp a) b a) (minus>a (cdr w) b))))

> (minus>a '(-1 2 -3 4 -5 7) 0)
(0 2 0 4 0 7)
```

Решение 4.71.5

```
(defun minus>a (w a)
  (loop for e in w if (minusp e) collect a else collect e))

> (minus>a '(-1 1) 10)
(10 1)
```

Решение 4.71.6

```
(defun minus>a (w a)
  (mapcar #'(lambda (e) (if (minusp e) a e)) w))

> (minus>a '(-1 1) 10)
(10 1)
```

Решение 4.71.7

```
(defun minus>a (w)
  (substitute-if 'a #'minusp w))

> (minus>a '(-1 1))
(A 1)
```

Задача 4.72 xelms.lisp

Описать функцию, которая берет в качестве аргумента список атомов и выдает список всех атомов, встречающихся в списке вместе с частотой их появления.

Решение 4.72.1

```
(defun xelms(w)
  (labels
      ((xlm (w ac)
         (cond ((null w) (reverse ac))
               ((find (car w) ac :key #'car) (xlm (cdr w) ac))
               ((xlm (cdr w) (cons (list (car w) (count (car w) w))
                                   ac))))))
    (xlm w nil)))

> (xelms'(a b a b a c a))
((A 4) (B 2) (C 1))
> (xelms'(ni to ni to))
((NI 2) (TO 2))
```

Решение 225.2

```
(defun xlm (w ac)
  (cond ((null w) (reverse ac))
        ((find (car w) ac :key #'car) (xlm (cdr w) ac))
        ((xlm (cdr w) (cons (list (car w) (count (car w) w)) ac)))))

(defun xelms (w)
  (xlm w nil))

> (xelms '(a b a b a c a))
((A 4) (B 2) (C 1))
> (xelms '(ni to ni to))
((NI 2) (TO 2))
```

Задача 4.73 matrix.lisp

Найти сумму всех чисел, которые входят числовыми компонентами в те списки, которые, являясь элементами матрицы, лежат на ее главной диагонали. Упомянутые списки должны быть подсписками уже третьего уровня в исходном списке, которым представлена сама матрица (не учитывать чисел, которые просто находятся на диагонали и являются обычными элементами матрицы):

```
исходная
 (1 2 5) (1 2) 6
 5 6 (1 2)
 6 8 (3 1)
результат
 (1 2 5 (8)) (1 2) 6
 5 6 (1 2)
 6 8 (3 1 (4))
```

Решение 4.73.1

```
(defun matrix (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((and (listp a) (or (= n 0) (= n 4) (= n 8)))
         (cons (append a (list (list (reduce #'+ a))))
               (matrix d (1+ n))))
        ((cons a (matrix d (1+ n))))))

> (matrix '((1 2 5) (1 2) 6 5 6 (1 2) 6 8 (3 1)))
((1 2 5 (8)) (1 2) 6 5 6 (1 2) 6 8 (3 1 (4)))
```

Задача 4.74 x-doubles.lisp

Построить рекурсивную функцию средствами элементарного лиспа, которая принимает список и возвращает список входящих в него атомов без повторений.

Решение 4.74.1

```
(defun x-doubles (m)
  (cond ((null m) nil)
        ((member (car m) (cdr m)) (x-doubles (cdr m)))
        (t (cons (car m) (x-doubles (cdr m))))))

> (x-doubles '(1 2 3 1 2 3 1 2 3))
(1 2 3)
```

Решение 4.74.2 * сохраняя порядок элементов

```
(defun doubles (m acc)
  (cond ((null m) (reverse acc))
        ((member (car m) acc) (doubles (cdr m) acc))
        (t (doubles (cdr m) (cons (car m) acc)))))

(defun x-doubles (m)
  (doubles m nil))

> (x-doubles '(1 3 2 1 2))
(1 3 2)
```

Решение 4.74.3

```
(defun x-doubles (m &optional ac)
  (cond ((null m) (reverse ac))
        ((member (car m) ac) (x-doubles (cdr m) ac))
        ((x-doubles (cdr m) (cons (car m) ac)))))

> (x-doubles '(1 3 2 1 2))
(1 3 2)
```

Задача 4.75 zeros.lisp

Определить функцию, заменяющую все отрицательные элементы списка значением 0.

Решение 4.75.1

```
(defun zeros (w)
  (cond ((null w) nil)
        ((< (car w) 0) (cons 0 (zeros (cdr w))))
        ((cons (car w) (zeros (cdr w))))))

> (zeros '(1 -5 6 -7 8 -9 -4))
(1 0 6 0 8 0 0)
```

Решение 4.75.2

```
(defun zeros (w)
  (substitute-if 0 #'minusp w))

> (zeros '(1 -5 6 -7 8 -9 -4))
```

```
(1 0 6 0 8 0 0)
```

Решение 4.75.3

```
(defun zeros (w)
  (mapcar #'(lambda (a) (if (minusp a) 0 a)) w))

> (zeros '(1 -5 6 -7 8 -9 -4))
(1 0 6 0 8 0 0)
```

Решение 4.75.4

```
(defun zeros (w)
  (loop for i in w
        if (minusp i) collect 0
        else collect i))

> (zeros '(1 -5 6 -7 8 -9 -4))
(1 0 6 0 8 0 0)
```

Задача 4.76 grow-deep.lisp

Дан список чисел произвольного уровня вложенности. Увеличить каждый элемент на единицу.

Решение 4.76.1

```
(defun grow-deep (w)
  (cond ((null w) nil)
        ((atom (car w)) (cons (1+ (car w)) (grow-deep (cdr w))))
        ((cons (grow-deep (car w)) (grow-deep (cdr w))))))

> (grow-deep '((1 (2 3)) ((4 (5) 6)) ((7 8) 9)))
((2 (3 4)) ((5 (6) 7)) ((8 9) 10))
```

Решение 4.76.2

```
(defun grow-deep (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((atom a) (cons (1+ a) (grow-deep d)))
        ((cons (grow-deep a) (grow-deep d)))))

> (grow-deep '((1 (2 3)) ((4 (5) 6)) ((7 8) 9)))
((2 (3 4)) ((5 (6) 7)) ((8 9) 10))
```

Решение 4.76.3

```
(defun grow-deep (x &aux r)
  (reverse (dolist (i x r)
             (cond ((numberp i) (setq r (cons (+ i 1) r)))
                   ((listp i) (setq r (cons (grow-deep i) r)))
                   ((setq r (cons (list i) r)))))))
```

```
> (grow-deep '((1 2 (3 4) (((5 6))) -1)))
((2 3 (4 5) (((6 7))) 0))
```

Решение 4.76.4

```
(defun grow-deep (w &aux (a (car w)) (d (cdr w)))
  (when w  (cons (if (atom a) (1+ a) (grow-deep a)) (grow-deep d))))

> (grow-deep '((1 2 (3 4) (((5 6))) -1)))
((2 3 (4 5) (((6 7))) 0))
```

Решение 4.76.5

```
(defun grow-deep (w &aux (a (car w)))
  (when w (cons (if (atom a) (1+ a) (grow-deep a))
                (grow-deep (cdr w)))))

> (grow-deep '((1 2 (3 4) (((5 6))) -1)))
((2 3 (4 5) (((6 7))) 0))
```

Решение 4.76.6

```
(defun grow-deep (w)
  (loop for a in w collect (if (atom a) (1+ a) (grow-deep a))))

> (grow-deep '(1 (2) ((3) 4) 5))
(2 (3) ((4) 5) 6)
```

Решение 4.76.7

```
(defun grow-deep (w)
  (loop for a in w collect
        (if (atom a) (if (numberp a) (1+ a) a) (grow-deep a))))

> (grow-deep '(1 (2 nil) ((3) nil 4) 5))
(2 (3 NIL) ((4) NIL 5) 6)
```

Задача 4.77 up-words.lisp

Дан текст. Сделать заглавной каждую букву каждого слова. Текст рекомендуется представлять списком списков: каждое предложение - список слов, весь текст - список предложений.

Решение 4.77.1

```
(defun up-word (w)
  (mapcar #'string-upcase w))

(defun up-words (w)
  (mapcar #'up-words w))
```

```
> (up-words '(("abc" "abc") ("abc" "abc")))
(("ABC" "ABC") ("ABC" "ABC"))
```

Задача 4.78 drop-dups.lisp

Удалить дубликаты из многоуровневого списка.

Решение 4.78.1

```
(defun drop-dups (w)
  (cond (w (let ((a (car w)) (d (cdr w)))
             (cond ((listp a) (cons (drop-dups a) (drop-dups d)))
                   ((member a d) (drop-dups d))
                   ((cons a (drop-dups d))))))))

> (drop-dups '(1 2 (2 2 3) 3))
(1 2 (2 3) 3)
```

Решение 4.78.2

```
(defun drop-dups (w)
  (cond (w ((lambda (a d)
              (cond ((listp a) (cons (drop-dups a) (drop-dups d)))
                    ((member a d) (drop-dups d))
                    ((cons a (drop-dups d)))))
            (car w) (cdr w)))))

> (drop-dups '(1 2 (2 2 3) 3))
(1 2 (2 3) 3)
```

Решение 4.78.3

```
(defun drop-dups (w &aux (a (car w)) (d (cdr w)))
  (when w (cond ((listp a) (cons (drop-dups a) (drop-dups d)))
                ((member a d) (drop-dups d))
                ((cons a (drop-dups d))))))

> (drop-dups '(1 2 (2 2 3) 3))
(1 2 (2 3) 3)
```

Решение 4.78.4

```
(defun drop-dups (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((listp a) (cons (drop-dups a) (drop-dups d)))
        ((member a d) (drop-dups d))
        ((cons a (drop-dups d)))))

> (drop-dups '(1 2 (2 2 3) 3))
(1 2 (2 3) 3)
```

Решение 4.78.5*

```
* Попутно превратить список в одноуровневый.

(defun drop-dups (w &optional acc)
  (cond ((null w) acc)
        ((listp w) (drop-dups (car w) (drop-dups (cdr w) acc)))
        ((member w acc) acc)
        ((cons w acc))))

> (drop-dups '(1 2 (2 2 3) 3 (3 4 4) 5))
(1 2 3 4 5)
```

Задача 4.79 exponent.lisp

По списку, состоящему из атомов-чисел, составить список значений exp(ai), где ai - i-й элемент списка. Значения exp(ai) большие (ai+2) не учитывать.

Решение 4.79.1

```
(defun exponent (w)
  (when w ((lambda (a d)
             (if (numberp a)
                 (if (< (exp a) (+ a 2))
                     (cons (exp a) (exponent d))
                     (exponent d))
                 (cons nil (exponent d))))
           (car w) (cdr w))))

> (exponent '(0.5 b 1 2 3 (10 20) 4))
(1.6487212 NIL 2.7182817 NIL)
```

Решение 4.79.2

```
(defun exponent (w)
  (cond (w ((lambda (a d)
              (cond ((numberp a)
                     (cond ((< (exp a) (+ a 2))
                            (cons (exp a) (exponent d)))
                           ((exponent d))))
                    ((cons nil (exponent d)))))
            (car w) (cdr w)))))

> (exponent '(0.5 b 1 2 3 (10 20) 4))
(1.6487212 NIL 2.7182817 NIL)
```

Задача 4.80 cap-words.lisp

Написать программу обработки текста естественного языка с использованием функционалов. Текст рекомендуется представлять списком списков: каждое предложение - список слов, весь текст - список предложений. Задание - сделать так, чтобы каждое слово начиналось с заглавной буквы.

```
Решение 4.80.1

(defun cap-words (text)
  (mapcar #'(lambda (sentence)
              (cons (string-capitalize (car sentence))
                    (cdr sentence)))
          text))

> (cap-words  '(("aa" "aa") ("aa" "aa")))
(("Aa" "aa") ("Aa" "aa"))

Решение 4.80.2

(defun cap-word (w)
  (mapcar #'string-capitalize w))

(defun cap-words (w)
  (mapcar #'cap-word w))

> (cap-words '(("abc" "abc") ("abc" "abc")))
(("Abc" "Abc") ("Abc" "Abc"))

Решение 4.80.3

(defun cap-words (w)
  (mapcar #'(lambda (v) (mapcar #'string-capitalize v)) w))

> (cap-sents '(("abc" "abc") ("abc" "abc")))
(("Abc" "Abc") ("Abc" "Abc"))
```

Задача 4.81 sum-pair.lisp

```
В числовом списке сложить каждые два соседних элемента, пример работы:
(f '(2 3 4 5 6 7)) > ((5) (9) (13))

Решение 4.81.1

(defun sum-pair (w)
  (cond ((null w) nil)
        ((cons (list (+ (car w) (cadr w))) (sum-pair (cddr w))))))

> (sum-pair '(2 3 4 5 6 7))
((5) (9) (13))

Решение 4.81.2

(defun sum-pair (w)
  (if (null w)
      nil
      (cons (list (+ (car w) (cadr w))) (sum-pair (cddr w)))))
```

```
> (sum-pair '(2 3 4 5 6 7))
((5) (9) (13))
```

Решение 4.81.3

```
(defun sum-pair (w)
  (when w (cons (list (+ (car w) (cadr w))) (sum-pair (cddr w)))))

> (sum-pair '(2 3 4 5 6 7))
((5) (9) (13))
```

Решение 4.81.4

```
(defun sum-pair (w)
  (loop for e on w by #'cddr collect (list (+ (car e) (cadr e)))))

> (sum-pair '(2 3 4 5 6 7))
((5) (9) (13))
```

Решение 4.81.5

В том числе для списков с нечетным числом элементов:

```
(defun sum-pair (w)
  (if (cdr w)
      (cons (list (+ (car w) (cadr w))) (sum-pair (cddr w)))
      (list w)))

> (sum-pair '(2 3 4 5 6 7 8))
((5) (9) (13) (8))
```

Задача 4.82 drop-pluspp.lisp

Определить функцию, которая из заданного списка удаляет все элементы являющиеся положительным числами. Список может быть многоуровневым и содержать не только числа.

Решение 4.82.1

```
(defun drop-pluspp (w)
  (cond ((null w) nil)
        ((consp (car w))
         (cons (drop-pluspp (car w)) (drop-pluspp (cdr w))))
        ((and (numberp (car w)) (plusp (car w)))
         (drop-pluspp (cdr w)))
        ((cons (car w) (drop-pluspp (cdr w))))))

> (drop-pluspp '(a 1 (-1 0) (0 (1))))
(A (-1 0) (0 NIL))
```

Решение 4.82.1

```
defun drop-pluspp (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((consp a) (cons (drop-pluspp a) (drop-pluspp d)))
        ((and (numberp a) (plusp a)) (drop-pluspp d))
        ((cons a (drop-pluspp d)))))

> (drop-pluspp '(a 1 (-1 0) (0 (1))))
(A (-1 0) (0 NIL))
```

Задача 4.83 sum-pairs.lisp

Написать функцию, которая принимает на вход последовательность, а возвращает последовательность с попарными суммами, т.е. (1 2 3 4 5) > (3 5 7 9).

Решение 4.83.1

```
(defun sum-pairs (w)
  (when (cdr w) (cons (+ (car w) (cadr w)) (sum-pairs (cdr w)))))

> (sum-pairs '(1 2 3 4 5))
(3 5 7 9)
```

Задача 4.84 remove-eventh.lisp

Из заданного списка удалить каждый второй элемент, результирующий список переписать в обратном порядке.

Решение 4.84.1

```
defun remove-eventh  (w &optional ac)
  (if (null w) ac (remove-eventh (cddr w) (cons (car w) ac))))

> (remove-eventh '(1 2 3 4 5))
(5 3 1)
```

Задача 4.85 elm-freq.lisp

Для списка произвольного уровня вложенности выполнить анализ частоты появления каждого символа. Все встречающиеся символы и их число повторов объединить в списки, из которых сформировать список результатов.

Решение 4.85.1

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun elm-freq (w)
  (mapcar #'(lambda (a) (list a (count a (flat w))))
```

```
          (delete-duplicates (flat w))))

> (elm-freq '(a ((a) (a (a)) b) (b (b) c) c))
((A 4) (B 3) (C 2))
```

Решение 4.85.2

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun xelm (w &optional ac &aux (a (car w)))
  (cond ((null w) (reverse ac))
        ((find a ac :key #'car) (xelm (cdr w) ac))
        ((xelm (cdr w) (cons (list a (count a w)) ac)))))

(defun elm-freq (w)
  (xelm (flat w)))

> (elm-freq '(a ((a) (a (a)) b) (b (b) c) c))
((A 4) (B 3) (C 2))
```

Решение 4.85.3

```
(defun flat (w acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun xelm (w ac)
  (cond ((null w) (reverse ac))
        ((find (car w) ac :key #'car) (xelm (cdr w) ac))
        ((xelm (cdr w) (cons (list (car w) (count (car w) w)) ac)))))

(defun elm-freq (w)
  (xelm (flat w nil) nil))

> (elm-freq '(a ((a) (a (a)) b) (b (b) c) c))
((A 4) (B 3) (C 2))
```

Решение 4.85.4

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun xelm (w &aux (v (remove-duplicates w)))
  (loop for a in v collect (list a (count a w))))

(defun elm-freq (w)
  (xelm (flat w)))
```

```
> (elm-freq '(a ((a) (a (a)) b) (b (b) c) c))
((A 4) (B 3) (C 2))
```

Задача 4.86 arrange--num-uscore-char.lisp

Создать рекурсивную функцию для обработки списка, чтоб каждый атом списка состоял из упорядоченных цифр, знака подчеркивания и букв.

Решение 4.86.1

```
(defun arrange-num-uscore-char (w)
  (when w (cons (intern (sort (string (car w)) #'string<))
                (arrange-num-ucore-char (cdr w)))))

> (arrange-num-uscore-char '("b_a72" "dc5_1"))
(|27_ab| |15_cd|)
```

Задача 4.87 double.lisp

Дан список, элементы которого — натуральные числа. Умножить каждый элемент на 2.

Решение 4.87.1

```
(defun double (w)
  (when w (cons (* (car w) 2) (double (cdr w)))))

> (double '(1 2))
(2 4)
```

Решение 4.87.2

```
(defun double (w)
  (loop for a in w collect (* a 2)))

> (double '(1 2))
(2 4)
```

Задача 4.88 rearrange.lisp

Задана последовательность из трех чисел. Если последовательность упорядочена по возрастанию, то упорядочить ее по убыванию и наоборот. разработать функцию двумя способами (if.....и cond....). Входные значения передаются в качестве списка и выходные тоже.

Решение 4.88.1

```
(defun rearrange (w)
  (if (or (apply #'< w) (apply #'> w)) (reverse w) (w)))
```

```
> (rearrange '(1 2 3))
(3 2 1)
```

Решение 4.88.2

```
(defun rearrange (w)
  (cond ((or (apply #'< w) (apply #'> w)) (reverse w)) (w)))

> (rearrange '(1 2 3))
(3 2 1)
```

Задача 4.89 drop-plusp.lisp

Определите функцию (f w), результатом которой является список, получающийся после удаления на всех уровнях всех положительных элементов списка чисел w.

Решение 4.89.1

```
(defun drop-plusp (w)
  (cond ((null w) nil)
        ((consp (car w))
         (cons (drop-plusp (car w)) (drop-plusp (cdr w))))
        ((plusp (car w)) (drop-plusp (cdr w)))
        ((cons (car w) (drop-plusp (cdr w))))))

> (drop-plusp '(1 (-1)))
((-1))
```

Решение 4.89.2

```
(defun drop-plusp (w &aux (a (car w)))
  (cond ((null w) nil)
        ((consp a) (cons (drop-plusp a) (drop-plusp (cdr w))))
        ((plusp a) (drop-plusp (cdr w)))
        ((cons a (drop-plusp (cdr w))))))

> (drop-plusp '(1 (-1)))
((-1))
```

Решение 4.89.3

```
(defun drop-plusp (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((consp a) (cons (drop-plusp a) (drop-plusp d)))
        ((plusp a) (drop-plusp d))
        ((cons a (drop-plusp d)))))

> (drop-plusp '(1 (-1)))
((-1))
```

Решение 4.89.4

```
(defun drop-plusp (w)
  (lambda (a d)
    (cond ((null w) nil)
          ((consp a) (cons (drop-plusp a) (drop-plusp d)))
          ((plusp a) (drop-plusp d))
          ((cons a (drop-plusp d)))))
  (car w) (cdr w))

> (drop-plusp '(1 (-1)))
((-1))
```

Задача 4.90 rotate.lisp

Реализовать функцию, меняющую местами первый и последний элементы исходного списка

Решение 4.90.1

```
(defun rotate (w)
  (rotatef (car w) (nth (1- (length w)) w)) w)

> (rotate '(a b c d e f))
(F B C D E A)
```

Решение 4.90.2

```
(defun rotate (w)
  (nconc (last w) (butlast (cdr w)) (cons (car w) nil)))

> (rotate '(a b c d e f))
(F B C D E A)
```

Задача 4.91 wvd.lisp

Реализовать: нахождение разности списков R=S1/S2 (элементы всех уровней). Форма представление результата: результатом есть список, содержащий исходный и результирующие.

Решение 4.91.1

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun wvd (w v)
  (list w v (set-difference (flat w) (flat v))))

> (wvd '(a b (c) d) '((c d) e))
((A B (C) D) ((C D) E) (A B))
```

Задача 4.92 cut-1.lisp

Дан список, элементы которого - натуральные числа. Вычесть из каждого элемента списка 1.

Решение 4.92.1

```
(defun cut-1 (w)
  (cond ((null w) nil)
        ((cons (1- (car w)) (cut-1 (cdr w))))))

> (cut-1 '(1 2))
(0 1)
```

Решение 4.92.2

```
(defun cut-1 (w)
  (if w (cons (1- (car w)) (cut-1 (cdr w))) nil))

> (cut-1 '(1 2))
(0 1)
```

Решение 4.92.3

```
(defun cut-1 (w)
  (when w (cons (1- (car w)) (cut-1 (cdr w)))))

> (cut-1 '(1 2))
(0 1)
```

Решение 4.92.4

```
(defun cut-1 (w)
  (mapcar #'1- w))

> (cut-1 '(1 2))
(0 1)
```

Задача 4.93 plusp10.lisp

Дан список, элементы которого натуральные числа. Заменить их на 1, если исходное число положительно, и на ноль в противном случае.

Решение 4.93.1

```
(defun plusp10 (w)
  (cond ((null w) nil)
        ((plusp (car w)) (cons 1 (plusp10 (cdr w))))
        ((cons 0 (plusp10 (cdr w))))))

> (plusp10 '(-1 1))
(0 1)
```

Задача 4.94 listt.lisp

Дан список с произвольными элементами. Заменить каждый элемент на T, если элемент - список. И на NIL в противном случае.

Решение 4.94.1

```
(defun listt (w)
  (cond ((null w) nil)
        ((cons (listp (car w)) (listt (cdr w))))))

> (listt '(a (a b)))
(NIL T)
```

Решение 4.94.2

```
(defun listt (w)
  (when w (cons (listp (car w)) (listt (cdr w)))))

> (listt '(a (a b)))
(NIL T)
```

Задача 4.95 even-plusp.lisp

Дан список, описывающий вызов арифметической функции. Написать функцию, что в случае четности результата вычисления списка делает его проверку на позитивность, а в противном случае выдает сам список. Вычисление списка делать с помощью встроенной функции eval.

Решение 4.95.1

```
(defun even-plusp (w)
  (or (and (evenp (eval w)) (plusp (eval w))) w))

> (even-plusp '(+ 1 1))
T
> (even-plusp '(+ 1 2))
(+ 1 2)
```

Задача 4.96 compress-word.lisp

Построить программу «сжатия», назначение которой - преобразование английских слов в их "звуковой" код. Этот процесс предусматривает "сжатие" примерно одинаково звучащих слов в одинаковый их код - своего рода, аббревиатуру этих слов. Слова "сжимаются" в соответствии со следующими правилами:
- первая буква слова сохраняется;
- все последующие за ней гласные, а также буквы "h", "w" и "y" удаляются;
- сдвоенные буквы заменяются одиночными;
- закодированное слово состоит не более чем из четырех букв, остальные

буквы удаляются.
Напишите функцию (f w), которая выдает результат сжатия слова, представленного в виде списка букв w. Примеры: (f '(b a r r i n g t o n)) = (b r n g) и (f '(1 1 e w e 1 1 y n)) =: (1 n).

Решение 4.96.1

```
(defun compress-word (w &optional ac &aux (a (car w)))
  (cond ((or (null w)(= (length ac) 4)) (reverse ac))
        ((or (member a '(a e i o u y h w)) (eq a (car ac)))
         (compress-word (cdr w) ac))
        ((compress-word (cdr w) (cons a ac)))))

> (compress-word '(b a r r i n g t o n))
(B R N G)
> (compress-word '(1 1 e w e 1 1 y n))
(1 N)
```

Решение 4.96.2

```
(defun compress-word (w)
  (compress (cdr w) (list (car w))))

(defun compress (w ac &aux (a (car w)))
  (cond ((or (null w)(= (length ac) 4)) (reverse ac))
        ((or (member a '(a e i o u y h w)) (eq a (car ac)))
         (compress (cdr w) ac))
        ((compress (cdr w) (cons a ac)))))

> (compress-word '(b a r r i n g t o n))
(B R N G)
> (compress-word '(a r r i n g t o n))
(A R N G)
```

Решение 4.96.3

```
(defun compress-word (w)
  (compress (cdr w) `(,(car w))))

(defun compress (w ac &aux (a (car w)))
  (if (or (null w)(= (length ac) 4))
      (reverse ac)
      (compress (cdr w)
                (if (or (member a '(a e i o u y h w))
                        (eq a (car ac)))
                    ac
                    (cons a ac)))))

> (compress-word '(b a r r i n g t o n))
(B R N G)
> (compress-word '(a r r i n g t o n))
(A R N G)
```

```
Решение 4.96.4* (для XLISP)

(defun compress-word (w)
  (compress w nil))

(defun compress (w ac)
  (cond ((or (null w)(= (length ac) 4)) (reverse ac))
        ((or (mmb (car w) '(a e i o u y h w)) (eql (car w) (car ac)))
         (compress (cdr w) ac))
        ((compress (cdr w) (cons (car w) ac)))))

(defun mmb (a w)
    (cond ((null w) nil)
    ((eql a (car w)) (cons (car w) (cdr w)))
    ((mmb a (cdr w)))))

> (compress-word '(b a r r i n g t o n))
(B R N G)
> (compress-word '(1 1 e w e 1 1 y n))
(1 N)
```

Задача 4.97 arrange.lisp

```
Отсортировать список вида ((A 3) (B 2) (C 5)) так чтобы получилось ((C
5) (A 3) (B 2)).

Решение 4.97.1

(defun arrange (w)
  (sort w #'(lambda (x y) (> (cadr x) (cadr y)))))

> (arrange '((A 3) (B 2) (C 5)))
((C 5) (A 3) (B 2))

Решение 4.97.2

(defun arrange (w)
  (sort w #'> :key #'second))

> (arrange '((A 3) (B 2) (C 5)))
((C 5) (A 3) (B 2))

Решение 4.97.3

(defun arrange (w)
  (sort w #'> :key #'cadr))

> (arrange '((A 3) (B 2) (C 5)))
((C 5) (A 3) (B 2))
```

Задача 4.98 arrange-vcnm.lisp

Заданный список произвольного уровня вложенности разбить на четыре списка: в первом – гласные, во втором – согласные, в третьем – цифры, в четвертом – все остальное. Первые три списка должны быть упорядочены и в них не должно быть повторов. Списки результатов объединить в один список списков.

Решение 4.98.1

```
(defun pose (w &optional v c n m &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (list (sort-vcn v #'char<)
                        (sort-vcn c #'char<)
                        (sort-vcn n #'<)  m))
        (t (case a
             ((#\a #\e #\i #\o #\y #\u)
              (pose d (push a v) c n m))
             ((#\b #\c #\d #\f #\g #\h #\j #\k #\l #\m #\n #\p
                   #\q #\r #\s #\t #\v #\w #\x #\z)
              (pose d v (push a c) n m))
             ((0 1 2 3 4 5 6 7 8 9)
              (pose d v c (push a n) m))
             (otherwise (pose d v c n (push a m)))))))

(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun sort-vcn (w p)
  (sort (remove-duplicates w) p))

(defun arrange-vcnm (w)
  (pose (flat w)))

> (arrange-vcnm '(3 (2 #\a) #\b (3) #\b (#\u 1) #\c 1 #\c (!)))
((#\a #\u) (#\b #\c) (1 2 3) (!))
```

Решение 4.98.2

```
(defun pose (w v c n m)
  (cond ((null w) (list (sort-smbs v)
                        (sort-smbs c)
                        (sort-vcn n '<)
                        m))
        ((member (car w) '(a e i o y u))
         (pose (cdr w) (cons (car w) v) c n m))
        ((member (car w) '(b c d f g h j k l m n p q r s t v w x z))
         (pose (cdr w) v (cons (car w) c) n m))
        ((member (car w) '(0 1 2 3 4 5 6 7 8 9))
         (pose (cdr w) v c (cons (car w) n) m))
        (t (pose (cdr w) v c n (cons (car w) m)))))
```

```
(defun sort-smbs (w)
  (sort (remove-dps w) 'string<))

(defun flat (w acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun sort-vcn (w p)
  (sort (remove-dps w) p))

(defun remove-dps (w)
  (cond ((null w) nil)
        ((member (car w) (cdr w)) (remove-dps (cdr w)))
        (t (cons (car w) (remove-dps (cdr w))))))

(defun arrange-vcnm (w)
  (pose (flat w nil) nil nil nil nil))

> (arrange-vcnm '(3 (2 a) b (3) b (u 1) c 1 c (!)))
((A U) (B C) (1 2 3) (!))
```

Решение 4.98.3

```
(defun pose (w &optional v c n m &aux (a (car w)) (d (cdr w)))
  (cond ((null w)
         (list (sort v #'string<) (sort c #'string<) (sort n #'<) m))
        ((member a '(a e i o y u))
         (pose d (pushnew a v) c n m))
        ((member a '(b c d f g h j k l m n p q r s t v w x z))
         (pose d v (pushnew a c) n m))
        ((member a '(0 1 2 3 4 5 6 7 8 9))
         (pose d v c (pushnew a n) m))
        (t (pose d v c n (push a m)))))

(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun arrange-vcnm (w)
  (pose (flat w)))

> (arrange-vcnm '(3 (2 a) b (3) b (u 1) c 1 c (!)))
((A U) (B C) (1 2 3) (!))
```

Решение 4.98.4

```
(defconstant vowels '(a e i o y u))
(defconstant consonants '(b c d f g h j k l m n p q r s t v w x z))
(defconstant digits '(0 1 2 3 4 5 6 7 8 9))
```

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun arrange-vcd (w)
  (list (loop for v in vowels when (member v w) collect v)
        (loop for c in consonants when (member c w) collect c)
        (loop for d in digits when (member d w) collect d)))

(defun arrange-vcdm (w &aux (u (flat w))(v (arrange-vcd u)))
  (nconc v (list (remove-if #'(lambda (a) (member a (flat v))) u))))

> (arrange-vcdm '(3 (2 a a a a) b (3) b (u 1) c 1 c (!)))
((A U) (B C) (1 2 3) (!))
```

Задача 4.99 decompose.lisp

Для списка произвольного уровня вложенности, состоящего из букв, цифр и строк, вычислить число объектов каждого вида и их процентное соотношение. Сам список превратить в список из объектов, в котором нет повторов.

Решение 4.99.1

```
(defun cns (w &optional ac (m 0) (c 0) (n 0) (s 0)
            &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (values
                   ac
                   (format nil "chrs ~a ~a%, nums ~a ~a%, strs ~a ~a%"
                           c (cnt c m)
                           n (cnt n m)
                           s (cnt s m))))
        ((numberp a) (if (member a d)
                         (cns d ac (1+ m) c (1+ n) c)
                         (cns d (push a ac) (1+ m) c (1+ n) s)))
        ((stringp a) (if (member a d :test #'equal)
                         (cns d ac (1+ m) c n (1+ s))
                         (cns d (push a ac) (1+ m) c n (1+ s))))
        (t (if (member a d)
               (cns d ac (1+ m) (1+ c) n s)
               (cns d (push a ac) (1+ m) (1+ c) n s)))))

(defun cnt (a m)
  (floor (float (/ (* a 100) m))))

(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun decompose (w)
```

```
  (cns (flat w)))

> (decompose '(1 (#\a) ("b" (1)) "b" (1)))
(1 "b" #\a)
"chrs 1 16%, nums 3 50%, strs 2 33%"
```

Решение 4.99.2 (muLISP)

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun pose (w &aux
             (a (count-if #'symbolp w))
             (b (count-if #'numberp w))
             (c (count-if #'stringp w))
             (m (+ a b c)))
  (values (remove-duplicates w)
          (format nil "smbs ~a ~a%, nums ~a ~a%, strs ~a ~a%"
                  a (cnt a m)
                  b (cnt b m)
                  c (cnt c m))))

(defun decompose (w)
  (pose (flat w)))

> (decompose '(A S D (E 4 R "mix") (Q W) ("string" M) 2 I M M 2 2 P))
(A S D E 4 R "mix" Q W "string" I M 2 P)
"smbs 12 66%, nums 4 22%, strs 2 11%"
```

Задача 4.100 translate.lisp

Создать программу "Переводчик", которая для заданного списка английских слов произвольного уровня вложенности будет формировать список соответствующих русских слов. Список результата не должен содержать вложений. Перевод осуществляется на основе заранее сформированного словаря (не менее 15 слов).

Решение 4.100.1

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun late (a w)
  (cond ((null w) nil)
        ((equal a (caar w)) (cons (cadar w) (late a (cdr w))))
        ((late a (cdr w)))))

(defun tran (w v)
```

```
  (cond ((null w) nil)
        ((nconc (late (car w) v) (tran (cdr w) v)))))

(defun translate (w v)
  (tran (flat w) v))

> (translate '(cakes (ale)) '((cakes pyrogy) (ale pyvo)))
(PYROGY PYVO)
```

Решение 4.100.2 (muLISP)

```
(setq slovar '((cakes pyrogy) (ale pivo) (phone telefon) (child rebe-
nok) (day den) (apple yabloko) (cat kot) (dog sobaka) (full polniy)))

(defun flat (w acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun late (a w)
  (cond ((null w) nil)
        ((eql a (car (car w)))
         (cons (car (cdr (car w))) (late a (cdr w))))
        ((late a (cdr w)))))

(defun tran (w v)
  (cond ((null w) nil)
        ((append (late (car w) v) (tran (cdr w) v)))))

(defun translate (w v)
  (tran (flat w nil) v))

> (translate '((apple) (phone) (day)) slovar)
(YABLOKO TELEFON DEN)
```

Задача 4.101 delete-first-atom.lisp

Определить функцию удаления первого атома в списке.

Решение 4.101.1

```
(defun delete-first-atom (w)
  (cond ((null w) nil)
        ((atom (car w)) (cdr w))
        ((cons (car w) (delete-first-atom (cdr w))))))

> (delete-first-atom '((a b) c (d) e))
((A B) (D) E)
```

Решение 4.101.2

```
(defun delete-first-atom (w)
```

```
  (loop for a in w and d on w
        if (consp a) collect a into as
        else return (nconc as (cdr d))))

> (delete-first-atom '((a b) c (d) e))
((A B) (D) E)

> (time (delete-first-atom (nconc (loop for i from 1 to 1000000 col-
lect '(a)) '(x (z)))))
Real time: 7.8451204 sec.
Run time: 7.81565 sec.
Space: 16004224 Bytes
GC: 12, GC time: 0.780005 sec.

Решение 4.101.3

(defun del-lead-atom (w)
  (delete-if #'atom w :count 1))

> (delete-first-atom '((a b) c (d) e))
((A B) (D) E)

> (time (del-lead-atom (nconc (loop for i from 1 to 1000000 collect
'(a)) '(x (z)))))
Real time: 7.7151184 sec.
Run time: 7.6908493 sec.
Space: 8129236 Bytes
GC: 8, GC time: 0.3276021 sec.
```

Задача 4.102 coin.lisp

Среди 12 монет есть одна фальшивая, отличающаяся от остальных по весу (неизвестно легче она или тяжелее). Необходимо тремя взвешиваниями определить фальшивую монету.

Решение 4.102.1 (алгоритм взвешиваний)

```
(defun coin (c1 c2 c3 c4 c5 c6 c7 c8 c9 c10 c11 c12)
  (cond ((= (+ c1 c2 c3 c4) (+ c5 c6 c7 c8))
         (cond ((= (+ c1 c2 c3) (+ c9 c10 c11))
                (if (< c1 c12) (rm "c12 H") (rm "c12 L")))
               ((> (+ c1 c2 c3) (+ c9 c10 c11))
                (cond ((= c9 c10) (rm "c11 L"))
                      ((< c9 c10) (rm "c9 L"))
                      ((rm "c10 L"))))
               ((cond ((= c9 c10) (rm"c11 H"))
                      ((< c9 c10) (rm "c9 H"))
                      ((rm "c10 H")))))))
        ((> (+ c1 c2 c3 c4) (+ c5 c6 c7 c8))
         (cond ((= (+ c1 c2 c5) (+ c3 c4 c12))
                (cond ((= c6 c7) (rm "c8 L"))
                      ((< c6 c7) (rm "c6 L"))
```

```
                      ((rm "c7 L"))))
               ((> (+ c1 c2 c5) (+ c3 c4 c12))
                (if (< c1 c2) (rm "c2 H") (rm "c1 H")))
               ((cond ((= c3 c4) (rm "c5 L"))
                      ((< c3 c4) (rm "c4 H"))
                      ((rm "c3 H"))))))
        ((cond ((= (+ c1 c2 c5) (+ c3 c4 c12))
                (cond ((= c6 c7) (rm "c8 H"))
                      ((< c6 c7) (rm "c6 H"))
                      ((rm "c7 H"))))
                ((< (+ c1 c2 c5) (+ c3 c4 c12))
                 (if (< c1 c2) (rm "c1 L") (rm "c2 L")))
                ((cond ((= c3 c4) (rm "c5 H"))
                       ((< c3 c4) (rm "c3 L"))
                       ((rm "c4 L")))))))))

(defun rm (s)
  (format t "~a" s))

> (coin 2 2 1.5 2 2 2 2 2 2 2 2 2)
c3 L
NIL
```

Задача 4.103 intersection-nums.lisp

Написать функцию, которая из списка цифр и символов выбирает числа, принадлежащие другому списку.

Решение 4.103.1

```
(defun inter-nums (w v)
  (cond ((null w) nil)
        ((and (numberp (car w)) (member (car w) v))
         (cons (car w) (inter-nums (cdr w) v)))
        ((inter-nums (cdr w) v))))

> (inter-nums '(a b 1 2 3) '(b c 2 3 4))
(2 3)
```

Решение 4.103.2

```
(defun ns (w v &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((and (realp a) (member a v)) (cons a (inter-nums d v)))
        ((inter-nums d v))))

> (inter-nums '(a 0 1) '(a 1 2))
(1)
```

Задача 4.104 remove-names.lisp

Написать программу, реализующую тел.справочник. в нем содержится информация о каждом абоненте: Ф,И,О, телефон, улица, номер дома, квартира. Есть список фактов:
("Ivanov" "Slava" 11111).
("Petrov" "Igor" 22222).
("Sokolov" "Fedor" 3333,).
("Sidorov" "Kostya" 4444).
("Vasijev" "Dmitrij" 5555).
("Petrov" "Ivan" 6666).
("Ivanov" "Fedor" 7777).
Как реализовать функцию, которая бы убирала все одинаковые фамилии?

Решение 4.104.1

```
(defun remove-names (w)
  (remove-duplicates w :key #'car :test #'string=))
```

```
> (rd '(("Ivanov" "Slava" 11111)("Petrov" "Igor" 22222) ("Sokolov"
"Fedor" 3333) ("Sidorov" "Kostya" 4444) ("Vasijev" "Dmitrij" 5555)
("Petrov" "Ivan" 6666) ("Ivanov" "Fedor" 7777)))
(("Sokolov" "Fedor" 3333) ("Sidorov" "Kostya" 4444) ("Vasijev"
"Dmitrij" 5555) ("Petrov" "Ivan" 6666) ("Ivanov" "Fedor" 7777))
```

Решение 4.104.2

```
(defun remove-names (w)
  (cond ((null w) nil)
        ((member (caar w) (cdr w) :key #'car :test #'string=)
         (remove-names (cdr w)))
        ((cons (car w) (remove-names (cdr w))))))
```

```
> (remove-names '(("Ivanov" "Slava" 11111)("Petrov" "Igor" 22222)
("Sokolov" "Fedor" 3333) ("Sidorov" "Kostya" 4444) ("Vasijev" "Dmi-
trij" 5555) ("Petrov" "Ivan" 6666) ("Ivanov" "Fedor" 7777)))
(("Sokolov" "Fedor" 3333) ("Sidorov" "Kostya" 4444) ("Vasijev"
"Dmitrij" 5555) ("Petrov" "Ivan" 6666) ("Ivanov" "Fedor" 7777))
```

Решение 4.104.3

```
(defun remove-names (w)
  (cond ((null w) nil)
        ((find (caar w) (cdr w) :key #'car :test #'string=)
         (remove-names (cdr w)))
        ((cons (car w) (remove-names (cdr w))))))
```

```
> (rd '(("Ivanov" "Slava" 11111)("Petrov" "Igor" 22222) ("Sokolov"
"Fedor" 3333) ("Sidorov" "Kostya" 4444) ("Vasijev" "Dmitrij" 5555)
("Petrov" "Ivan" 6666) ("Ivanov" "Fedor" 7777)))
(("Sokolov" "Fedor" 3333) ("Sidorov" "Kostya" 4444) ("Vasijev"
"Dmitrij" 5555) ("Petrov" "Ivan" 6666) ("Ivanov" "Fedor" 7777))
```

Задача 4.105 rooms&price.lisp

Есть следующий набор данных:
```
 ("Grand" "5" 9 420)
 ("Nobis" "5" 4 318)
 ("Berns" "4" 8 333)
 ("Diplomat" "4" 30, 320)
 ("Attache" "3" 8 164)
 ("General" "3" 7 207)
 ("Vanadis" "2" 5 99)
 ("Dialog" "2"  5  78),
```
который представляет данные об отелях, где на третьем места стоит количество свободных номеров, а на четвертом - цена за номер. Необходимо посчитать общее количество свободных номеров (сумма по третьей позиции), и среднюю цену за номер (среднее значение по четвертой).

Решение 4.105.1

```
(defun rooms&price (w)
  (list
   (reduce #'+ (mapcar #'caddr w))
   (float (/ (reduce #'+ (mapcar #'cadddr w)) (length w)))))

> (rooms&price '(("Grand" "5" 9 420) ("Nobis" "5" 4 318) ("Berns" "4"
8 333) ("Diplomat" "4" 30 320) ("Attache" "3" 8 164) ("General" "3" 7
207) ("Vanadis" "2" 5 99) ("Dialog" "2" 5 78)))
(76 242.375)
```

Задача 4.106 edges-up.lisp

Написать функцию, которая для списка-аргумента формирует список-результат по правилу: если первый и последний элементы списка-аргумента – четные положительные целые числа, то включить в список-результат первым элементом – квадрат последнего элемента исходного списка, вторым – четвертую степень первого; в противном случае сформировать список из первого и последнего элементов.

Решение 4.106.1

```
(defun edges-up (a)
  (if (and (plus-even (car a))
           (plus-even (car (reverse a))))
      (list (expt (car (reverse a)) 2) (expt (car a) 4))
      (list (car a) (car (reverse a)))))

(defun plus-even (x)
  (and (realp x) (plusp x) (evenp x)))

> (edges-up '(28 4 5 6 7 82))
(6724 614656)
> (edges-up '(20 4 5 6 7 89))
(20 89)
```

```
Решение 4.106.2

(defun edges-up (list)
  ((lambda (edges)
     (if (equal (mapcar #'plus-even edges) '(T T))
         (reverse (mapcar #'expt edges '(4 2)))
         edges))
   (cons (car list) (last list))))

(defun plus-even (x)
  (and (realp x) (plusp x) (evenp x)))

> (edges-up '(20 4 5 6 7 82))
(6724 160000)
> (edges-up '(20 4 5 6 7 81))
(20 81)
> (edges-up '(20 4 5 6 7 g))
(20 G)

Решение 4.106.3

(defun edges-up (w &aux (edges (cons (car w) (last w))))
  (if (every #'(lambda (x) (and (realp x) (plusp x) (evenp x)))
             edges)
      (nreverse (mapcar #'expt edges '(4 2)))
      edges))

> (edges-up '(20 4 5 6 7 82))
(6724 160000)
> (edges-up '(20 4 5 6 7 81))
(20 81)
> (edges-up '(20 4 5 6 7 g))
(20 G)
```

Задача 4.107 sort-numbers.lisp

```
Упорядочить цифры в каждом атоме списка a53r2 > a23r5.

Решение 4.107.1

(defun split-string (s)
  (map 'list #'string s))

(defun wsmb-to-wwstrs (w)
  (mapcar #'split-string (mapcar #'string w)))

(defun extract-nums (w)
  (delete-if-not
   #'(lambda (a) (parse-integer a :junk-allowed t)) w))

(defun sort-nums (w)
```

```
  (mapcar
   #'write-to-string(sort (mapcar
                           #'parse-integer w) #'<)))

(defun arrange (w)
  (mapcar
   #'(lambda (a)
       (xmint a (sort-nums (extract-nums (copy-list a)))))
   w))

(defun xmint (w n)
  (cond ((null w) nil)
        ((parse-integer (car w) :junk-allowed t)
         (cons (car n) (xmint (cdr w) (cdr n))))
        ((cons (car w) (xmint (cdr w) n)))))

(defun join-strs (w)
  (cond ((null w) nil)
        ((concatenate 'string
                      (car w)
                      (join-strs (cdr w))))))

(defun sort-numbers (w)
   (mapcar
    #'intern
    (mapcar
     #'join-strs (arrange (wsmb-to-wwstrs w)))))

> (sort-numbers '(a53r2 v8n74))
(A23R5 V4N78)
```

Задача 4.108 compress-words.lisp

Дан текст. В каждом слове каждого предложения для повторяющихся литер произвести следующую замену: повторные вхождения литер удалить, к первому вхождению литеры приписать число вхождений литеры в слово. Пример: ((aaabb ccccddd)(eeefggg hhkl)) преобразуется в ((a3b2 c4d3)(e3fg3 h2kl)).

Решение 4.108.1

```
(defun ss (w)
  (mapcar #'sw (mapcar #'string w)))

(defun sw (s)
 (map 'list #'string s))

(defun cn (w &optional (n 1) (acc ""))
  (cond ((null w) acc)
        ((equal (car w) (cadr w)) (cn  (cdr w) (1+ n) acc))
        ((cn (cdr w) 1 (concatenate
                        'string acc (car w)
```

```
                        (if (> n 1) (write-to-string n) "")))))))

(defun squeeze (w)
  (mapcar #'cn (ss w)))

(defun compress-words (w)
  (mapcar #'squeeze w))

> (compress-words '((aaabb ccccddd)(eeefggg hhkl)))
(("A3B2" "C4D3") ("E3FG3" "H2KL"))
```

Решение 4.108.2

```
(defun ss (w)
  (mapcar #'sw (mapcar #'string w)))

(defun sw (s)
 (map 'list #'string s))

(defun cn (w &optional (n 1) (acc ""))
  (cond ((null w) acc)
        ((equal (car w) (cadr w)) (cn  (cdr w) (1+ n) acc))
        ((cn (cdr w) 1 (concatenate
                        'string acc (car w)
                        (if (> n 1) (write-to-string n) "")))))))

(defun squeeze (w)
  (mapcar #'cn (ss w)))

(defun str-sym (w)
  (mapcar #'(lambda (a) (read-from-string a)) w))

(defun compress-words (w)
  (mapcar #'str-sym (mapcar #'squeeze w)))

> (compress-words '((aaabb ccccddd)(eeefggg hhkl)))
((A3B2 C4D3) (E3FG3 H2KL))
```

Решение 4.108.3 (MuLisp)

```
(defun compress-words (w)
  (mapcar #'(lambda (x) (cn (unpack x) 1 "")) w))

(defun cn (w n ac)
  (cond ((null w) ac)
        ((equal (car w) (cadr w)) (cn  (cdr w) (1+ n) ac))
        ((cn (cdr w)
             1
             (pack (list ac (car w) (if (> n 1) n "")))))))

> (compress-words '((aaabb ccccddd)(eeefggg hhkl)))
((A3B2 C4D3) (E3FG3 H2KL))
```

Задача 4.109 odd>0-even>1.lisp

Дан список, элементы которого натуральные числа. Поменять их на число 1, если исходное число - четное, и на 0 в противном случае.

Решение 4.109.1

```
(defun odd>0-even>1 (w)
  (cond ((null w) nil)
        ((odd-0-even-1enp (car w))
         (cons 1 (odd>0>even-1 (cdr w))))
        ((cons 0 (odd>0>even-1 (cdr w))))))

> (odd>0-even>1 '(2 3))
(1 0)
```

Задача 4.110 del-blast.lisp

Написать программу удаления предпоследнего элемента в списке.

Решение 4.110.1

```
(defun del-blast (w)
  (append (butlast w 2) (last w)))

> (del-blast '(a b c))
(A C)
```

Решение 4.110.2

```
(defun del-blast (w)
  (delete-if #'identity w :from-end t :start 1 :end 2))

> (del-blast '(a b c))
(A C)
```

Задача 4.111 sum-digits.lisp

Вычислить сумму цифр натуральных чисел. Вот пример ввода: (f 34 12 29 111 16 882), результат: (7 3 11 3 7 18).

Решение 4.111.1

```
(defun sn (n)
  (apply #'+
         (mapcar #'parse-integer
                 (map 'list #'string (write-to-string n)))))

(defun sum-digits (&rest w)
  (mapcar #'sn w))
```

```
> (sum-digits 34 12 29 111 16 882)
(7 3 11 3 7 18)
```

Решение 4.111.2

```
(defun sn (n)
  (if (< n 10)
      n
      ((lambda (r) (+ r (sn (/ (- n r) 10)))) (mod n 10))))

(defun sum-digits (&rest w)
  (mapcar #'sn w))

> (sum-digits 34 12 29 111 16 882)
(7 3 11 3 7 18)
```

* Что делают mapcar, apply и reduce?

```
> (funcall #'+ 1 1 1)
3

> (apply #'+ '(1 1 1))
3

> (mapcar #'1+ '(1 2 3))
(2 3 4)

> (mapcar #'+ '(1 2 3) '(1 2 3))
(2 4 6)

> (intersection '(a b) '(b c))
(B)

> (intersection '(a b c) '(b c d) '(c d a))
; Evaluation aborted

> (intersection (intersection '(a b c) '(b c d)) '(c d a))
(C)

> (reduce #'intersection '((a b c) (b c d) (c d a)))
(C)
```

Задача 4.112 sort-words.lisp

Записать предложение в порядке возрастания количества гласных букв в слове. Предложение - список слов.

Решение 4.112.1

```
(defun ws (w)
 (mapcar #'(lambda (a) (coerce (string a) 'list)) w))
```

```
(defun cn (w &optional (n 0))
  (cond ((null w) n)
        ((member (car w) '(#\A #\E #\I #\O))
         (cn (cdr w) (1+ n)))
        ((cn (cdr w) n))))

(defun ez (w)
  (mapcar #'(lambda (e) (list e (cn e))) w))

(defun ts (w)
  (mapcar
   #'(lambda (a) (intern (coerce (car a) 'string)))
   (sort w #'< :key #'cadr)))

(defun sort-words (w)
  (ts (ez (ws w))))

> (sort-words '(green red))
(RED GREEN)
```

Задача 4.113 list-caars.lisp

Построить список, состоящий из первых элементов вложенных списков. Например ((A B) (C D) (A D)) -> (A C A).

Решение 4.113.1

```
(defun list-caars (w)
  (cond ((null w) nil)
        ((cons (caar w) (list-caars (cdr w))))))

> (list-caars '((A B) (C D) (A D)))
(A C A)
```

Решение 4.113.2

```
(defun list-caars (w)
  (if (null w) nil (cons (caar w) (list-caars (cdr w)))))

> (list-caars '((A B) (C D) (A D)))
(A C A)
```

Решение 4.113.3

```
(defun list-caars (w)
  (mapcar #'car w))

> (list-caars '((A B) (C D) (A D)))
(A C A)
```

Задача 4.114 remove-odds.lisp

Написать функцию, удаляющую из списка члены с нечетными номерами.

Решение 4.114.1

```
(defun remove-odds  (w)
  (cond ((null (cdr w)) nil)
        ((cons (cadr w) (remove-odds (cddr w))))))

> (remove-odds  '(1 2 3 4 5))
(2 4)
```

Задача 4.115 smbs-only.lisp

Опишите функцию, которая из исходного списка формирует список, содержащий только символьные атомы.

Решение 4.115.1

```
(defun smbs-only (w)
  (cond ((null w) nil)
        ((symbolp (car w)) (cons (car w) (smbs-only (cdr w))))
        ((smbs-only (cdr w)))))

> (smbs-only '(1 a))
(A)
```

Задача 4.116 list-products.lisp

Сформировать список из элементов: произведение всех элементов исходного списка, произведение n-1 последних элементов, произведение n-2 последних элементов и т.д. Пример: для списка `(1 2 3 4 5 6) результатом будет: `(720 720 360 120 30 6).

Решение 4.116.1

```
(defun list-products (w)
  (cond ((null w) nil)
        ((cons (apply #'* w) (list-products (cdr w))))))

> (list-products '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Решение 4.116.2

```
(defun list-products (w)
  (when w  (cons (apply #'* w) (list-products (cdr w)))))

> (list-products '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Решение 4.116.3

```
(defun list-products (w)
  (maplist #'(lambda (a) (reduce #'* a)) w))

> (list-products '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Решение 4.116.4

```
(defun list-products (w)
  (maplist #'ma w))

(defun ma (v)
  (reduce #'* v))

> (list-products '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Задача 4.117 select-sort.lisp

Написать функцию сортировки списка методом прямого выбора.

Решение 4.117.1

```
(defun select-sort (w)
  (when w (let ((m (reduce #'min w)))
            (cons m (select-sort (remove m w :count 1))))))

> (select-sort '(1 2 3 4 1 2 3 4))
(1 1 2 2 3 3 4 4)
```

Задача 4.118 selection-sort.lisp

Написать функцию сортировки списка методом прямого выбора. Встроенные функции max и min не использовать. Можно использовать только средства строго функционального языка программирования (без использования функций присваивания).

Решение 4.118.1

```
(defun selection-sort (w)
  (when w (cons (mini w) (selection-sort (remove (mini w) w)))))

(defun mini (w)
  (mi (cdr w) (car w)))

(defun mi (w m)
  (cond ((null w) m)
        ((< (car w) m) (mi (cdr w) (car w)))
```

```
        ((mi (cdr w) m))))

> (selection-sort '(5 4 7 1 3))
(1 3 4 5 7)
```

Задача 4.119 sign-to-word.lisp

В списке заменить все атомы-числа: положительные на слово "положительное", отрицательные на слово "отрицательное", нулевые на "нуль".

Решение 4.119.1

```
(defun sign-to-word (w)
  (mapcar #'(lambda (a)
              (if (numberp a)
                  (cond ((zerop a) "zero")
                        ((plusp a) "positive")
                        (t "negative"))
                  a))
          w})

> (sine-toword '(a -1 0 1))
(A "negative" "zero" "positive")
```

Задача 4.120 del-dkrange.lisp

Из произвольного списка удалить латинские буквы расположенные в алфавите между d и k.

Решение 4.120.1

```
(defun del-dkrange (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((and (stringp a) (< 100 (char-code (char a 0)) 107))
         (del-dkrange d))
        ((cons a (del-dkrange d)))))

> (del-dkrange '("a" "e" "j" "z"))
("a" "z")
```

Решение 4.120.2

```
(defun del-dkrange (w)
  (delete-if
   #'(lambda (a)
       (and (stringp a) (< 100 (char-code (char a 0)) 107)))
   w))

> (del-dkrange '("a" "e" "j" "z"))
("a" "z")
```

Задача 4.121 rev&del-pairs.lisp

Дан список из десяти элементов, его надо инвертиртировать задом наперед и удалить из него равные элементы, которые стоят рядом. Т.е.:
2 3 3 4 4 5 6 3 4 4 - исходный список
4 4 3 6 5 4 4 3 3 2 - инвертированный
3652 - полученный.

Решение 4.121.1

```
(defun rev&del-pairs (w &optional ac &aux (a (car w)))
  (cond ((null w) ac)
        ((eq a (cadr w)) (rev&del-pairs (cddr w) ac))
        ((rev&del-pairs (cdr w) (push a ac)))))

> (rev&del-pairs '(2 3 3 4 4 5 6 3 4 4))
(3 6 5 2)
```

Задача 4.122 even-odd.lisp

Из исходного списка получить два списка, содержащие четные и нечетные элементы исходного списка.

Решение 4.122.1

```
(defun even-odd (w &optional e o &aux (a (car w)))
  (cond ((null w) (list e o))
        ((evenp a) (even-odd (cdr w) (cons a e) o))
        ((even-odd (cdr w) e (cons a o)))))

> (even-odd '(3 45 458 3 4 3 434))
((434 4 458) (3 3 45 3))
```

Задача 4.123 minus-plus.lisp

Дан список из атомов (-7 1 8 -2 -9 4 9 -1). Нужно отсортировать элементы так, чтобы сначала в списке были отрицательные, а потом положительные элементы. Причем они должны быть в той же последовательности, что и в изначальном списке, т.е. должен получиться такой список (-7 -2 -9 -1 1 8 4 9).

Решение 4.123.1

```
(defun minus-plus (w &optional m p &aux (a (car w)))
  (cond ((null w) (nconc (nreverse m) (nreverse p)))
        ((plusp a) (minus-plus (cdr w) m (cons a p)))
        ((minus-plus (cdr w) (cons a m) p))))

> (minus-plus '(-7 1 8 -2 -9 4 9 -1))
(-7 -2 -9 -1 1 8 4 9)
```

Задача 4.124 xlambda.lisp

Расставить скобки, чтобы функция вернула список (g66 fc l):

```
((lambda (x y z) (cons (car (cdr x)) (cons (car (cdr y)) (cons (car
(cdr (cdr z))) nil)))) '(g55 g66 g777) '(9 fc i) ' n i l t d j (ii
jj)))
```

Решение 4.124.1

```
> ((lambda (a b c) (cons (car (cdr a)) (cons (car (cdr b)) (cons (car
(cdr (cdr c))) nil)))) '(g55 g66 g777) '(9 fc i) '(n i l t d j (ii
jj)))
(G66 FC L)
```

Решение 4.124.2

```
> ((lambda (a b c) (cons (cadr a) (cons (cadr b) (cons (caddr c)
nil)))) '(g55 g66 g777) '(9 fc i) '(n i l t d j (ii jj)))
(G66 FC L)
```

Задача 4.125 append-reverse.lisp

Напишите функцию, которая соединяет два подсписка, причем оба подсписка должны быть обращены, например:

```
> (f '((a b) (c d))}
(B A D C)
```

Решение 4.125.1.

```
(defun append-reverse (w)
  (nconc (rv (car w)) (rv (cadr w))))

(defun rv (v &optional ac)
  (cond ((null v) ac)
        ((rv (cdr v) (push (car v) ac)))))

> (append-reverse '((a b) (c d)))
(B A D C)
```

Задача 4.126 grow-numbers.lisp

Дан список чисел ((x x x x...) (x x x x...)). Увеличить каждый элемент на единицу.

Решение 4.126.1

```
(defun grow-numbers (w)
  (mapcar #'(lambda (a) (loop for x in a collect (1+ x))) w))
```

```
> (grow-numbers '((1 2 3) (4 5 6) (7 8 9)))
((2 3 4) (5 6 7) (8 9 10))
```

Решение 4.126.2

```
(defun grow-numbers (w)
 (when w (cons (mapcar #'1+ (car w)) (grow-numbers (cdr w)))))
```

```
> (grow-numbers '((1 2 3) (4 5 6) (7 8 9)))
((2 3 4) (5 6 7) (8 9 10))
```

Решение 4.126.3

```
(defun grow-numbers (w)
   (mapcar #'(lambda (a) (mapcar #'1+ a)) w))
```

```
> (grow-numbers '((1 2 3) (4 5 6) (7 8 9)))
((2 3 4) (5 6 7) (8 9 10))
```

Задача 4.127 quicksort.lisp

Реализовать сортировку Хоара (англ. – Hoare) для списка.

Решение 4.127.1

```
(defun quicksort (w &aux (pivot (car w)))
  (if (> (length w) 1)
      (nconc (quicksort (remove-if-not #'(lambda (a) (< a pivot)) w))
             (remove-if-not #'(lambda (a) (= a pivot)) w)
             (quicksort (remove-if-not #'(lambda (a) (> a pivot)) w)))
      w))
```

```
> (quicksort '(7 4 9 2 4 3 8 1 0 6 9 3 4 2 5))
(0 1 2 2 3 3 4 4 4 5 6 7 8 9 9)
```

Решение 4.127.2

```
(defun quicksort (w &aux (pivot (car w)))
  (if (> (length w) 1)
      (nconc (quicksort (remove-if #'(lambda (a) (>= a pivot)) w))
             (remove-if #'(lambda (a) (/= a pivot)) w)
             (quicksort (remove-if #'(lambda (a) (<= a pivot)) w)))
      w))
```

```
> (quicksort '(7 4 9 2 4 3 8 1 0 6 9 3 4 2 5))
(0 1 2 2 3 3 4 4 4 5 6 7 8 9 9)
```

Решение 4.127.3

```
(defun quicksort (w &aux (pivot (car w)))
  (if (cdr w)
      (nconc (quicksort (remove-if #'(lambda (a) (>= a pivot)) w))
```

```
              (remove-if #'(lambda (a) (/= a pivot)) w)
              (quicksort (remove-if #'(lambda (a) (<= a pivot)) w)))
      w))

> (quicksort '(7 4 9 2 4 3 8 1 0 6 9 3 4 2 5))
(0 1 2 2 3 3 4 4 4 5 6 7 8 9 9)
```

Решение 4.127.4

```
(defun quicksort (w &aux (pivot (car w)))
  (if (cdr w)
      (nconc (quicksort (remove-if-not #'(lambda (a) (< a pivot)) w))
             (remove-if-not #'(lambda (a) (= a pivot)) w)
             (quicksort (remove-if-not #'(lambda (a) (> a pivot)) w)))
      w))

> (quicksort '(7 4 9 2 4 3 8 1 0 6 9 3 4 2 5))
(0 1 2 2 3 3 4 4 4 5 6 7 8 9 9)
```

Решение 4.127.5

```
(defun quicksort (w)
  (if (cdr w)
      (macrolet ((pivot (test) `(remove (car w) w :test-not #',test)))
        (nconc (quicksort (pivot >)) (pivot =) (quicksort (pivot <))))
      w))

> (quicksort '(8 4 5 0 2 3 9 5 2 3 9 5 4 2 8 3 7))
(0 2 2 2 3 3 3 4 4 5 5 5 7 8 8 9 9)
```

Задача 4.128 swap-elms.lisp

Определить функцию, которая меняет местами соседние элементы списка.

Решение 4.128.1

```
(defun swap-elms (w)
  (if (cdr w)
      (cons (cadr w)
            (cons (car w)
                  (swap-elms (cddr w))))
      w))

> (swap-elms '(1 2 3 4 5))
(2 1 4 3 5)
```

Задача 4.129 sin>cos-sqrt>abs.lisp

Заменить в заданном математическом предложении все вхождения sin на cos и sqrt на abs.

Решение 4.129.1

```
(defun sin>cos-sqrt>abs (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((listp a) (cons (sin>cos-sqrt>abs a) (sin>cos-sqrt>abs d)))
        ((eql a 'sin) (cons 'cos (sin>cos-sqrt>abs d)))
        ((eql a 'sqrt) (cons 'abs (sin>cos-sqrt>abs d)))
        ((cons a (sin>cos-sqrt>abs d)))))

> (sin>cos-sqrt>abs '((sin (sqrt 90)) + (sin (sqrt 90))))
((COS (ABS 90)) + (COS (ABS 90)))
```

Задача 4.130 popular-names.lisp

Имеется список студентов с информацией: фамилия, имя, отчество, пол, возраст и курс. Определить самое распространенное мужское и женское имя..

Решение 4.130.1

```
(defparameter w '((ivanov ivan ivanovich m 20 2)
                  (petrov ivan ivanovich m 20 2)
                  (petrova anna ivanovna f 20 2)
                  (petrova inna ivanovna f 20 2)
                  (ivanova anna ivanovna f 20 2)))

(defun popular-names (w &optional m f)
  (cond ((null w) (list (caar (sort m #'> :key #'cadr))
                        (caar (sort f #'> :key #'cadr))))
        ((member (cadar w) m :key #'car) (popular-names (cdr w) m f))
        ((member (cadar w) f :key #'car) (popular-names (cdr w) m f))
        ((eql (cadddr (car w)) 'm)
         (popular-names (cdr w)
                        (cons (list (cadar w)
                                    (count (cadar w) w :key #'cadr))
                              m)
                        f))
        ((popular-names (cdr w)
                        m
                        (cons (list (cadar w)
                                    (count (cadar w) w :key #'cadr))
                              f)))))

> (time (popular-names w))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 96 Bytes
(IVAN ANNA)
```

Решение 4.130.2

```
(defparameter w '((ivanov ivan ivanovich m 20 2)
```

```
                  (petrov ivan ivanovich m 20 2)
                  (petrova anna ivanovna f 20 2)
                  (petrova inna ivanovna f 20 2)
                  (ivanova anna ivanovna f 20 2)))

(defun drop-one (w &optional acc ac)
  (cond ((null w) acc)
        ((member (car w) ac) (drop-one (cdr w) (cons (car w) acc) ac))
        ((drop-one (cdr w) acc (cons (car w) ac)))))

(defun max-occur (w)
  (if (cdr w) (max-occur (drop-one w)) (car w)))

(defun names-s (w p)
  (cond ((null w) nil)
        ((eq (cadar w) p) (cons (caar w)  (names-s (cdr w) p)))
        ((names-s (cdr w) p))))

(defun names (w)
  (mapcar #'(lambda (a) (list (cadr a) (cadddr a))) w))

(defun popular-names (w)
  (mapcar #'max-occur (list (names-s (names w) 'm)
                            (names-s (names w) 'f))))

> (popular-names w)
(IVAN ANNA)
```

Задача 4.131 list-data.lisp

Написать функцию, которая из списка (Ann 20 160cm) строит список ((name Anna)(age 20)(height 151cm))

Решение 4.131.1

```
(defun list-data (w)
  (list (list 'name (car w))
        (list 'age (cadr w))
        (list 'height (caddr w))))

> (list-data '(Ann 20 160cm))
((NAME ANN) (AGE 20) (HEIGHT 160CM))
```

Решение 4.131.2

```
(defun list-data (w &optional (v '(name age height)))
  (loop for a in w
        for b in v collect (list b a)))

> (list-data '(Ann 20 160cm))
((NAME ANN) (AGE 20) (HEIGHT 160CM))
```

Задача 4.132 read-numbers.lisp

Прочитать файл целых чисел и вывести числа на экран.

Решение 4.132.1

```
(defun read-numbers (path)
  (with-open-file (s path :direction :input)
    (do ((line (read-line s nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
      (format t "~a~%" line))))
```

```
> (read-numbers "c:/numbers.txt")
0 1 2 3 4
5 6 7 8 9
NIL
```

Задача 4.133 cons-n.lisp

Сформировать из простого списка ассоциативный, поставив ключи в виде последовательных чисел.

Решение 4.133.1

```
(defun cons-n (w &optional (n 1))
  (when  w (cons (cons n (car w)) (cons-n (cdr w) (1+ n)))))
```

```
> (cons-n '(11 12 13 14))
((1 . 11) (2 . 12) (3 . 13) (4 . 14))
```

Решение 4.133.2

```
(defun cons-n (w)
  (loop for a in w
        for b upfrom 1
        collect (cons a b)))
```

```
> (cons-n '(11 12 13 14))
((1 . 11) (2 . 12) (3 . 13) (4 . 14))
```

Решение 4.133.3

```
(defun cons-n (w)
  (loop for a in w
        for b from 1
        collect (cons a b)))
```

```
> (cons-n '(11 12 13 14))
((1 . 11) (2 . 12) (3 . 13) (4 . 14))
```

Задача 4.134 drop-deep-duplicates.lisp

Определить функцию, которая удаляет дубликаты атомов в многоуровневом списке.

Решение 4.134.1

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun drop-duplicates (w)
  (cond ((null w) nil)
        ((member (car w) (cdr w)) (drop-duplicates (cdr w)))
        ((cons (car w) (drop-duplicates (cdr w))))))

(defun drop-deep-duplicates (w)
  (drop-duplicates (flat w)))

> (drop-deep-duplicates '((1) 2 (3 (4 1) 2) ((3)) 4 ((1) 2)))
(3 4 1 2)
```

Задача 4.135 meet.lisp

Определить функцию, которая преобразует список (a (b (c (d (e)))))) к виду (a b c d e).

Решение 4.135.1

```
(defun meet (w)
  (when w (cons (car w) (meet (cadr w)))))

> (meet '(a (b (c (d (e))))))
(A B C D E)
```

Задача 4.136 show-ironian-names.lisp

Задать структуру: фамилия, город, улица, дом, возраст. Определить функцию, которая возвращает фамилии студентов, живущих по одному адресу в разных городах.

Решение 4.136.1

```
(defstruct student name city str num apt age)

 (defun sp (n)
   (cond ((= n 0) nil)
         (t(cons (make-student
                  :name (read)
                  :city (read)
```

```
                  :str (read)
                  :num (read)
                  :apt (read)) (sp (- n 1))))))

(defun check-address (a b)
  (and (eq (student-str a) (student-str b))
       (eq (student-num a) (student-num b))
       (eq (student-apt a) (student-apt b))))

(defun ironian (a w)
  (cond ((null w) nil)
        ((check-address a (car w)) t)
        ((ironian a (cdr w)))))

(defun show-ironian-names (w)
  (loop for a in w when (ironian a (remove a w))
        collect (student-name a)))

> (setq w (sp 3))
ivanov
rostov
lenina
10
11
petrov
noginsk
lenina
100
111
petrova
podolsk
lenina
10
11
(#S(STUDENT :NAME IVANOV :CITY ROSTOV :STR LENINA :NUM 10 :APT 11)
 #S(STUDENT :NAME PETROV :CITY NOGINSK :STR LENINA :NUM 100 :APT 111)
 #S(STUDENT :NAME PETROVA :CITY PODOLSK :STR LENINA :NUM 10 :APT 11))
> (show-ironian-names w)
(IVANOV PETROVA)
```

Решение 4.136.2

```
(defstruct student name city str num apt age)

 (defun sp (n)
   (cond ((= n 0) nil)
         (t(cons (make-student
                  :name (read)
                  :city (read)
                  :str (read)
                  :num (read)
                  :apt (read)
                  :age (read)) (sp (- n 1))))))
```

```
(defun check-address (a b)
  (and (eq (student-str a) (student-str b))
       (eq (student-num a) (student-num b))
       (eq (student-apt a) (student-apt b))))

(defun ironian (a w)
  (cond ((null w) nil)
        ((check-address a (car w)) t)
        ((ironian a (cdr w)))))

(defun remove-elm (a w)
  (cond ((null w) nil)
        ((equalp (car w) a) (remove-elm a (cdr w)))
        ((cons (car w) (remove-elm a (cdr w))))))

(defun show-ironian (w v)
  (cond ((null w) nil)
        ((ironian (car w) (remove-elm  (car w) v))
         (cons (student-name (car w)) (show-ironian (cdr w) v)))
        ((show-ironian (cdr w) v))))

(defun show-ironian-names (w)
  (show-ironian w w))

> (setq w (sp 3))
ivanov
rostov
lenina
10
11
20
petrov
noginsk
lenina
100
111
20
petrova
podolsk
lenina
10
11
20
(#S(STUDENT :NAME IVANOV :CITY ROSTOV :STR LENINA :NUM 10 :APT 11 :AGE
20)
 #S(STUDENT :NAME PETROV :CITY NOGINSK :STR LENINA :NUM 100 :APT 111
:AGE 20)
 #S(STUDENT :NAME PETROVA :CITY PODOLSK :STR LENINA :NUM 10 :APT 11
:AGE 20))
> (show-ironian-names w)
(IVANOV PETROVA)
```

Задача 4.137 massp.lisp

Определите функцию, которая выдает список всех элементов, удовлетворяющих некоторому предикату и встречающихся в исходном списке более n раз.

Решение 4.137.1

```
(defun massp (n w p)
  (labels ((mass  (n v p)
             (cond ((null v) nil)
                   ((> (count (car v) v) n)
                    (cons (car v) (mass n (delete (car v) v) p)))
                   ((mass n (cdr v) p)))))
    (mass n (delete-if-not p w) p)))

> (massp  2 '(1 2 3 1 3 2 3 2 2) #'evenp)
(2)
> (massp  2 '(1 2 3 1 3 2 3 2 2) #'oddp)
(3)
> (massp  2 '(1 2 5 3 1 3 5 2 3 2 5 2) #'oddp)
(5 3)
```

Решение 4.137.2

```
(defun massp (n w p)
  (labels ((mass  (n v p)
             (when v (if (> (count (car v) v) n)
                         (cons (car v) (mass n (delete (car v) v) p))
                         (mass n (cdr v) p)))))
    (mass n (remove-if-not p w) p)))

> (massp  2 '(1 2 3 1 3 2 3 2 2) #'evenp)
(2)
> (massp  2 '(1 2 3 1 3 2 3 2 2) #'oddp)
(3)
> (massp  2 '(1 2 5 3 1 3 5 2 3 2 5 2) #'oddp)
(5 3)
```

Решение 4.137.2* (для XLISP)

```
(defun massp (n w p)
  (mass n w p nil))

(defun mass (n w p v)
  (cond ((null w) (reverse v))
        ((and (funcall p (car w))
              (not (mmb (car w) v))
              (> (cnt (car w) w) n))
         (mass n (cdr w) p (cons (car w) v)))
        ((mass n (cdr w) p v))))
```

```
(defun mmb (a w)
    (cond ((null w) nil)
    ((eql a (car w)) (cons (car w) (cdr w)))
    ((mmb a (cdr w)))))

(defun cnt (a w)
    (cond ((null w) 0)
          ((eql a (car w)) (+ 1 (cnt a (cdr w))))
          ((cnt a (cdr w)))))

> (massp  2 '(1 2 3 1 3 2 3 2 2) #'evenp)
(2)
> (massp  2 '(1 2 3 1 3 2 3 2 2) #'oddp)
(3)
> (massp  2 '(1 2 5 3 1 3 5 2 3 2 5 2) #'oddp)
(5 3)
```

Задача 4.138 word-right-p.lisp

Даны список строк и буква. Определить функцию, которая возвращает список строк оканчивающихся на эту букву.

Решение 4.138.1

```
(defun word-right-p (r w)
  (loop for a in w when (eq (car (last (coerce a 'list))) r)
        collect a))

> (word-right-p #\z '("aaab" "aaac" "aaaz" "bbbc" "bbbz"))
("aaaz" "bbbz")
```

Решение 4.138.2 * дополнительно вывести количество слов

```
(defun word-right (r w)
  (loop for a in w when (eq (car (last (coerce a 'list))) r)
        collect a into ws
        finally (return (values ws (length ws)))))

> (word-right #\z '("aaab" "aaac" "aaaz" "bbbc" "bbbz"))
("aaaz" "bbbz")
2
```

Решение 4.138.3 * дополнительно вывести количество слов

```
(defun word-right (r w)
  (loop for a in w when (eq (car (last (coerce a 'list))) r)
        collect a into ws
        finally (return (list ws (length ws)))))

> (word-right #\z '("aaab" "aaac" "aaaz" "bbbc" "bbbz"))
(("aaaz" "bbbz") 2)
```

Задача 4.139 ordered-chaos.lisp

Список целых чисел вернуть без изменений, если он упорядочен по возрастанию или убыванию. В противном случае: каждый четный по счету элемент списка утроить, а каждый элемент, стоящий на нечетном месте и кратный четырем, удалить.

Решение 4.139.1

```
(defun chaos (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((zerop (mod a 4)) (cons (* (car d) 3) (chaos (cdr d))))
        ((cons a (cons (* (car d) 3) (chaos (cdr d)))))))

(defun ordered (w)
  (or (null w) (apply #'<= w) (apply #'>= w)))

(defun ordered-chaos (w)
  (if (ordered w) w (chaos w)))

> (ordered-chaos '(1 7 3 4 8 6 12 8))
(1 21 3 12 18 24)
```

Задача 4.140 un-even.lisp

Проверить, является ли первый элемент списка четным числом. Если является, то вернуть исходный список, выполнив замену первого элемента на ближайшее большее нечетное число, иначе удалить первый элемент.

Решение 4.140.1

```
(defun un-even (w)
  (if (oddp (car w)) (cdr w) (cons (1+ (car w)) (cdr w))))

> (un-even '(2 1))
(3 1)
```

Задача 4.141 minus-odd.lisp

Проверить, является ли второй элемент списка отрицательным нечетным числом. Если является, то вернуть исходный список без последнего элемента, иначе поменять местами первый и второй элементы исходного списка.

Решение 4.141.1

```
(defun minus-odd (w)
  (if (and (minusp (cadr w)) (oddp (cadr w)))
      (butlast w)
      (cons (cadr w) (cons (car w) (cddr w)))))
```

```
> (minus-odd '(1 2))
(2 1)
> (minus-odd '(2 -1))
(2)
```

Задача 4.142 snow-flake.lisp

Создать функцию, которая сортирует список. Например, переделывает из списка (3 2 1 5 6 0 4) список (5 3 1 0 2 4 6). То есть, находит самое малое значение, помещает его в середину списка, а другие элементы списка раскладывает по убыванию и возрастанию вокруг наименьшего значения.

Решение 4.142.1

```
(defun snow-flake (w &optional ac &aux (v (sort w #'<)))
  (if w (snow-flake (cdr v) (cons (car v) (reverse ac)))
      (reverse ac)))

> (time (snow-flake '(3 2 1 5 6 0 4)))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 376 Bytes
(5 3 1 0 2 4 6)
```

Решение 4.142.2

```
(defun snow (w &optional ac)
  (if w (snow (cdr w) (cons (car w) (reverse ac)))
      (reverse ac)))

(defun snow-flake (w)
  (snow (sort w #'<)))

> (time (snow-flake '(3 2 1 5 6 0 4)))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 304 Bytes
(5 3 1 0 2 4 6)
```

Решение 4.142.3

```
(defun snow (w &optional ac)
  (if w (snow (cdr w) (cons (car w) (nreverse ac)))
      (nreverse ac)))

(defun snow-flake (w)
  (snow (sort w #'<)))

> (time (snow-flake '(3 2 1 5 6 0 4)))
Real time: 0.0 sec.
Run time: 0.0 sec.
```

```
Space: 80 Bytes
(5 3 1 0 2 4 6)
```

Задача 4.143 max>min-substitute.lisp

Дан массив целых чисел, если минимальный элемент меньше максимального в два раза, то все элементы, предшествующие максимальному элементу установить равными минимальному элементу.

Решение 4.143.1

```
(defun max>min (w n x)
  (if (eq (car w) x) w (cons n (max>min (cdr w) n x))))

(defun max>min-check (w n x)
  (if (= (* n 2) x) (max>min w n x) w))

(defun max>min-substitute (w)
  (max>min-check w  (apply #'min w) (apply #'max w)))

> (max>min-substitute '(1 5 3 2 1 2 3 5 6 12))
(1 5 3 2 1 2 3 5 6 12)
> (max>min-substitute '(5 5 3 4 4 3 5 6 3 4 5))
(3 3 3 3 3 3 6 3 4 5)
```

Задача 4.144 insert-asterisc.lisp

Определите функцию, вставляющую перед каждым элементом списка, обладающим определенным свойством, символ *

Решение 4.144.1

```
(defun insert-asterisc (p w &aux (a ( car w)) (d ( cdr w)))
  (cond ((null w) nil)
        ((funcall p a) (cons '* (cons a (insert-asterisc p d))))
        ((cons a (insert-asterisc p d)))))

> (insert-asterisc #'plusp '(-1 2 -3 -4 -5 -6 7 8 9))
(-1 * 2 -3 -4 -5 -6 * 7 * 8 * 9)
> (insert-asterisc #'(lambda (a) (not (minusp a))) '(0 -1 2 -3 -4))
(* 0 -1 * 2 -3 -4)
> (insert-asterisc #'evenp '(-1 2 -3 -4 -5 -6 7 8 9))
(-1 * 2 -3 * -4 -5 * -6 7 * 8 9)
```

Задача 4.145 pair-unpaired.lisp

Составить функцию, группирующую смежные элементы списка в подсписки, которые должны содержать по два элемента (последний подсписок может содержать один элемент).

Решение 4.145.1

```
(defun pair-unpaired (w)
  (if (cddr w)
      (cons (list (car w) (cadr w)) (pair-unpaired (cddr w)))
      (when  w (list w))))

> (pair-unpaired '(1 2 3 4 5))
((1 2) (3 4) (5))
> (pair-unpaired '(1 2 3 4))
((1 2) (3 4))
> (pair-unpaired '())
NIL
```

Задача 4.146 map-repeat.lisp

На входе список и число n, создать новый список, в котором каждый элемент из исходного повторяется n раз

Решение 4.146.1

```
(defun repeat (a n)
  (loop for i from 1 to n collect a))

(defun map-repeat (w n)
  (mapcan #'(lambda (a) (repeat a n)) w))

> (map-repeat '(1 2 3) 2)
(1 1 2 2 3 3)
```

Решение 4.146.2

```
(defun map-repeat (w n)
  (mapcan #'(lambda (a) (loop for i from 1 to n collect a)) w))

> (map-repeat '(1 2 3) 2)
(1 1 2 2 3 3)
```

Задача 4.147 surnames-common-ages.lisp

Имеется список студентов с информацией: фамилия, имя, отчество, пол, возраст, и курс. Определить фамилии тех студенток, возраст которых является самым распространённым.

Решение 4.147.1

```
(defstruct student surname name1 name2 sex age year)

(defun res (n)
  (when (> n 0)
    (cons (make-student :surname (read)
                        :name1 (read)
```

```
                        :name2 (read)
                        :sex (read)
                        :age (read)
                        :year (read))
          (res (1- n)))))

(defun surname-age (w)
  (mapcan #'(lambda (a)
              (when (eq (student-sex a) 'f)
                (list (list (student-surname a)
                            (student-age a)))))
          w))

(defun ages (w &optional ac)
  (if w (ages (cdr w) (pushnew (cadar w) ac)) ac))

(defun common-age (w v)
  (sort (mapcar
         #'(lambda (a) (list a (count a v :key #'cadr)))
         w) #'> :key #'cadr))

(defun age (w)
  (caar (common-age (ages (surname-age w)) (surname-age w))))

(defun surnames (w n)
  (cond ((null w) nil)
        ((eq (cadar w) n)
         (cons (caar w) (surnames (cdr w) n)))
        ((surnames (cdr w) n))))

(defun surnames-common-ages (w)
  (surnames (surname-age w) (age w)))
> (map-repeat '(1 2 3) 2)
(1 1 2 2 3 3)

> (setq w (res 3))
ivanova
anna
savishna
f
20
3
petrova
olga
andreevna
f
21
3
semenova
alina
sergeevna
f
20
```

```
2
(#S(STUDENT :SURNAME IVANOVA :NAME1 ANNA :NAME2 SAVISHNA :SEX F :AGE
20
    :YEAR 3)
 #S(STUDENT :SURNAME PETROVA :NAME1 OLGA :NAME2 ANDREEVNA :SEX F :AGE
21
    :YEAR 3)
 #S(STUDENT :SURNAME SEMENOVA :NAME1 ALINA :NAME2 SERGEEVNA :SEX F
:AGE 20
    :YEAR 2))
> (surnames-common-ages w)
(IVANOVA SEMENOVA)
```

Задача 4.148 xine-twins.lisp

Определить функцию, которая полностью удаляет из списка повторяющиеся элементы.

Решение 4.148.1

```
(defun xine-twins (w &optional ac &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((member a ac) (xine-twins d ac))
        ((member a d) (xine-twins d (cons a ac)))
        ((cons a (xine-twins d ac)))))

> (xine-twins '(a b a a c))
(B C)
```

Задача 4.149 count-atoms.lisp

Определить количество каждого атома в многоуровневом списке.

Решение 4.149.1

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun count-atoms (w &aux (h (make-hash-table)) ac)
  (dolist (a (flat w)) (incf (gethash a h 0)))
  (maphash #'(lambda (e n) (push `(,e . ,n) ac)) h) ac)
```

Задача 4.150 list-cadr.lisp

Дан список (a b c d). Необходимо преобразовать его в список (a (b) c (d)).

Решение 4.150.1

```
(defun list-cadr (w)
  (if (cdr  w)
      (cons (car w) (cons (list (cadr w)) (list-cadr (cddr w))))
      w))

> (list-cadr '(a b c d))
(A (B) C (D))
> (list-cadr '(a b c d e))
(A (B) C (D) E)
```

Задача 4.151 delete-on-list.lisp

Разработать функцию, удаляющую из исходного списка элементы, порядковые номера которых заданы во втором списке.

Решение 4.151.1

```
(defun delete-on-list (w v &optional (n 1))
  (cond ((null v) w)
        ((= (car v) n) (delete-on-list (cdr w) (cdr v) (1+ n)))
        ((cons (car w) (delete-on-list (cdr w) v (1+ n))))))

> (delete-on-list '(a s d f g h j) '(3 4 5))
(A S H J)
```

Решение 4.151.2 * для несортированного списка

```
(defun delete-on-list (w v &optional (n 1))
  (cond ((null v) w)
        ((= (car v) n) (delete-on-list (cdr w) (cdr v) (1+ n)))
        ((cons (car w) (delete-on-list (cdr w) v (1+ n))))))

(defun delete-sort (w v)
  (delete-on-list w (sort v #'<)))

> (time (delete-sort '(a b c d e f g h j k l m) '(10 9 8 7 5 4 3 2)))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 48 Bytes
(A F L M)
```

Решение 4.151.3 * для несортированного списка

```
(defun delete-sort (w v)
  (labels ((on-list (w v n)
             (cond ((null v) w)
                   ((= (car v) n) (on-list (cdr w) (cdr v) (1+ n)))
                   ((cons (car w) (on-list (cdr w) v (1+ n)))))))
    (on-list w (sort v #'<) 1)))

> (time (delete-sort '(a b c d e f g h j k l m) '(10 9 8 7 5 4 3 2)))
Real time: 0.0 sec.
```

```
Run time: 0.0 sec.
Space: 48 Bytes
(A F L M)
```

Только удаление:

```
(defun delete-sort (w v)
  (labels ((on-list (w v n)
             (cond ((null v) w)
                   ((= (car v) n) (on-list (cdr w) (cdr v) (1+ n)))
                   ((cons (car w) (on-list (cdr w) v (1+ n)))))))
    (on-list w v 1)))

> (time (delete-sort '(a b c d e f g h j k l m) '(2 3 4 5 7 8 9 10)))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 16 Bytes
(A F L M)
```

Отдельно сортировка:

```
(defun delete-sort (v) (sort v #'<))

> (time (delete-sort '(10 9 8 7 5 4 3 2)))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 32 Bytes
(2 3 4 5 7 8 9 10)
```

Задача 4.152 insection.lisp

Определить функцию (пересечение x y), результатом которой является список атомов (без повторения) одновременно входящих в x и y.

Решение 4.152.1

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun inter (w v)
  (cond ((null w) nil)
        ((or (not (member (car w) v)) (member (car w) (cdr w)))
         (inter (cdr w) v))
        ((cons (car w) (inter (cdr w) v)))))

(defun insection (w v)
  (inter (flat w) (flat v)))

> (insection '((1) 3 4 (3)) '((3 4) 5))
(4 3)
```

Решение 4.152.2

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun inter (w v &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((or (not (member a v)) (member a d)) (inter d v))
        ((cons a (inter d v)))))

(defun insection (w v)
  (inter (flat w) (flat v)))

> (insection '((1) 3 4 (3)) '((3 4) 5))
(4 3)
```

Решение 4.152.3

```
(defun unique (w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((member a d) (unique d))
        ((cons a (unique d)))))

(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun inter (w v &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((member a v) (cons a (inter d v)))
        ((inter d v))))

(defun insection (w v)
  (inter (unique (flat w)) (unique (flat v))))

> (insection '((1) 3 4 (3)) '((3 4) 5))
(4 3)
```

Задача 4.153 show-names.lisp

Имеется список студентов с информацией об итогах сессии, в котором указаны фамилия, номер группы, оценки по трем предметам. Определить фамилии студентов, имеющих задолженность хотя бы по одному предмету.

Решение 4.153.1

```
(defstruct student name group lisp smalltalk haskell)
```

```
(defun res (n)
  (when (> n 0)
    (cons (make-student
           :name (read)
           :group (read)
           :lisp (read)
           :smalltalk (read)
           :haskell (read)) (res (- n 1)))))

(defun check-2 (a)
  (or (eq (student-lisp a) 2)
      (eq (student-smalltalk a) 2)
      (eq (student-haskell a) 2)))

(defun show-names (w)
  (cond ((null w) nil)
        ((check-2 (car w))
         (cons (student-name (car w)) (show-names (cdr w))))
        ((show-names (cdr w)))))

> (setf w (res 3))
ivanov
2
4
5
2
petrov
4
4
4
4
sidorov
3
2
2
2
(#S(STUDENT :NAME IVANOV :GROUP 2 :LISP 4 :SMALLTALK 5 :HASKELL 2)
 #S(STUDENT :NAME PETROV :GROUP 4 :LISP 4 :SMALLTALK 4 :HASKELL 4)
 #S(STUDENT :NAME SIDOROV :GROUP 3 :LISP 2 :SMALLTALK 2 :HASKELL 2))
> (show-names w)
(IVANOV SIDOROV)
```

Задача 4.154 product-tail.lisp

Описать функцию, которая на основе списка чисел формирует список-результат следующим образом : первый элемент есть произведение элементов списка, второй – произведение элементов хвоста, третий – произведение элементов хвоста хвоста и так далее. Пример : для списка '(1 2 3 4 5 6) результатом будет : '(720 720 360 120 30 6).

Решение 4.154.1

```
defun product-tail (w)
  (cond ((null w) nil)
        ((cons (apply '* w) (product-tail (cdr w))))))

> (product-tail '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Решение 4.154.2

```
(defun product-tail (w)
  (when w (cons (apply '* w) (product-tail (cdr w)))))

> (product-tail '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Решение 4.154.3

```
(defun product-tail (w)
  (maplist #'(lambda (a) (apply #'* a)) w))

> (product-tail '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Решение 4.154.4

```
(defun product-tail (w)
  (maplist #'(lambda (a) (reduce #'* a)) w))

> (product-tail '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Решение 4.154.5

```
(defun product-tail (w)
  (loop for a on w collect (apply #'* a)))

> (product-tail '(1 2 3 4 5 6))
(720 720 360 120 30 6)
```

Задача 4.155 min-trio.lisp

Определить функцию, вычисляющую три минимальных элемента списка.

Решение 4.155.1

```
(defun min-trio (w) (subseq (sort w #'<) 0 3 ))

> (min-trio '(6 5 4 3 2 1 1 2 3))
(1 1 2)
```

Решение 4.155.2

```
(defun min-trio (w &optional (a (car w)) (b a) (c a))
  (cond ((null w) (list a b c))
        ((< (car w) a) (min-trio (cdr w) (car w) a b))
        ((< (car w) b) (min-trio (cdr w) a (car w) b))
        ((< (car w) c) (min-trio (cdr w) a  b (car w)))
        ((min-trio (cdr w) a b c))))

> (min-trio '(6 5 4 3 2 1 1 2 3))
(1 1 2)
```

Задача 4.156 delete-on.lisp

Определить функцию, удаляющую из списка все элементы, являющиеся подписками и числами, лежащими в некотором интервале.

Решение 4.156.1

```
(defun delete-on (w n m &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((or (listp a) (and (numberp a) (<= n a m)))
         (delete-on d n m))
        ((cons a (delete-on d n m)))))

> (delete-on '(1 2 3 4 11 22 33 (w e r) w e r) 2 4)
(1 11 22 33 W E R)
```

Задача 4.157 list>a.lisp

Определить функцию, заменяющую в списке все элементы, являющиеся вложенными списками на атом A.

Решение 4.157.1

```
(defun list>a (w)
  (cond ((null w) nil)
        ((listp (car w)) (cons 'a (list>a (cdr w))))
        ((cons (car w) (list>a (cdr w))))))

> (list>a '((3 4) 5 6 (5 (5 8))))
(A 5 6 A)
```

Задача 4.158 neighbors.lisp

Дан одноуровневый числовой список. Необходимо заменить каждое положительное число ближайшим отрицательным справа или слева. Если нет отрицательных - заменять на nil.

Решение 4.158.1

```
(defun mi (w) (find-if #'minusp w))
```

```
(defun neighbors (w &optional ac &aux (a (car w)) (d (cdr w)))
  (when w
    (if (plusp a)
        (cond ((and (some #'minusp ac) (some #'minusp d))
               (if (< (position-if #'minusp ac)
                      (position-if #'minusp d))
                   (cons (mi ac) (neighbors d (cons a ac)))
                   (cons (mi d) (neighbors d (cons a ac)))))
              ((some #'minusp d)
               (cons (mi d) (neighbors d (cons a ac))))
              ((some #'minusp ac)
               (cons (mi ac) (neighbors d (cons a ac))))
              ((cons nil (neighbors d (cons a ac)))))
        (cons a (neighbors d (cons a ac))))))

> (neighbors '(1 -2 3 -4 -5 6 7 7 7 -2 8))
(-2 -2 -4 -4 -5 -5 -5 -2 -2 -2 -2)
```

Решение 4.158.2

```
(defun mi (w) (find-if #'minusp w))

(defun sm (w) (some #'minusp w))

(defun ps (w) (position-if #'minusp w))

(defun ne (w a v) (neighbors w (cons a v)))

(defun neighbors (w &optional ac &aux (a (car w)) (d (cdr w)))
  (when w (if (plusp a)
              (cond ((and (sm ac) (sm d))
                     (if (< (ps ac) (ps d))
                         (cons (mi ac) (ne d a ac))
                         (cons (mi d) (ne d a ac))))
                    ((sm d) (cons (mi d) (ne d a ac)))
                    ((sm ac) (cons (mi ac) (ne d a ac)))
                    ((cons nil (ne d a ac))))
              (cons a (ne d a ac)))))

> (neighbors '(1 -2 3 -4 -5 6 7 7 7 -2 8))
(-2 -2 -4 -4 -5 -5 -5 -2 -2 -2 -2)
```

Задача 4.159 sbst.lisp

Определить функцию f (a w), где a – S-выражение, а w - список, которая заменяет на a все атомы списка x.

Решение 4.159.1

```
(defun sbst (a w) (substitute-if a #'atom w))

> (sbst '(+ 1 2) '(a b c))
```

```
((+ 1 2) (+ 1 2) (+ 1 2))
```

Задача 4.160 za-sentence.lisp

Дан текст. Переписать каждое предложение, расположив слова в обратном алфавитном порядке.

Решение 4.160.1

```
(defun za-words (s)
  (sort s  #'char-greaterp :key #'(lambda (a) (char a 0))))

(defun za-sentence (w)
  (mapcar #'za-words w))

> (za-sentence '(("aaa" "ccc" "bbb") ("ddd" "fff" "eee")))
(("ccc" "bbb" "aaa") ("fff" "eee" "ddd"))
```

Задача 4.161 pos/neg.lisp

Создать функцию, которая разделит исходный список из целых чисел на два списка: список положительных чисел и список отрицательных чисел.

Решение 4.161.1

```
(defun pos/neg (w)
  (loop for a in w
        if (plusp a) collect a into pos
        if (minusp a) collect a into neg
        finally (return (values pos neg))))

> (pos/neg '(-2 -1 0 1 2))
(1 2)
(-2 -1)
```

Задача 4.162 ini-lets.lisp

Создать функцию, которая возвращает список первых букв слов, представленных списками.

Решение 4.162.1

```
(defun ini-lets (w)
  (loop for a in w when a collect (car a)))

> (ini-lets'((W o r l d) (W a r) (I)))
(W W I)
```

Задача 4.163 check.lisp

Написать функцию, что для аргумента-списка формирует список-результат по правилу: если первый и последний элементы списка-аргумента - символы, то сформировать список из первого и последнего элементов, в противном случае вернуть исходный список, из которого изъят второй элемент.

Решение 4.163.1

```
(defun check (w &aux (a (car w)) (z (last w)))
  (cons a (if (every #'symbolp (cons a z)) z (cddr w))))

> (check '(a b c g e))
(A E)
> (check '(a b c g 10))
(A C G 10)
```

Задача 4.164 num-letters.lisp

Написать функцию, определяющую количество символов в каждом слове. Пример: исходная строка (abc defg hijk), результат (3 4 4)

Решение 4.164.1

```
(defun num-letters (w)
  (mapcar #'length  (mapcar #'symbol-name w)))

> (num-letters '(abc defg hijk))
(3 4 4)
```

Решение 4.164.2

```
(defun num-letters (w)
  (mapcar #'(lambda (a) (length (symbol-name a))) w))

> (num-letters '(abc defg hijk))
(3 4 4)
```

Решение 4.164.3

```
(defun num-letters (w)
  (loop for a in w collect (length (symbol-name a))))

> (num-letters '(abc defg hijk))
(3 4 4)
```

Решение 4.164.4

```
(defun num-letters (w)
  (mapcar #'length  (mapcar #'string w)))

> (num-letters '(a7bc d7efg h7ijk))
(4 5 5)
```

Решение 4.164.5

```
(defun num-letters (w)
  (mapcar #'(lambda (a) (length (string a))) w))

> (num-letters '(a7bc d7efg h7ijk))
(4 5 5)
```

Решение 4.164.6

```
(defun num-letters (w)
  (loop for a in w collect (length (string a))))

> (num-letters '(a7bc d7efg h7ijk))
(4 5 5)
```

Решение 4.164.7

```
(defun num-letters (w)
  (mapcar
   #'(lambda (a)
       (length (if (symbolp a) (string a) (write-to-string a)))) w))

> (num-letters '(abc d7efg 123))
(3 5 3)
```

Решение 4.164.8

```
(defun num-letters (w)
  (loop for a in w collect
        (length (if (symbolp a) (string a) (write-to-string a)))))

> (num-letters '(abc d7efg 123))
(3 5 3)
```

Задача 4.165 polish-notation.lisp

Разработать функцию, преобразующую арифметическое выражение, заданное в форме списка, в польскую обратную запись. Например: Вход: (3 * 2 - 5). Выход: (- * 3 2 5).

Решение 4.165.1

```
(defun polish-notation (w)
  (cond ((null w) nil)
        ((numberp (car w)) (cons (car w) (polish-notation (cdr w))))
        ((cons (cadr w) (cons (car w) (polish-notation (cddr w)))))))

> (polish-notation '(3 * 2 - 5))
(3 2 * 5 -)
```

Решение 4.165.2

```
(defun polish-notation (w &aux (a (car w)))
  (when w
    (if (numberp a)
        (cons a (polish-notation (cdr w)))
        (cons (cadr w)
              (cons a (polish-notation (cddr w)))))))

> (polish-notation '(3 * 2 - 5))
(3 2 * 5 -)
```

Задача 4.166 odd-even.lisp

Разработать функцию, увеличивающую в исходном списке нечетные элементы (по значению) на 1 и уменьшающую четные на 2.

Решение 4.166.1

```
(defun odd-even (w &aux (a (car w)))
  (when w (cons (if (oddp a) (1+ a) (- a 2)) (odd-even (cdr w)))))

> (odd-even '(1 2 3 4 5 6 7 8 9))
(2 0 4 2 6 4 8 6 10)
```

Задача 4.167 reverse-annex.lisp

Разработать функцию, которая преобразует список вида (a b c d) в список вида ((((d)c)b)a).

Решение 4.167.1

```
(defun reverse-annex (w &aux (v (reverse w)))
  (reduce #'list (cdr v) :initial-value (list (car v))))

> (reverse-annex '(a b c d e))
(((((E) D) C) B) A)
```

Задача 4.168 unique-symb.lisp

Из заданного предложения выбрать только те символы, которые встречаются в нем только один раз, в том порядке, как они расположены в тексте.

Решение 4.168.1

```
(defun unique-symb (w)
  (loop for a in w when (= (count a w) 1) collect a))

> (unique-symb '(aa f aa bb ff gg ff))
(F BB GG)
```

Задача 4.169 sum-shift.lisp

Разработать функцию, формирующую на основе исходного списка длины N список, содержащий суммы элементов с номерами 1 и N/2+1, 2 и N/2+2 и т. д. N — аргумент функции. Например: Вход: (1 2 3 4 5 6 7 8 9 10). Выход: (7 9 11 13 15).

Решение 4.169.1

```
(defun sum-shift (w &optional (v (nthcdr (/ (length w) 2) w)))
  (when v (cons (+ (car w) (car v)) (sum-shift (cdr w) (cdr v)))))

> (time (sum-shift (loop for i from 1 to 100  collect (random 100))))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 5436 Bytes
(106 167 154 34 61 50 71 147 190 115 118 120 67 36 188 89 184 121 93
74 106 95 135 64 94 147 158 45 116 73 87 120 101 73 154 67 114 35 32
105 160 169 155 78 101 92 107 120 114 92)
```

Решение 4.169.2

```
(defun sum-shift (w)
  (loop for a in w
        for b in (subseq w (/ (length w) 2))
        collect (+ a b)))

> (time (sum-shift (loop for i from 1 to 100  collect (random 100))))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 5836 Bytes
(115 93 24 87 54 83 10 49 76 79 41 30 75 172 102 148 68 85 80 166 66
84 115 180 128 69 28 87 164 107 151 56 114 124 150 146 130 153 60 87
177 111 92 101 93 80 96 11 78 72)
```

Решение 4.169.3

```
(defun sum-shift (w)
  (loop for a in w
        for b in (nthcdr (/ (length w) 2) w)
        collect (+ a b)))

> (time (sum-shift (loop for i from 1 to 100  collect (random 100))))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 5436 Bytes
(68 88 116 93 173 153 17 28 29 114 39 141 55 135 41 124 147 134 57 106
134 106 99 25 112 109 112 111 87 69 52 165 151 160 74 124 85 151 64
115 0 71 85 89 117 49 99 109 108 151)
```

Задача 4.170 del-list.lisp

Написать функцию, которая в списке удаляет все не атомарные элементы.

Решение 4.170.1

```
(defun del-list (w) (delete-if #'listp w))

> (del-list '(1 (2) 3 (4) 5))
(1 3 5)
```

Задача 4.171 add.lisp

Дан список. Увеличить каждый элемент на 1.

Решение 4.171.1

```
(defun add (w)
  (when w
    (cons (if (atom (car w)) (1+ (car w)) (add (car w)))
          (add (cdr w)))))

> (add '((1 2) (3 4) 6))
((2 3) (4 5) 7)
```

Решение 4.171.2

```
(defun add (w)
  (mapcar #'(lambda (a) (if (atom a) (1+ a) (add a))) w))

> (add '((1 2) (3 4) 6))
((2 3) (4 5) 7)
```

Решение 4.171.3

```
(defun add (w)
  (loop for a in w collect (if (atom a) (1+ a) (add a))))

> (add '((1 2) (3 4) 6))
((2 3) (4 5) 7)
```

Задача 4.172 increase.lisp

Увеличить каждый элемент списка в произвольное число раз. Анонимная функция должна быть передана параметром в вызываемую функцию.

Решение 4.172.1

```
(defun increase (w f &aux (a (car w)))
  (cond ((null w) nil)
        ((atom a) (cons (funcall f a) (increase (cdr w) f)))
        ((cons (increase a f) (increase (cdr w) f)))))
```

```
> (increase '(((1 2) (3 4)) (5)) (lambda (x) (* x 3)))
(((3 6) (9 12)) (15))
```

Задача 4.173 let-number.lisp

Дан список. Заменить буквы английского алфавита на соответствующий им номер в алфавите.

Решение 4.173.1

```
(defun let-number
    (w &aux
     (p (position
         (car w)
         '(a b c d e f g h i j k l m n o p q r s t u v w x y z))))
  (when w (cons (if p (1+ p) (car w)) (let-number (cdr w)))))

> (let-number '(a s d ! r z))
(1 19 4 ! 18 26)
```

Задача 4.174 list-unlist.lisp

Написать программу - элементы списка должны стать подсписками, а подсписки - элементами главного списка: (abc(de)ef(gk))->((abc)de(ef)gk).

Решение 4.174.1

```
(defun list-unlist (w &optional ac &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (when ac (cons (reverse ac) nil)))
        ((atom a) (list-unlist d (cons a ac)))
        (ac (nconc (cons (reverse ac) a) (list-unlist d nil)))
        ((nconc a (list-unlist d nil)))))


> (list-unlist '((a) a b c (d e) f g (h i j) k l))
(A (A B C) D E (F G) H I J (K L))
> (list-unlist '(a b c (d e) f g (h i j) k l))
((A B C) D E (F G) H I J (K L))
> (list-unlist '(a b c (d e) f g (h i j)))
((A B C) D E (F G) H I J)
```

Решение 4.174.2 (автор - VH, www.cyberforum.ru)

```
(defun list-unlist (w)
  (when w ((lambda (e r) (if (atom e)
                             (if (atom (car r))
                                 (cons (list e) r)
                                 (cons (cons e (car r)) (cdr r)))
                             (nconc e r)))
           (car w) (list-unlist (cdr w)))))
```

```
> (list-unlist '((a) b c (d e) f g (h i)))
(A (B C) D E (F G) H I)
```

Задача 4.175 last-first.lisp

Определите функцию, которая меняет местами первый и последний элементы списка, оставляя остальные на своих местах.

Решение 4.175.1

```
(defun last-first (w)
  (nconc (last w) (cdr (butlast w)) `(,(car w))))

> (last-first '(nil 1 2 3 4 5))
(5 1 2 3 4 NIL)
> (last-first '(1 2 3 4 5))
(5 2 3 4 1)
```

Задача 4.176 min-max.lisp

Найти элементы одноуровневого числового списка, имеющие минимальное значение и максимальное значение, вывести на экран их порядковые номера.

Решение 4.176.1

```
(defun min-max (w &aux
                (n (apply #'min w))
                (x (apply #'max w))
                (v (loop for a in w
                         for b from 0 to (1- (length w))
                         collect (list a b))))
  (loop for c in v
        if (eql (car c) n) collect (cadr c) into nn
        if (eql (car c) x) collect (cadr c) into xx
        finally (return (values nn xx))))

> (min-max '(1 2 3 4 1 2 3 4 1 2 3 4))
(0 4 8)
(3 7 11)
```

Решение 4.176.2* (для XLISP)

```
(defun min-max (w)
  (list-min-max (list-an w) (apply #'min w) (apply #'max w)))

(defun list-min-max (w n x &optional nn xx)
  (cond ((null w) (list (reverse nn) (reverse xx)))
        ((eql (caar w) n)
         (list-min-max (cdr w) n x (cons (cadar w) nn) xx))
        ((eql (caar w) x)
```

```
         (list-min-max (cdr w) n x nn (cons (cadar w) xx)))
        (t (list-min-max (cdr w) n x nn xx))))

(defun list-an (w &optional (n 0))
  (when w (cons (list (car w) n) (list-an (cdr w) (1+ n)))))

> (min-max '(1 2 3 4 1 2 3 4 1 2 3 4))
((0 4 8) (3 7 11))
```

Задача 4.177 letter-likeness.lisp

Для заданного текста определите пару слов, буквенный состав которого наиболее схож.

Решение 4.177.1

```
(defun letter-likeness (w)
  (mapcar #'
   (lambda (a) (read-from-string (apply #'concatenate 'string a)))
           (cdar (sort (len-w1-w2 (cartesian-mill (letters w)))
                       #'> :key #'car))))

(defun len-w1-w2 (w)
  (mapcar #'(lambda (a)
              (cons (length (intersection (str-sym (car a))
                                          (str-sym (cadr a))))
                    a)) w))

(defun str-sym (w)
  (mapcar #'(lambda (a) (read-from-string a)) w))

(defun cartesian-mill (w)
  (when (cdr w) (nconc (cartesian (list (car w)) (cdr w))
                       (cartesian-mill (cdr w)))))
(defun cartesian (w v)
  (mapcan (lambda (e) (mapcar (lambda (a) `(,a ,e)) w)) v))

(defun letters (w)
  (mapcar #'(lambda (a) (map 'list #'string (string a))) w))

> (letter-likeness '(abcde defgi bcdef))
(ABCDE BCDEF)
```

Задача 4.178 matrix-rotate-max-min.lisp

Найти в каждой строке наибольший и наименьший элементы матрицы и поменять их местами.

Решение 4.178.1

```
(defun rotate-max-min (w &aux (x (apply #'max w)) (n (apply #'min w)))
```

```
  (rotatef (nth (position x w) w) (nth (position n w) w)) w)

(defun matrix-rotate-max-min (w) (mapcar #'rotate-max-min w))

> (matrix-rotate-max-min '((1 2 3 4 5) (5 4 3 2 1) (1 2 3 4 5)))
((5 2 3 4 1) (1 4 3 2 5) (5 2 3 4 1))
```

Решение 4.178.2 (для XLISP)

```
(defun rotate-max-min
    (w &aux
     (x (apply #'max w))
     (n (apply #'min w))
     (px (pos x w))
     (pn (pos n w)))
  (let ((tmp (nth px w)))
    (setf (nth px w) (nth pn w) (nth pn w) tmp))
  w)

(defun pos (a w &optional (p 0))
  (cond ((null w) nil)
        ((eql (car w) a) p)
        (t (pos a (cdr w) (+ p 1)))))

(defun matrix-rotate-max-min (w) (mapcar #'rotate-max-min w))

> (matrix-rotate-max-min '((1 2 3 4 5) (5 4 3 2 1) (1 2 3 4 5)))
((5 2 3 4 1) (1 4 3 2 5) (5 2 3 4 1))
```

Задача 4.179 del-second.lisp

Найти в каждой строке наибольший и наименьший элементы матрицы и поменять их местами.

Решение 4.179.1

```
(defun del-second (w)
  (when w (cons (car w) (del-second (cddr w)))))

> (del-second '(1 2 3 4 5 6 7))
(1 3 5 7)
```

Решение 4.179.2

```
(defun del-second (w)
  (loop for a in w by #'cddr collect a))

> (del-second '(1 2 3 4 5 6 7))
(1 3 5 7)
```

Задача 4.180 sublist-cdddr.lisp

Дан многоуровневый список. Вывести все списки второго уровня, где количество элементов больше трех.

Решение 4.180.1

```
(defun sublist-cdddr (w)
  (remove-if-not #'(lambda (a) (and (listp a) (cdddr a))) w))

> (sublist-cdddr '(1 (2 3) (4 5 6 7) 8 (9 0)))
((4 5 6 7))
```

Задача 4.181 pos-max-min.lisp

Найти элементы одномерного числового массива, имеющие максимальное значение и минимальное значение, вывести на экран их порядковые номера.

Решение 4.181.1

```
(defun pos-max-min (w)
  (max-min (cdr w) (list (car w) 0) (list (car w) 0)))

(defun max-min (w x n &optional (u 1))
  (cond ((null w) (list (cadr x) (cadr n)))
        ((> (car w) (car x))
         (max-min (cdr w) (list (car w) u) n (1+ u)))
        ((< (car w) (car n))
         (max-min (cdr w) x (list (car w) u) (1+ u)))
        (t (max-min (cdr w) x n (1+ u)))))

> (time (pos-max-min (loop for i from 1 to 4096 collect (random
1000))))
Real time: 0.0312001 sec.
Run time: 0.0312002 sec.
Space: 37260 Bytes
(1622 605)
```

Решение 4.181.2

```
(defun pos-max-min (v)
  (mapcar #'(lambda (a) (position a v))
          `(,(apply #'max v) ,(apply #'min v))))

> (time (pos-max-min (loop for i from 1 to 4096 collect (random
1000))))
Real time: 0.0300004 sec.
Run time: 0.0312002 sec.
Space: 37036 Bytes
(468 937)
```

Решение 4.181.3

```
(defun pos-max-min (v)
  (values (pos #'max v) (pos #'min v)))

(defun pos (p v) (position (reduce p v) v))

> (time (pos-max-min (loop for i from 1 to 4096 collect (random
1000))))
Real time: 0.0337001 sec.
Run time: 0.0312002 sec.
Space: 37004 Bytes
GC: 1, GC time: 0.0 sec.
624
879
```

Задача 4.182 name-years.lisp

Напишите функцию, которая спрашивает у пользователя фамилию студента из группы и выдает год его рождения.

Решение 4.182.1

```
(defstruct student name year)

(defun group (n)
  (when (> n 0) (cons (make-student :name (read) :year (read))
                      (group (1- n)))))

(defun years (w input)
  (loop for a in w
        when (equalp (student-name a) input)
        collect (student-year a)))

(defun name-years (w)
 (years w (read)))

> (setf 3a (group 3))
ivanov
1994
petrov
1993
ivanov
1995
(#S(STUDENT :NAME IVANOV :YEAR 1994) #S(STUDENT :NAME PETROV :YEAR
1993) #S(STUDENT :NAME IVANOV :YEAR 1995))
> (name-years 3a)
petrov
(1993)
> (name-years 3a)
ivanov
(1994 1995)
```

Решение 4.182.2

```
(defparameter students '((Ivanov 1994) (Petrov 1993) (Ivanov 1995)))

(defun years (w input)
  (loop for a in w
        when (equalp (car a) input)
        collect (cadr a)))

(defun name-years (w)
 (years w (read)))

> (name-years students)
petrov
(1993)
> (name-years students)
ivanov
(1994 1995)
```

Решение 4.182.3

```
(defun years (w input)
  (loop for a in w
        when (equalp (car a) input)
        collect (cadr a)))

(defun name-years (w)
 (years w (read)))

> (name-years '((Ivanov 1994) (Petrov 1993) (Ivanov 1995)))
petrov
(1993)
> (name-years '((Ivanov 1994) (Petrov 1993) (Ivanov 1995)))
ivanov
(1994 1995)
```

Решение 4.182.4

```
(defun years (w input)
  (if (numberp input)
      (loop for a in w when (eq (cadr a) input)
            collect (car a))
      (loop for a in w when (eq (car a) input)
            collect (cadr a))))

(defun name-years (w)
 (years w (read)))

> (name-years '((Ivanov 1994) (Petrov 1993) (Ivanov 1995)))
ivanov
(1994 1995)
> (name-years '((Ivanov 1994) (Petrov 1993) (Ivanov 1995)))
```

```
1993
(PETROV)
```

Задача 4.183 flat-zero.lisp

Определить рекурсивную функцию, которая выравнивает список и применяет предикат zerop к каждому элементу списка.

Решение 4.183.1

```
(defun flat-zero (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons (zerop w) acc))
        ((flat-zero (car w) (flat-zero (cdr w) acc)))))

> (flat-zero '(((1)) 0 (1)))
(NIL T NIL)
```

Задача 4.184 n-atoms-n-lists.lisp

Создать рекурсивную функцию, которая считает сколько в списке атомов и сколько подсписков.

Решение 4.184.1

```
(defun n-atoms-n-lists (w)
  (list (n-atoms w) (n-lists w)))

(defun n-atoms (w)
  (cond ((null w) 0)
        ((atom w) 1)
        ((+ (n-atoms (car w)) (n-atoms (cdr w))))))

(defun n-lists (w)
  (cond ((null w) 0)
        ((atom (car w)) (n-lists (cdr w)))
        ((+ 1 (n-lists (car w)) (n-lists (cdr w))))))

> (n-atoms-n-lists '((1 2) (3 (4)) 5 ((6))))
(6 5)
```

Решение 4.184.2

```
(defun n-atoms-n-lists (w)
  (values (n-atoms w) (n-lists w)))

(defun n-atoms (w)
  (cond ((null w) 0)
        ((atom w) 1)
        ((+ (n-atoms (car w)) (n-atoms (cdr w))))))
```

```
(defun n-lists (w)
  (cond ((null w) 0)
        ((atom (car w)) (n-lists (cdr w)))
        ((+ 1 (n-lists (car w)) (n-lists (cdr w))))))

> (n-atoms-n-lists '((1 2) (3 (4)) 5 ((6))))
6
5
```

Решение 4.184.3

```
(defun n-atoms-n-lists (w)
  (values (n-atoms w) (n-lists w)))

(defun n-atoms (w)
  (if w
      (if (atom w)
          1
          (+ (n-atoms (car w)) (n-atoms (cdr w))))
      0))

(defun n-lists (w &aux(a (car w)))
  (if w
      (+ (if (atom a)
             0
             (1+ (n-lists a)))
         (n-lists (cdr w)))
      0))

> (n-atoms-n-lists '((1 2) (3 (4)) 5 ((6))))
6
5
```

Задача 4.185 multiply.lisp

Определить функцию (multiply (list-1) (list-2) ... (list-n)), которая перемножит почленно списки и вернет новый список.

Решение 4.185.1

```
(defun multiply (&rest w) (apply #'mapcar #'* w))

> (multiply '(1 2) '(10 20) '(100 200))
(1000 8000)
```

Задача 4.186 pair-chainge.lisp

Определить функцию, попарно меняющуя элементы списка.

Решение 4.186.1

```
(defun pair-change (w)
  (if (cdr w) (cons (cadr w) (cons (car w) (pair-change (cddr w))))
w))

> (pair-change '(1 2 3 4 5 6))
(2 1 4 3 6 5)
> (pair-change '(1 2 3 4 5 6 7))
(2 1 4 3 6 5 7)
```

Задача 4.187 del-over-average.lisp

Определить функцию, удаяющую из списка элементы, которые больше среднего арифметического.

Решение 4.187.1

```
(defun del-over-average (w)
  (over-aver w (/ (sum w) (len w))))

(defun len (w)
  (if w (1+ (len (cdr w))) 0))

(defun sum (w)
  (if w (+ (car w) (sum (cdr w))) 0))

(defun over-aver (w n)
  (cond ((null w) nil)
        ((> (car w) n) (over-aver (cdr w) n))
        ((cons (car w)(over-aver (cdr w) n)))))

> (del-over-average '(1 2 3 4 5 6 7))
(1 2 3 4)
```

Задача 4.188 eventh-max-minusp.lisp

Определить функцию, заменяющую значения элементов с четными номерами на значение максимального отрицательного элемента.

Решение 4.188.1

```
defun eventh-max-minusp (w)
  (eventh w (max-minusp w)))

(defun eventh (w n)
  (when (cdr w) (cons (car w) (cons n (eventh (cddr w) n)))))

(defun max-minusp (w)
  (reduce #'max (remove-if-not #'minusp w)))

> (eventh-max-minusp '(-2 -1 0 1 2 3))
(-2 -1 0 -1 2 -1)
```

Задача 4.189 _reverse.lisp

Определить функцию, обращающую список.

Решение 4.189.1

```
(defun _reverse (w &optional ac)
  (if w (_reverse (cdr w) (cons (car w) ac)) ac))

> (_reverse '(1 2 3 4 5))
(5 4 3 2 1)
```

Решение 4.189.2

```
(defun _reverse (w)
  (cond ((null w) nil)
        ((append (_reverse (cdr w)) (list (car w))))))

> (_reverse '(1 2 3 4 5))
(5 4 3 2 1)
```

Решение 4.189.3

```
(defun _reverse (w)
  (when w (append (_reverse (cdr w)) (list (car w)))))

> (_reverse '(1 2 3 4 5))
(5 4 3 2 1)
```

Решение 4.189.4

```
(defun _reverse (w)
  (cond ((null w) nil)
        ((_append (_reverse (cdr w)) (list (car w))))))

(defun _append (w v)
  (cond ((null w) v)
        ((cons (car w) (_append (cdr w) v)))))

> (_reverse '(1 2 3 4 5))
(5 4 3 2 1)
```

Решение 4.189.5

```
(defun _reverse (w)
  (when w (_append (_reverse (cdr w)) (list (car w)))))

(defun _append (w v)
  (if w (cons (car w) (_append (cdr w) v)) v))

> (_reverse '(1 2 3 4 5))
(5 4 3 2 1)
```

Решение 4.189.6

```
(defun _reverse (w)
  (when w (nconc (_reverse (cdr w)) (list (car w)))))

> (_reverse '(1 2 3 4 5))
(5 4 3 2 1)
```

Задача 4.190 drop-eventh.lisp

Определить функцию, удаляющую из списка все элементы с четными номерами.

Решение 4.190.1

```
(defun drop-eventh (w)
  (cond ((null w) nil)
        (t (cons (car w) (drop-eventh (cddr w))))))

> (drop-eventh '(1 2 3 4 5 6))
(1 3 5)
```

Решение 4.190.2

```
(defun drop-eventh (w)
  (cond ((null w) nil)
        (t (cons (car w) (drop-eventh (cddr w))))))

> (drop-eventh '(1 2 3 4 5 6))
(1 3 5)
```

Решение 4.190.3

```
(defun drop-eventh (w &optional f)
  (cond ((null w) nil)
        (f (drop-eventh (cdr w) nil))
        ((cons (car w) (drop-eventh (cdr w) t)))))

> (drop-eventh '(1 2 3 4 5 6))
(1 3 5)
```

Решение 4.190.4

```
(defun drop-eventh (w &optional f)
  (when w (if f
              (drop-eventh (cdr w) nil)
              (cons (car w) (drop-eventh (cdr w) t)))))

> (drop-eventh '(1 2 3 4 5 6))
(1 3 5)
```

```
Решение 4.190.5

(defun drop-eventh (w)
  (loop for a in w by #'cddr collect a))

> (drop-eventh '(1 2 3 4 5 6))
(1 3 5)
```

Задача 4.191 delm-thirds.lisp

```
Определить функцию, удаляющую каждое третье вхождение каждого элемента
в список.

Решение 4.191.1

(defun numbers (w &optional ac &aux (a (car w)))
  (when w (cons (1+ (count a ac)) (numbers (cdr w) (cons a ac)))))

(defun delm-third (w v)
  (cond ((null w) nil)
        ((zerop (rem (car v) 3)) (delm-third (cdr w) (cdr v)))
        ((cons (car w) (delm-third (cdr w) (cdr v))))))

(defun delm-thirds (w)
  (delm-third w (numbers w)))

> (delm-thirds '(a 1 s 2 d 3 a 1 s 2 2 d a 1 s d))
(A 1 S 2 D 3 A 1 S 2 D)

Решение 4.191.2

(defun numbers (w &optional ac &aux (a (car w)))
  (when w (cons (1+ (count a ac)) (numbers (cdr w) (cons a ac)))))

(defun delm-thirds (w)
  (loop for a in w for b in (numbers w)
        unless (zerop (rem b 3)) collect a))

> (delm-thirds '(a 1 s 2 d 3 a 1 s 2 2 d a 1 s d))
(A 1 S 2 D 3 A 1 S 2 D)
```

Задача 4.192 drop-sym-with-digits.lisp

```
Определить функцию, удаляющую из списка атомы, содержащие цифры.

Решение 4.192.1

(defun drop-sym-with-digits (w)
  (labels
      ((drop (v)
         (cond ((null v) nil)
```

```
               ((some #'digit-char-p
                      (coerce (string (car v)) 'list))
                (drop (cdr v)))
               ((cons (car v) (drop (cdr v)))))))
    (drop (remove-if #'numberp w))))

> (drop-sym-with-digits '(asd gh f12s 6 7 a s h7))
(ASD GH A S)
```

Решение 4.192.2

```
(defun drop (w)
  (cond ((null w) nil)
        ((some #'digit-char-p
               (coerce (string (car w)) 'list))
         (drop (cdr w)))
        ((cons (car w) (drop (cdr w))))))

(defun drop-sym-with-digits (w)
  (drop (remove-if #'numberp w)))

> (drop-sym-with-digits '(asd gh f12s 6 7 a s h7))
(ASD GH A S)
```

Решение 4.192.3

```
(defun drop-sym-with-digits (w)
  (remove-if
   #'(lambda (a)
       (some #'digit-char-p
             (coerce (string a) 'list)))
   (remove-if #'numberp w)))

> (drop-sym-with-digits '(asd gh f12s 6 7 a s h7))
(ASD GH A S)
```

Решение 4.192.4

```
(defun drop-sym-with-digits (w)
  (remove-if
   #'(lambda (a)
       (or (numberp a)
           (some #'digit-char-p
                 (coerce (string a) 'list))))
   w))

> (drop-sym-with-digits '(asd gh f12s 6 7 a s h7))
(ASD GH A S)
```

Задача 4.193 drop-core.lisp

Определить функцию удаления элемента списка, стоящего в середине. Если

количество элементов четное то удалить два элемента, что в середине.

Решение 4.193.1

```
(defun drop-core (w &aux
                  (n (length w))
                  (h (floor n 2))
                  (m (zerop (rem n 2))))
  (when w (if m
              (nconc (subseq w 0 (1- h)) (subseq w (1+ h)))
              (nconc (subseq w 0 h) (subseq w (1+ h))))))


> (drop-core '(1 2 3 4))
(1 4)
> (drop-core '(1 2 3))
(1 3)
> (drop-core '(1 2))
NIL
> (drop-core '(1))
NIL
> (drop-core '())
NIL
```

Решение 4.193.2

```
(defun drop-core (w &aux
                  (n (length w))
                  (h (floor n 2))
                  (m (zerop (rem n 2))))
  (when w (nconc (if m (subseq w 0 (1- h)) (subseq w 0 h))
                 (subseq w (1+ h)))))

> (drop-core '(1 2 3 4))
(1 4)
```

Решение 4.193.3

```
(defun drop-core (w &aux (n (length w)) (h (floor n 2)))
  (when w (nconc (if (zerop (rem n 2))
                     (subseq w 0 (1- h))
                     (subseq w 0 h))
                 (subseq w (1+ h)))))

> (drop-core '(1 2 3 4))
(1 4)
```

Задача 4.194 sum-underscore-code.lisp

Составить рекурсивную функцию для обработки списка из любого количества атомов, которые составлены из любых знаков. Каждый атом составить из суммы цифр, знака подчеркивания и ASCII кодов букв. Пример: на вход

подается список (2d 4r 1s). На выходе должно быть (7_100 7_114 7_115).

Решение 4.194.1

```
(defun sum-digits (w &aux (a (car w)) (d (cdr w)))
  (if w (+ (apply #'+ (mapcar #'digit-char-p
                              (remove-if-not
                               #'digit-char-p
                               (coerce (string a) 'list))))
           (sum-digits d))
      0))

(defun charr (a)
  (car (remove-if-not #'alpha-char-p
                      (coerce (string-downcase (string a))
                              'list))))

(defun sum-code (w s)
  (mapcar
   #'(lambda (a)
       (concatenate 'string
                    (write-to-string s)
                    "_"
                    (write-to-string (char-code (charr a)))))
         w))

(defun sum-underscore-code (w)
  (mapcar #'intern (sum-code w (sum-digits w))))

> (sum-underscore-code '(2d 4r 1s))
(|7_100| |7_114| |7_115|)
```

Решение 4.194.2

```
(defun mapcr (p w)
  (when w (cons (funcall p (car w)) (mapcr p (cdr w)))))

(defun rem-if-not (p w)
  (cond ((null w) nil)
        ((funcall p (car w)) (cons (car w) (rem-if-not p (cdr w))))
        ((rem-if-not p (cdr w)))))

(defun app (w)
  (if w (+ (car w) (app (cdr w))) 0))

(defun sum-digits (w &aux (a (car w)) (d (cdr w)))
  (if w (+ (app (mapcr #'digit-char-p
                       (rem-if-not
                        #'digit-char-p
                        (coerce (string a)
                                'list))))
           (sum-digits d))
      0))
```

```
(defun charr (a)
  (car (rem-if-not #'alpha-char-p
                   (coerce (string-downcase (string a)) 'list))))

(defun sum-code (w s)
  (mapcr
   #'(lambda (a)
       (concatenate 'string
                    (write-to-string s)
                    "_"
                    (write-to-string (char-code (charr a)))))
         w))

(defun sum-underscore-code (w)
  (mapcr #'intern (sum-code w (sum-digits w))))

> (sum-underscore-code '(2d 4r 1s))
(|7_100| |7_114| |7_115|)
```

Задача 4.195 _maplist.lisp

Определите функционал, аналогичный встроенному предикату maplist для одноуровневого списка. Используйте применяющие функционалы. Проверьте работу функционала для функции reverse.

Решение 4.195.1

```
(defun _maplist (p w)
  (when w (cons (funcall p w) (_maplist p (cdr w)))))

> (_maplist #'reverse '(1 2 3 4 5 6))
((6 5 4 3 2 1) (6 5 4 3 2) (6 5 4 3) (6 5 4) (6 5) (6))
```

Задача 4.196 sum-list-atom.lisp

Дан список, который состоит из подсписков и атомов. Необходимо построить список атомов и сумм каждого подсписка.

Решение 4.196.1

```
(defun sum-list-atom (w)
  (when w (cons (if (atom (car w)) (car w) (apply #'+ (car w)))
                (sum-list-atom (cdr w)))))

> (sum-list-atom '(1 (1 1)))
(1 2)
```

Решение 4.196.2

```
(defun sum-list-atom (w)
```

```
  (loop for a in w collect (if (atom a) a (apply #'+ a))))

> (sum-list-atom '(1 (1 1)))
(1 2)
```

Задача 4.197 drop-elm.lisp

Удалить заданный элемент из списка на всех уровнях вложенности.

Решение 4.197.1

```
(defun drop-elm (a w)
  (cond ((null w) nil)
        ((equal a (car w)) (drop-elm a (cdr w)))
        ((consp (car w))
         (cons (drop-elm a (car w)) (drop-elm a (cdr w))))
        ((cons (car w) (drop-elm a (cdr w))))))

> (drop-elm '(2) '(1 ((1 2 (2)) 1) (2) 1 2))
(1 ((1 2) 1) 1 2)
```

Задача 4.198 flat-unique.lisp

Определить функцию (множество w), аргументом которой является список с подсписками, а результатом – список атомов без повторения.

Решение 4.198.1

```
(defun flat-unique (w &optional ac)
  (cond ((null w) ac)
        ((atom w) (if (member w ac) ac (cons w ac)))
        ((flat-unique (car w) (flat-unique (cdr w) ac)))))

> (flat-unique '((a b) ((a)) a c))
(B A C)
```

Задача 4.199 compress.lisp

Определить функцию, которая оставляет в списке w из каждой группы подряд идущих одинаков элементов только один.

Решение 4.199.1

```
(defun compress (w &optional b)
  (cond ((null w) nil)
        ((equal (car w) b) (compress (cdr w) b))
        ((cons (car w) (compress (cdr w) (car w))))))

> (compress '(a a b c c c d e e))
(A B C D E)
```

Решение 4.199.2

```
(defun compress (w)
  (cond ((null w) nil)
        ((equal (car w) (cadr w)) (compress (cdr w)))
        ((cons (car w)
               (labels ((press (w b)
                          (cond ((null w) nil)
                                ((equal (car w) b) (press (cdr w) b))
                                ((cons (car w)
                                       (press (cdr w) (car w)))))))
                 (press (cdr w) nil))))))
```

```
> (compress '(nil nil nil a a b c nil nil nil c c d e e nil nil nil))
(NIL A B C NIL C D E NIL)
```

Решение 4.199.3

```
(defun compress (w &optional b f)
  (cond ((null w) nil)
        (f (if (equal (car w) b)
               (compress (cdr w) b t)
               (cons (car w) (compress (cdr w) (car w) t))))
        ((cons (car w) (compress (cdr w) (car w) t)))))
```

```
> (compress '(nil nil nil a a b c nil nil nil c c d e e nil nil nil))
(NIL A B C NIL C D E NIL)
```

Решение 4.199.4

```
(defun compress (w &optional b f)
  (cond ((null w) nil)
        ((and f (equal (car w) b)) (compress (cdr w) b t))
        ((cons (car w) (compress (cdr w) (car w) t)))))
```

```
> (compress '(nil nil nil a a b c nil nil nil c c d e e nil nil nil))
(NIL A B C NIL C D E NIL)
```

Задача 4.200 del-deviant.lisp

Определить функцию, которая удаляет из числового списка числа, наруша-
ющие возрастающий порядок.

Решение 4.200.1

```
(defun del-deviant (w &optional (b (car w)))
  (cond ((null w) nil)
        ((> b (car w)) (del-deviant (cdr w) b))
        ((cons (car w) (del-deviant (cdr w) (car w))))))
```

```
> (del-deviant '(1 2 3 4 1))
(1 2 3 4)
```

```
> (del-deviant '(1 2 3 1 4))
(1 2 3 4)
```

Решение 4.200.2

```
(defun del-deviant (w)
  (if (cdr w)
      (if (<= (car w) (cadr w))
          (cons (car w) (del-deviant (cdr w)))
          (del-deviant (cons (car w) (cddr w))))
      w))

> (del-deviant '(1 2 3 4 1))
(1 2 3 4)
> (del-deviant '(1 2 3 1 4))
(1 2 3 4)
```

Решение 4.200.3

```
(defun del (w a)
  (if (< a (car w)) w (cons a w)))

(defun del-deviant (w)
  (when w
    (nreverse
     (reduce #'del (cdr w) :initial-value `(,(car w))))))

> (del-deviant '(1 2 3 1 4))
(1 2 3 4)
> (del-deviant '(1 2 3 4 1))
(1 2 3 4)
```

Решение 4.200.4

```
(defun del-deviant (w)
  (cond ((null (cdr w)) w)
        ((< (cadr w) (car w)) (del-deviant (cons (car w) (cddr w))))
        ((cons (car w) (del-deviant (cdr w))))))

> (del-deviant '(1 2 3 1 4))
(1 2 3 4)
> (del-deviant '(1 2 3 4 1))
(1 2 3 4)
```

Задача 4.201 maxeq.lisp

Определить функцию, которая находит в одноуровневом числовом списке возрастающий подсписок максимальной длины.

Решение 4.201.1

```
(defun seq (w &optional (b (car w)))
```

```
  (cond ((null w) nil)
        ((> b (car w)) nil)
        ((cons (car w) (seq (cdr w) (car w))))))

(defun maxeq (w &optional ac &aux (v (seq w)))
  (cond ((null w) ac)
        ((> (length v) (length ac)) (maxeq (cdr w) v))
        ((maxeq (cdr w) ac))))

> (maxeq '(1 2 3 1 2 3 4 1 2))
(1 2 3 4)
```

Решение 4.201.2

```
(defun seq (w &optional (b (car w)))
  (when w (when (<= b (car w)) (cons (car w) (seq (cdr w) (car w))))))

(defun maxeq (w &optional ac &aux (v (seq w)))
  (cond ((null w) ac)
        ((> (length v) (length ac)) (maxeq (cdr w) v))
        ((maxeq (cdr w) ac))))

> (maxeq '(1 2 3 1 2 3 4 1 2))
(1 2 3 4)
```

Решение 4.201.3

```
(defun seq (w &optional (b (car w)))
  (when w (when (<= b (car w)) (cons (car w) (seq (cdr w) (car w))))))

(defun maxeq (w &optional ac &aux (v (seq w)))
  (if w
      (if  (> (length v) (length ac))
           (maxeq (cdr w) v)
           (maxeq (cdr w) ac))
      ac))

> (maxeq '(1 2 3 1 2 3 4 1 2))
(1 2 3 4)
```

Решение 4.201.4

```
(defun seq (w &optional (b (car w)))
  (when w (when (<= b (car w)) (cons (car w) (seq (cdr w) (car w))))))

(defun maxeq (w &optional ac &aux (v (seq w)))
  (if w (maxeq (cdr w) (if (> (length v) (length ac)) v ac)) ac))

> (maxeq '(1 2 3 1 2 3 4 1 2))
(1 2 3 4)
```

Решение 4.201.5

```
(defun seq (w &optional (b (car w)))
  (when (and w (<= b (car w))) (cons (car w) (seq (cdr w) (car w)))))

(defun maxeq (w &optional ac &aux (v (seq w)))
  (if w (maxeq (cdr w) (if (> (length v) (length ac)) v ac)) ac))

> (maxeq '(1 2 3 1 2 3 4 1 2))
(1 2 3 4)
```

Решение 4.201.6

```
(defun check (w v)
  (cond ((null w) nil)
        ((null v) t)
        ((check (cdr w) (cdr v)))))

(defun seq (w &optional (b (car w)))
  (when (and w (<= b (car w))) (cons (car w) (seq (cdr w) (car w)))))

(defun maxeq (w &optional ac &aux (v (seq w)))
  (if w (maxeq (cdr w) (if (check v ac) v ac)) ac))

> (maxeq '(1 2 3 1 2 3 4 1 2))
(1 2 3 4)
```

Решение 4.201.7

```
(defun check (w v)
  (when w (if  v (check (cdr w) (cdr v)) t)))

(defun seq (w &optional (b (car w)))
  (when (and w (<= b (car w))) (cons (car w) (seq (cdr w) (car w)))))

(defun maxeq (w &optional ac &aux (v (seq w)))
  (if w (maxeq (cdr w) (if (check v ac) v ac)) ac))

> (maxeq '(1 2 3 1 2 3 4 1 2))
(1 2 3 4)
```

Задача 4.202 max-chain.lisp

Дан список, элементами которого являются списки, состоящие с 3 элементов: две буквы и одна цифра. Необходимо составить цепочки из списков (по буквам без учета цифр) и вывести ту цепочку, сумма третьих элементов которых максимальная.

Решение 4.202.1

```
(defun del (v w)
  (cond ((null w) nil)
        ((equal (car w) v) (del v (cdr w)))
        ((cons (car w) (del v (cdr w))))))
```

```
(defun chains (w z)
  (when w (cons (chain w z) (chains (cdr w) z))))

(defun chain (w z &optional (b (car w)) &aux (v (same b z)))
  (if v (cons b (chain (cdr w) (del b (del v z)) v)) `(,b)))

(defun same (v w)
  (cond ((null w) nil)
        ((eq (cadr v) (caar w)) (car w))
        ((same v (cdr w)))))

(defun lens (w)
  (mapcar #'(lambda (a) (apply #'+ (mapcar #'caddr a))) w))

(defun max-chain (w &aux (v (chains w w)) (z (lens v)))
  (values v (nth (position (apply #'max z) z) v)))


> (max-chain '((c e 5) (a b 1) (d k 2) (b c 3) (f p 4) (p f 6) (k b
7)))
(((C E 5)) ((A B 1) (B C 3) (C E 5)) ((D K 2) (K B 7) (B C 3) (C E 5))
((B C 3) (C E 5)) ((F P 4) (P F 6)) ((P F 6) (F P 4)) ((K B 7) (B C 3)
(C E 5)))
((D K 2) (K B 7) (B C 3) (C E 5))
```

Задача 4.203 our-mapcon.lisp

Построить функцию mapcon - аналог maplist, но объединяет результаты (подсписки) в один список.

Решение 4.203.1

```
(defun our-mapcon (f w)
  (when w (cons (apply f w) (our-mapcon f (cdr w)))))

> (our-mapcon #'list '(1 2 3 4 5))
((1 2 3 4 5) (2 3 4 5) (3 4 5) (4 5) (5))
> (mapcon #'list '(1 2 3 4 5))
((1 2 3 4 5) (2 3 4 5) (3 4 5) (4 5) (5))
```

Задача 4.204 drop-last.lisp

Определить рекурсивную функцию удаления последнего элемента списка

Решение 4.204.1

```
(defun drop-last (w)
  (when (cdr w) (cons (car w) (drop-last (cdr w)))))

> (drop-last '(1 2 3 4))
```

```
(1 2 3)
```

Задача 4.205 deep-reverse.lisp

Дан многоуровневый список. Построить новый список, изменив порядок элементов в исходном списке на обратный и изменить порядок на обратный в каждом подсписке.

Решение 4.205.1

```
(defun deep-reverse (w &optional acc &aux (a (car w)))
  (if w
      (deep-reverse (cdr w)
                    (push (if (atom a) a (deep-reverse a)) acc))
      acc))

> (deep-reverse '(1 (((2 3) 4) 5) (6 (7))))
(((7) 6) (5 (4 (3 2))) 1)
```

Решение 4.205.2

```
(defun deep-reverse (w &optional acc)
  (if w
      (deep-reverse (cdr w)
                    (cons (if (atom (car w))
                              (car w)
                              (deep-reverse (car w)))
                          acc))
      acc))

> (deep-reverse '(1 (((2 3) 4) 5) (6 (7))))
(((7) 6) (5 (4 (3 2))) 1)
```

Задача 4.206 not-numbers-and-sum.lisp

Определить функцию, возвращающую элементы исходного списка, не являющиеся числами, и сумму всех элементов – чисел.

Решение 4.206.1

```
(defun not-numbers-and-sum (w)
  (values (remove-if #'numberp w)
          (apply #'+ (remove-if-not #'numberp w))))

> (not-numbers-and-sum '(1 a (2 3) (a b) 2 (4)))
(A (2 3) (A B) (4))
3
```

Решение 4.206.2

```
(defun not-numbers-and-sum (w)
```

```
  (loop for a in w
        if (numberp a) collect a into numbers
        else collect a into notnumbers
        finally return (values notnumbers (apply #'+ numbers))))

> (not-numbers-and-sum '(1 a (2 3) (a b) 2 (4)))
(A (2 3) (A B) (4))
3
```

Задача 4.207 elm-share.lisp

Определить функцию, вычисляющую отношения количества атомов – не чисел, чисел, подсписков к общему числу элементов исходного списка y, и записывающую эти отношения в список.

Решение 4.207.1

```
(defun elm-share (w &aux (z (length w)))
  (when w (list (/ (count-if #'symbolp w) z)
                (/ (count-if #'numberp w) z)
                (/ (count-if #'listp w) z))))

> (elm-share '(1 a (2 3) (a b) 2 (4)))
(1/6 1/3 1/2)
```

Задача 4.208 smash-mx.lisp

Пусть дана прямоугольная матрица a(m*n), элементами которой являются целые числа. Замените все положительные четные числа на числа, являющиеся их "перевертышами". Составьте функцию, получающую для заданного числа его "перевертыш" (число a будем считать "перевертышем" числа b, если, читая a справа на лево, получим число b).

Решение 4.208.1

```
(defun reverse-digits (n &optional (m 0))
  (cond ((zerop n) m)
        (t (multiple-value-bind (q r) (truncate n 10)
             (reverse-digits q (+ (* m 10) r))))))

(defun smash (w &aux (a (car w)))
  (cond ((null w) nil)
        ((and (numberp a) (plusp a) (evenp a))
         (cons (reverse-digits a) (smash (cdr w))))
        ((cons a (smash (cdr w))))))

(defun smash-mx (w)
  (mapcar #'smash w))

> (smash-mx '((12 45 -72) (a b 15) (49 c 18)))
((21 45 -72) (A B 15) (49 C 81))
```

Решение 4.208.2

```
(defun reverse-digits (n)
  (labels ((next (n v)
             (if (zerop n) v
                 (multiple-value-bind (q r)
                     (truncate n 10)
                   (next q (+ (* v 10) r))))))
    (next n 0)))

(defun smash (w)
 (loop for a in w
       if (and (numberp a) (plusp a) (evenp a))
       collect (reverse-digits a)
       else collect a))

(defun smash-mx (w)
  (mapcar #'smash w))

> (smash-mx '((12 45 -72) (a b 15) (49 c 18)))
((21 45 -72) (A B 15) (49 C 81))
```

Решение 4.208.3

```
(defun reverse-digits (n)
  (parse-integer (reverse (write-to-string n))))

(defun smash (w)
 (mapcar #'(lambda (a)
             (if (and (numberp a) (plusp a) (evenp a))
                 (reverse-digits a)
                 a))
         w))

(defun smash-mx (w)
  (mapcar #'smash w))

> (smash-mx '((12 45 -72) (a b 15) (49 c 18)))
((21 45 -72) (A B 15) (49 C 81))
```

Двлее, в одну функцию (плохой стиль):

Решение 4.208.4

```
(defun smash-mx (w)
  (loop for v in w
        collect
        (loop for a in v
              if (and (numberp a) (plusp a) (evenp a))
              collect (parse-integer (reverse (write-to-string a)))
              else collect a)))
```

```
> (smash-mx '((12 45 -72) (a b 15) (49 c 18)))
((21 45 -72) (A B 15) (49 C 81))
```

Решение 4.208.5

```
(defun smash-mx (w)
  (mapcar #'
   (lambda (v)
     (mapcar #'
      (lambda (a) (if (and (numberp a) (plusp a) (evenp a))
                      (parse-integer (reverse (write-to-string a)))
                      a))
      v))
   w))
```

```
> (smash-mx '((12 45 -72) (a b 15) (49 c 18)))
((21 45 -72) (A B 15) (49 C 81))
```

Задача 4.209 primes.lisp

Определить функцию, которая проверяет, являются ли элементы списка простыми числами (то есть, делящимися нацело только на себя и на единицу). Если элемент является простым числом, функция возвращает в результирующем списке значение T. Если элемент не является простым числом, функция возвращает в результирующем списке NIL.

Решение 4.209.1

```
(defun primep (n)
  (loop for i from 2 to (isqrt n) never (zerop (rem n i))))

(defun primes (w)
  (mapcar #'primep w))
```

```
> (primes '(2 3 4 5 6 7 8 9 10 11))
(T T NIL T NIL T NIL NIL NIL T)
```

Решение 4.209.2

```
(defun primes (w)
  (mapcar #'(lambda (n) (loop for i from 2 to (isqrt n)
                              never (zerop (rem n i)))) w))
```

```
> (primes '(2 3 4 5 6 7 8 9 10 11))
(T T NIL T NIL T NIL NIL NIL T)
```

Задача 4.210 all-chains.lisp

Сормировать такие цепочки из списка списков, в которых последний элемент предыдущего списка совпадает с первым следующего. Пример: ((a 2) (n 6) (2 f) (a v) (6 b) (f p) (d a)) => ((a 2) (2 f) (f p)) и ((n 6)

(6 b)) и т.п. И чтобы подсписки, которые уже включены в какую-нибудь из цепочек, сразу после этого удалялись из основного списка.

Решение 4.210.1

```
(defun del (v w)
  (cond ((null w) nil)
        ((equal (car w) v) (del v (cdr w)))
        ((cons (car w) (del v (cdr w))))))

(defun chains (w z)
  (when w (cons (chain w z) (chains (cdr w) z))))

(defun clear (v w)
  (cond ((null v) nil)
        ((some #'(lambda (e) (subsetp (car v) e :test #'equalp))
               (remove (car v) w))
         (clear (cdr v) w))
        ((cons (car v) (clear (cdr v) w)))))

(defun chain (w z &optional (b (car w)) &aux (v (same b z)))
  (if v (cons b (chain (cdr w) (del b (del v z)) v)) `(,b)))

(defun same (v w)
  (cond ((null w) nil)
        ((eq (cadr v) (caar w)) (car w))
        ((same v (cdr w)))))

(defun all-chains (w &aux (z (chains w w)))
  (delete-if-not #'cdr (clear z z)))

> (all-chains '((a 2) (n 6) (2 f) (a v) (6 b) (f p) (d a)))
(((N 6) (6 B)) ((D A) (A 2) (2 F) (F P)))
```

Решение 4.210.2

```
(defun chains (w z)
  (when w (cons (chain w z) (chains (cdr w) z))))

(defun clear (v w)
  (cond ((null v) nil)
        ((some #'(lambda (e) (subsetp (car v) e :test #'equal))
               (remove (car v) w))
         (clear (cdr v) w))
        ((cons (car v) (clear (cdr v) w)))))

(defun chain (w z &optional (b (car w)) &aux (v (same b z)))
  (if v (cons b (chain (cdr w) (remove b (remove v z)) v)) `(,b)))

(defun same (v w)
  (cond ((null w) nil)
        ((eq (cadr v) (caar w)) (car w))
        ((same v (cdr w)))))
```

```
(defun all-chains (w &aux (z (chains w w)))
  (delete-if-not #'cdr (clear z z)))

> (all-chains '((a 2) (n 6) (2 f) (a v) (6 b) (f p) (d a)))
(((N 6) (6 B)) ((D A) (A 2) (2 F) (F P)))
```

Задача 4.211 reverse-letters.lisp

В списке слова зашифрованы – каждое из них написано наоборот. Написать программу расшифровки.

Решение 4.211.1

```
(defun reverse-letters (w)
  (mapcar #'reverse w))

> (reverse-letters '("latrommi" "sdeed"))
("immortal" "deeds")
```

Задача 4.212 reverse-words.lisp

В строке слова зашифрованы – каждое из них написано наоборот. Написать программу расшифровки.

Решение 4.212.1

```
(defun reverse-words (s)
  (string-right-trim
   " "
   (format
    nil "~{~s ~}"
    (mapcar
     #'intern
     (mapcar
      #'reverse
      (mapcar
       #'string
       (read-from-string
        (concatenate
         'string
         "(" s ")")))))))))

> (time (reverse-words "latrommi sdeed"))
Space: 1340 Bytes
"IMMORTAL DEEDS"
```

Решение 4.212.2

```
(defun reverse-words (s)
  (string-left-trim
```

```
   " "
   (apply
    #'concatenate
    'string
    (mapcar
     #'reverse
     (mapcar
      #'(lambda (a) (concatenate 'string a " "))
      (mapcar
       #'string
       (read-from-string
        (concatenate 'string "(" s ")"))))))))

> (time (reverse-words "latrommi sdeed"))
Space: 492 Bytes
"IMMORTAL DEEDS"
```

Решение 4.212.3

```
(defun rv (w &optional ac &aux (a (car w)))
  (cond ((null w) ac)
        ((alpha-char-p a) (rv (cdr w) (cons a ac)))
        ((nconc ac (cons a (rv (cdr w) nil))))))

(defun reverse-words (s)
  (coerce (rv (loop for c across s collect c)) 'string))

> (time (reverse-words "latrommi sdeed"))
Space: 288 Bytes
"immortal deeds"
```

Решение 4.212.4

```
(defun rv (w &optional ac &aux (a (car w)))
  (cond ((null w) ac)
        ((alpha-char-p a) (rv (cdr w) (cons a ac)))
        ((nconc ac (cons a (rv (cdr w) nil))))))

(defun reverse-words (s)
  (coerce (rv (map 'list #'identity s)) 'string))

> (time (reverse-words "latrommi sdeed"))
Space: 288 Bytes
"immortal deeds"
```

Решение 4.212.5

```
(defun rv (w &optional ac &aux (a (car w)))
  (cond ((null w) ac)
        ((alpha-char-p a) (rv (cdr w) (cons a ac)))
        ((nconc ac (cons a (rv (cdr w) nil))))))

(defun reverse-words (s)
```

```
  (coerce (rv (coerce s 'list)) 'string))

> (time (reverse-words "latrommi sdeed"))
Space: 288 Bytes
"immortal deeds"
```

Задача 4.213 reverse-input.lisp

Вводимые слова зашифрованы – каждое из них написано наоборот. Написать программу расшифровки.

Решение 4.213.1

```
(defun reverse-input ()
  (string-right-trim
   " "
   (format
    nil "~{~s ~}"
    (mapcar
     #'intern
     (mapcar
      #'reverse
      (mapcar
       #'string
       (read-from-string
        (concatenate
         'string
         "(" (read-line) ")")))))))))

> (reverse-input)
latrommi sdeed
"IMMORTAL DEEDS"
```

Решение 4.213.1

```
(defun rv (w &optional ac)
  (cond ((null w) ac)
        ((alpha-char-p (car w)) (rv (cdr w) (cons (car w) ac)))
        ((nconc ac (cons (car w) (rv (cdr w) nil))))))

(defun reverse-input ()
  (coerce (rv (coerce (read-line) 'list)) 'string))

> (reverse-input)
latrommi sdeed
"immortal deeds"
```

Задача 4.214 eventh-min.lisp

Заменить значения элементов с нечетными номерами на значение минимального положительного элемента.

Решение 4.214.1

```
(defun eventh-min (w)
  (ev w (apply #'min (remove-if-not #'plusp w))))

(defun ev (w m)
  (if (cdr w) (cons (car w) (cons m (ev (cddr w) m))) w))
```

```
> (eventh-min '(0 1 2 3 4 5 6 7 8))
(0 1 2 1 4 1 6 1 8)
> (eventh-min '(0 1 2 3 4 5 6 7))
(0 1 2 1 4 1 6 1)
```

Задача 4.215 second-half.lisp

Определить функцию, возвращающую вторую половину списка.

Решение 4.215.1

```
(defun second-half (w)
  (nthcdr (truncate (/ (length w) 2)) w))
```

```
> (second-half '(1 2 3 4 5 6))
(4 5 6)
> (second-half '(1 2 3 4 5 6 7))
(4 5 6 7)
```

Задача 4.216 compress&ratio.lisp

Составьте функцию "сжатия" исходной последовательности символов, которая заменяет последовательность, состоящую из одинаковых символов, текстом вида x(k), где x-символ последовательности, k-число вхождений. Определите для указанной последовательности коэффициент сжатия (отношение исходной длины последовательности к полученной).

Решение 4.216.1

```
(defun compress (w &optional (n 1) ac (m (count-nil (reverse w))))
  (cond ((null w) (if (zerop m)
                      (reverse ac)
                      (if (= m 1)
                          (reverse (cons nil ac))
                          (reverse (cons (list nil m) ac)))))
        ((equal (car w) (cadr w)) (compress (cdr w) (1+ n) ac m))
        ((> n 1) (compress (cdr w) 1 (cons (list (car w) n) ac) m))
        ((compress (cdr w) 1 (cons (car w) ac) m))))

(defun count-nil (w)
  (if (or (null w) (car w)) 0 (1+ (count-nil (cdr w)))))
```

```
(defun compress&ratio (w &aux (v (compress w)))
  (values v (float (/ (length w) (length v)))))

> (compress&ratio '(nil nil a b b b c c c d nil nil nil))
((NIL 2) A (B 3) (C 3) D (NIL 3))
2.1666667
```

Решение 4.216.2

```
(defun compress (w &optional (n 1) ac (m (count-nil (reverse w))))
  (cond ((null w)
         (if (zerop m)
             (reverse ac)
             (reverse (cons (if (= m 1) nil (list nil m)) ac))))
        ((equal (car w) (cadr w)) (compress (cdr w) (1+ n) ac m))
        ((> n 1) (compress (cdr w) 1 (cons (list (car w) n) ac) m))
        ((compress (cdr w) 1 (cons (car w) ac) m))))

(defun count-nil (w)
  (if (or (null w) (car w)) 0 (1+ (count-nil (cdr w)))))

(defun compress&ratio (w &aux (v (compress w)))
  (values v (float (/ (length w) (length v)))))

> (compress&ratio '(nil nil a b b b c c c d nil nil nil))
((NIL 2) A (B 3) (C 3) D (NIL 3))
2.1666667
```

Решение 4.216.3

```
(defun compress (w &optional (n 1) ac)
  (cond ((null w) (reverse ac))
        ((equal w '(nil))
         (compress (cdr w) 1 (cons (if (= n 1)
                                       (car w )
                                       (list (car w) n))
                                   ac)))
        ((equal (car w) (cadr w)) (compress (cdr w) (1+ n) ac))
        ((> n 1) (compress (cdr w) 1 (cons (list (car w) n) ac)))
        ((compress (cdr w) 1 (cons (car w) ac)))))

(defun compress&ratio (w &aux (v (compress w)))
  (values v (float (/ (length w) (length v)))))

> (compress&ratio '(nil nil a b b b c c c d nil nil nil))
((NIL 2) A (B 3) (C 3) D (NIL 3))
2.1666667
```

Решение 4.216.4

```
(defun mp (w &optional (n 1) ac)
  (cond ((null w) (reverse ac))
        ((or (not (eq (car w) (cadr w))) (equal w '(nil)))
         (mp (cdr w) 1 (cons (if (= n 1) (car w) (list (car w) n))
```

```
                          ac)))
        ((mp (cdr w) (1+ n) ac))))

(defun compress&ratio (w &aux (v (mp w)))
  (values v (float (/ (length w) (length v)))))

> (compress&ratio '(nil nil a b b b c c c d nil nil nil))
((NIL 2) A (B 3) (C 3) D (NIL 3))
2.1666667
```

Задача 4.217 plus-num-sum.lisp

Определить количество положительных элементов одномерного массива. Вывести на экран их количество и значение их суммы.

Решение 4.217.1

```
(defun cnt (w)
  (cond ((null w) 0)
        ((plusp (car w)) (1+ (cnt (cdr w))))
        ((cnt (cdr w)))))

(defun sum (w)
  (cond ((null w) 0)
        ((plusp (car w)) (+ (car w) (sum (cdr w))))
        ((sum (cdr w)))))

(defun plus-num-sum (w)
  (values (cnt w) (sum w)))

> (plus-num-sum '(-1 0 1 2 3))
3
6
```

Решение 4.217.2

```
(defun plus-num-sum (w &aux (v (delete-if-not #'plusp w)))
  (values (length v) (apply #'+ v)))

> (plus-num-sum '(-1 0 1 2 3))
3
6
```

Решение 4.217.3 * для списков с числом элементов больше 4096

```
(defun plus-num-sum (w &aux (v (delete-if-not #'plusp w)))
  (values (length v) (reduce #'+ v)))

> (plus-num-sum '(-1 0 1 2 3))
3
6
```

Задача 4.218 div3.lisp

Выбрать из числового списка все элементы, делящиеся на 3, и сформировать из них список.

Решение 4.218.1

```
(defun div3 (w)
  (cond ((null w) nil)
        ((zerop (rem (car w) 3)) (cons (car w) (div3 (cdr w))))
        ((div3 (cdr w)))))

> (div3 '(1 2 3 4 5 6 7 8 9 10))
(3 6 9)
```

Решение 4.218.2

```
(defun div3 (w)
  (remove-if-not #'(lambda (a) (zerop (rem a 3))) w))

> (div3 '(1 2 3 4 5 6 7 8 9 10))
(3 6 9)
```

Задача 4.219 div.lisp

Выбрать из числового списка все элементы, делящиеся на n, и сформировать из них список.

Решение 4.219.1

```
(defun div (w n)
  (cond ((null w) nil)
        ((zerop (rem (car w) n)) (cons (car w) (div (cdr w) n)))
        ((div (cdr w) n))))

> (div '(1 2 3 4 5 6 7 8 9 10) 3)
(3 6 9)
> (div '(1 2 3 4 5 6 7 8 9 10) 4)
(4 8)
```

Решение 4.219.2

```
(defun div (w n)
  (remove-if-not #'(lambda (a) (zerop (rem a n))) w))

> (div '(1 2 3 4 5 6 7 8 9 10) 3)
(3 6 9)
> (div '(1 2 3 4 5 6 7 8 9 10) 4)
(4 8)
```

Решение 4.219.3

```
(defun div (w n)
  (loop for a in w when (zerop (rem a n)) collect a))

> (div '(1 2 3 4 5 6 7 8 9 10) 3)
(3 6 9)
> (div '(1 2 3 4 5 6 7 8 9 10) 4)
(4 8)
```

Задача 4.220 column-plus-sum-num.lisp

Вычислить сумму и число положительных элементов каждого столбца матрицы.

Решение 4.220.1

```
(defun transpose (m) (apply #'mapcar #'list m))

(defun plus-num (w)
  (mapcar #'(lambda (a) (count-if #'plusp a)) w))

(defun plus-sum (w)
  (mapcar #'(lambda (e) (apply #'+ e))
          (mapcar #'(lambda (a) (delete-if-not #'plusp a)) w)))

(defun column-plus-sum-num (w &aux (v (transpose w)))
  (values (plus-sum v) (plus-num v)))

> (column-plus-sum-num '((-1 2 3) (4 -5 6) (7 8 -9)))
(11 10 9)
(2 2 2)
```

Решение 4.220.2

```
(defun transpose (m) (apply #'mapcar #'list m))

(defun plus-num (w)
  (mapcar #'(lambda (a) (length a)) w))

(defun plus-sum (w)
  (mapcar #'(lambda (e) (apply #'+ e)) w))

(defun column-plus-sum-num
    (w &aux (v (mapcar
                #'(lambda (u) (remove-if-not #'plusp u))
                (transpose w))))
  (values (plus-sum v) (plus-num v)))

> (column-plus-sum-num '((-1 2 3) (4 -5 6) (7 8 -9)))
(11 10 9)
(2 2 2)
```

Задача 4.221 nx-insert.lisp

Дан список чисел. Добавить элемент с максимальным значением после минимального элемента.

Решение 4.221.1

```
(defun nx-insert (w)
  (insert w (reduce #'min w) (reduce #'max w)))

(defun insert (w n x)
  (cond ((null w) nil)
        ((eq (car w) n) (cons n (cons x (insert (cdr w) n x))))
        ((cons (car w) (insert (cdr w) n x)))))

> (nx-insert '(0 1 2 5 1 2 3 0 1 2 3 0 4))
(0 5 1 2 5 1 2 3 0 5 1 2 3 0 5 4)
```

Решение 4.221.2

```
(defun nx-insert
    (w &optional (n (reduce #'min w)) (x (reduce #'max w)))
  (cond ((null w) nil)
        ((eq (car w) n) (cons n (cons x (nx-insert (cdr w) n x))))
        ((cons (car w) (nx-insert (cdr w) n x)))))

> (nx-insert '(0 1 2 5 1 2 3 0 1 2 3 0 4))
(0 5 1 2 5 1 2 3 0 5 1 2 3 0 5 4)
```

Задача 4.222 max-min-mx.lisp

Найти в каждой строке матрицы максимальный и минимальный элементы и поместить их на место первого и последнего элемента строки соответственно.

Решение 4.222.1

```
(defun max-min-mx (w)
  (mapcar #'max-min w))

(defun max-min (w &aux (x (apply #'max w)) (n (apply #'min w)))
  (cons x (append (butlast (cdr w)) `(,n))))

> (max-min-mx '((1 2 3) (4 5 6) (7 8 9)))
((3 2 1) (6 5 4) (9 8 7))
```

Решение 4.222.2

```
(defun max-min (w)
  (cons (apply #'max w)
        (nconc (butlast (cdr w)) `(,(apply #'min w)))))
```

```
(defun max-min-mx (w)
  (mapcar #'max-min w))

> (max-min-mx '((1 2 3) (4 5 6) (7 8 9)))
((3 2 1) (6 5 4) (9 8 7))
```

Решение 4.222.3

```
(defun max-min-mx (w)
  (mapcar #'(lambda (v) (cons (apply #'max v)
                              (nconc (butlast (cdr v))
                                     `(,(apply #'min v)))))
          w))

> (max-min-mx '((1 2 3) (4 5 6) (7 8 9)))
((3 2 1) (6 5 4) (9 8 7))
```

Задача 4.223 sets.lisp

Для заданного атомного списка построить список всех подмножеств этого списка.

Решение 4.223.1

```
(defun sets (w)
  (if w ((lambda (r)
           (nconc r (mapcar #'(lambda (a) (cons (car w) a)) r)))
         (sets (cdr w)))
      '(nil)))

> (sets '(a b c))
(NIL (C) (B) (B C) (A) (A C) (A B) (A B C))
```

Задача 4.224 integers.lisp

Определить функцию, проверяющую являются ли элементы списка натуральными числами. Если элемент является натуральными числом, функция возвращает в результирующем списке значение T. Если элемент не является натуральными числом, функция возвращает в результирующем списке NIL.

Решение 4.224.1

```
(defun integers (w)
  (mapcar #'integerp w))

> (integers '(1 2 3 a a 2.3 4 5.6))
(T T T NIL NIL NIL T NIL)
```

Решение 4.224.2

```
(defun integers (w)
```

```
  (loop for a in w collect (integerp a)))

> (integers '(1 2 3 a a 2.3 4 5.6))
(T T T NIL NIL NIL T NIL)
```

Решение 4.224.3

```
(defun integers (w)
  (when w (cons (integerp (car w)) (integers (cdr w)))))

> (integers '(1 2 3 a a 2.3 4 5.6))
(T T T NIL NIL NIL T NIL)
```

Задача 4.225 rotate.lisp

Написать функцию, меняющую местами элементы списка w с номерами n и m.

Решение 4.225.1

```
(defun rotate (w a b &optional ac n acc f)
  (cond ((zerop b)
         (nreconc ac (cons (car w) (nreconc acc (cons n (cdr w))))))
        (f (rotate (cdr w) 0 (1- b) ac n (cons (car w) acc) t))
        ((zerop a) (rotate (cdr w) 0 (1- b) ac (car w) acc t))
        ((rotate (cdr w) (1- a) (1- b) (cons (car w) ac) n acc nil))))

> (setf w '(1 2 3 4 5 6 7 8 9))
(1 2 3 4 5 6 7 8 9)
> (rotate w 3 8)
(1 2 3 9 5 6 7 8 4)
> w
(1 2 3 4 5 6 7 8 9)
```

Решение 4.225.2 (автор – VH, www.cyberforum.ru)

```
(defun rotate (w a b)
  (when w
    (if (zerop a)
        (if (zerop b)
            w
            ((lambda (result)
               (if (= b 1)
                   (cons (cadr w) result)
                   (cons (car result) (cons (cadr w) (cdr result)))))
             (rotate (cons (car w) (cddr w)) 0 (1- b))))
        (cons (car w) (rotate (cdr w) (1- a) (1- b))))))

> (setf w '(1 2 3 4 5 6 7 8 9))
(1 2 3 4 5 6 7 8 9)
> (rotate w 3 8)
(1 2 3 9 5 6 7 8 4)
> w
```

```
(1 2 3 4 5 6 7 8 9)
```

Решение 4.225.3

```
(defun rotate (w a b)
  (when w
    (if (zerop a)
        (if (zerop b)
            w
            (labels ((next (r)
                       (nconc (if (= b 1)
                                  (list (cadr w) (car r))
                                  (list (car r) (cadr w)))
                              (cdr r))))
              (next (rotate (remove (cadr w) w :count 1) 0 (1- b)))))
        (cons (car w) (rotate (cdr w) (1- a) (1- b))))))
```

```
> (setf w '(1 2 3 4 5 6 7 8 9 10))
(1 2 3 4 5 6 7 8 9 10)
> (rotate w 3 8)
(1 2 3 9 5 6 7 8 4 10)
> w
(1 2 3 4 5 6 7 8 9)
```

Решение 4.225.4

```
(defun rotate (w a b)
  (when w
    (if (zerop a)
        (if (zerop b)
            w
            (labels ((next (r)
                       (nconc (if (= b 1)
                                  `(,(cadr w) ,(car r))
                                  `(,(car r) ,(cadr w)))
                              (cdr r))))
              (next (rotate (remove (cadr w) w :count 1) 0 (1- b)))))
        (cons (car w) (rotate (cdr w) (1- a) (1- b))))))
```

```
> (setf w '(1 2 3 4 5 6 7 8 9 10))
(1 2 3 4 5 6 7 8 9 10)
> (rotate w 3 8)
(1 2 3 9 5 6 7 8 4 10)
> w
(1 2 3 4 5 6 7 8 9 10)
```

Решение 4.225.5

```
(defun rotate (w a b)
  (when w
    (if (zerop a)
        (if (zerop b)
            w
```

```
            (labels ((next (r &aux (v (list (car r) (cadr w))))
                       (nconc (if (= b 1)
                                  (reverse v)
                                  v)
                              (cdr r))))
              (next (rotate (remove (cadr w) w :count 1) 0 (1- b)))))
        (cons (car w) (rotate (cdr w) (1- a) (1- b))))))

> (setf w '(1 2 3 4 5 6 7 8 9 10))
(1 2 3 4 5 6 7 8 9 10)
> (rotate w 3 8)
(1 2 3 9 5 6 7 8 4 10)
> w
(1 2 3 4 5 6 7 8 9 10)
```

Решение 4.225.6

```
(defun rotate (w a b)
  (when w
    (if (zerop a)
        (if (zerop b)
            w
            (labels ((next (r &aux (v `(,(car r) ,(cadr w))))
                       (nconc (if (= b 1) (reverse v) v) (cdr r))))
              (next (rotate (remove (cadr w) w :count 1) 0 (1- b)))))
        (cons (car w) (rotate (cdr w) (1- a) (1- b))))))

> (setf w '(1 2 3 4 5 6 7 8 9 10))
(1 2 3 4 5 6 7 8 9 10)
> (rotate w 3 8)
(1 2 3 9 5 6 7 8 4 10)
> (1 2 3 4 5 6 7 8 9 10)
```

Решение 4.225.7

```
(defun rotate (w a b)
  (when w
    (if (zerop a)
        (if (zerop b)
            w
            (labels ((next (r &aux (v `(,(car r) ,(cadr w))))
                       (nconc (if (= b 1) (reverse v) v) (cdr r))))
              (next (rotate (cons (car w) (cddr w)) 0 (1- b)))))
        (cons (car w) (rotate (cdr w) (1- a) (1- b))))))

> (setf w '(1 2 3 4 5 6 7 8 9 10))
(1 2 3 4 5 6 7 8 9 10)
> (rotate w 3 8)
(1 2 3 9 5 6 7 8 4 10)
> (1 2 3 4 5 6 7 8 9 10)
```

Задача 4.226 nrotate.lisp

Написать функцию, меняющую местами элементы с номерами n и m, разрушающую исходный список w.

Решение 4.226.1

```
(defun nrotate (w n m)
  (progn
    (rotatef (nth n w) (nth m w))
    w))

> (setf w '(1 2 3 4 5 6 7 8 9))
(1 2 3 4 5 6 7 8 9)
> (nrotate w 3 8)
(1 2 3 9 5 6 7 8 4)
> w
(1 2 3 9 5 6 7 8 4)
```

Задача 4.227 drop-center.lisp

Дан список, элементами которого являются списки. Удалить из каждого подсписка элементы, расположенные посередине, но только если длина соответствующего подсписка нечетная.

Решение 4.227.1

```
(defun drop-center (w &aux (a (car w)) (n (length a)))
  (when w (cons (if (zerop (rem n 2))
                    a
                    (let ((m (truncate (/ n 2))))
                      (append (subseq a 0 m) (subseq a (1+ m)))))
                (drop-center (cdr w)))))

> (drop-center '((3 7) (2 7 4) (2 9 5) (7 1) (8 9 2 4 5)))
((3 7) (2 4) (2 5) (7 1) (8 9 4 5))
```

Решение 4.227.2

```
(defun drop-center (w)
  (mapcar
   #'(lambda (a)
       (if (zerop (rem (length a) 2))
           a
           (append (subseq a 0 (truncate (/ (length a) 2)))
                   (subseq a (1+ (truncate (/ (length a) 2)))))))
       w))

> (drop-center '((3 7) (2 7 4) (2 9 5) (7 1) (8 9 2 4 5)))
((3 7) (2 4) (2 5) (7 1) (8 9 4 5))
```

Решение 4.227.3

```
(defun drop-center (w)
  (mapcar
   #'(lambda (a)
       (let* ((n (length a)) (m (truncate (/ n 2))))
         (if (zerop (rem n 2))
             a
             (append (subseq a 0 m)
                     (subseq a (1+ m))))))
       w))

> (drop-center '((3 7) (2 7 4) (2 9 5) (7 1) (8 9 2 4 5)))
((3 7) (2 4) (2 5) (7 1) (8 9 4 5))
```

Задача 4.228 main-dmax.lisp

Дана матрица. Найти в каждой строке наибольший элемент и элемент главной диагонали и поменять их местами.

Решение 4.228.1

```
(defun main-dmax (w &optional (n 0) &aux (a (car w)))
  (when w (cons (progn
                  (rotatef (nth (position (reduce #'max a) a) a)
                           (nth n a))
                  a)
                (main-dmax (cdr w) (1+ n)))))

> (main-dmax '((1 2 3) (4 5 6) (7 8 9)))
((3 2 1) (4 6 5) (7 8 9))
```

Задача 4.229 simplify-n.lisp

Арифметическое выражение имеет форму a + - + + - a - - - + a + + - a где a это произвольное число. Упростите выражение, применяя правила замещения:

```
+ + → +
+ - → -
- + → -
- - → +
```

Решение 4.229.1

```
(defun simplify-n (w &optional b ac &aux (a (car w)))
  (if w
      (simplify-n (cdr w)
                  (cond ((or (and (eq b '+) (eq a '+))
                             (and (eq b '-) (eq a '-))) '+)
                        ((or (and (eq b '+) (eq a '-))
                             (and (eq b '-) (eq a '+))) '-)
                        ((numberp a) nil)
```

```
                    (t a))
                (if (numberp a) (cons a (when b (cons b ac))) ac))
      (reverse ac)))

> (simplify-n '(1 + - + + - 1 - - - + 1 + + - 1))
(1 + 1 - 1 - 1)
```

Задача 4.230 simplify-a.lisp

Арифметическое выражение имеет форму a + - + + - a - - - + a + + - a где a это произвольный атом. Упростите выражение, применяя правила замещения:

```
+ + → +
+ − → −
− + → −
− − → +
```

Решение 4.230.1

```
(defun s (w &optional ac &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((or (eq a '+) (eq a '-)) (s d (cons a ac)))
        ((cons (if (evenp (count '- ac)) '+ '-) (cons a (s d nil))))))

(defun simplify-a (w) (cdr (s w)))

> (simplify-a '(a + - + + - a - - - + a + + - a))
(A + A - A - A)
> (simplify-a '(1 + - + + - 1 - - - + 1 + + - 1))
(1 + 1 - 1 - 1)
```

Задача 4.231 cohere.lisp

Задан список произвольного уровня вложенности, превратить его в список атомов, который будет состоять только из букв, если сумма их кодов превышает сумму цифр исходного.В противном случае список будет состоять только из цифр. В результирующем списке не должно быть повторов.

Решение 4.231.1

```
(defun cohere (w)
  (show (split (flat w))))

(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun split (w &aux n m)
  (dolist (a w) (if (numberp a) (push a n) (push a m)))
  (list n m))
```

```
(defun show (w)
  ((lambda (n m)
     (if (< (reduce #'+ n)
            (reduce #'+
                    (mapcar #'char-code
                            (mapcar #'(lambda (a) (char a 0)) m))))
         (remove-duplicates m :test #'string-equal)
         (remove-duplicates n)))
   (car w) (cadr w)))

> (cohere '(1 ("a" ("b") 300) (1 "A")))
(300 1)
```

Задача 4.232 opposite-word.lisp

Написать функцию, которая на вход, к примеру, получает список ("up" "right" "small"), а на выходе будет ("down" "left" "big").

Решение 4.232.1

```
(defun opposite-word (w)
  (mapcar
   #'(lambda (a)
       (case a
         (down 'up)
         (up 'down)
         (left 'right)
         (right 'left)
         (big 'small)
         (small 'big)
         (otherwise a)))
   w))

> (opposite-word '(up right small))
(DOWN LEFT BIG)
```

Задача 4.233 prefix.lisp

Арифметическое выражение с инфиксной записью и использующем только знаки + и * преобразовать в выражение с префиксной записью.

Решение 4.233.1

```
(defun pre (w &optional ac acc)
  (cond ((null w) (if (cdr ac)
                      (reverse (cons (cons '* (reverse ac)) acc))
                      (reverse (nconc ac acc))))
        ((eq (cadr w) '+) (pre (cddr w)
                               nil
                               (cons (if ac
```

```
                                        (cons '* (reverse
                                                  (cons (car w) ac)))
                                        (car w))
                                    acc)))
        ((pre (cddr w) (cons (car w) ac) acc))))

(defun prefix (w)
  (if (member '+ w) (cons '+ (pre w)) (car (pre w))))

> (prefix '(a + b + c * d * e + f * g))
(+ A B (* C D E) (* F G))
```

Решение 4.233.2

```
(defun pre (w &optional ac acc)
  (cond ((null w) (if (cdr ac)
                      (reverse (cons (cons '* (reverse ac)) acc))
                      (reverse (nconc ac acc))))
        ((eq (cadr w) '+) (pre (cddr w)
                               nil
                               (cons (if ac
                                         (cons '* (reverse
                                                   (cons (car w) ac)))
                                         (car w))
                                     acc)))
        ((pre (cddr w) (cons (car w) ac) acc))))

(defun prefix (w)
  (if (member '+ w) (cons '+ (pre w)) (cons '* (remove '* w))))

> (prefix '(a + b + c * d * e + f * g))
(+ A B (* C D E) (* F G))
```

Решение 4.233.3

```
(defun pre (w &optional ac acc &aux (a (car w)) (d (cddr w)))
  (if w
      (if (eq (cadr w) '+)
          (pre d
               nil
               (cons (if ac
                         (cons '* (reverse (cons a ac)))
                         a)
                     acc))
          (pre d (cons a ac) acc))
      (reverse  (if (cdr ac)
                    (cons (cons '* (reverse ac)) acc)
                    (nconc ac acc)))))

(defun prefix (w)
  (if (member '+ w) (cons '+ (pre w)) (car (pre w))))

> (prefix '(a + b + c * d * e + f * g))
```

```
(+ A B (* C D E) (* F G))
```

Задача 4.234 rotate-core.lisp

Дан список. Переставить в обратном порядке элементы списка, расположенные между его минимальным и максимальным элементами, включая минимальный и максимальный элементы.

Решение 4.234.1

```
(defun rotate-core (w &aux (n (apply #'min w)) (x (apply #'max w))
                    (a (position n w)) (b (position x w)))
 (if (< a b)
     (nconc (subseq w 0 a)
            (reverse (subseq w a (1+ b)))
            (subseq w (1+ b)))
     (nconc (subseq w 0 b)
            (reverse (subseq w b (1+ a)))
            (subseq w (1+ a)))))
```

```
> (rotate-core '(5 6 1 2 3 4 8 2 3))
(5 6 8 4 3 2 1 2 3)
> (rotate-core '(5 6 8 2 3 4 1 2 3))
(5 6 1 4 3 2 8 2 3)
> (rotate-core '(8 8 8 8 8 8 8 8 8))
(8 8 8 8 8 8 8 8 8)
```

Задача 4.235 filter.lisp

Дан список w. Напишите функцию, которая созвращает элементы списка w, удовлетворяющие предикату p.

Решение 4.235.1

```
(defun filter (p w)
  (cond ((null w) nil)
        ((funcall p (car w)) (cons (car w) (filter p (cdr w))))
        (t (filter p (cdr w)))))
```

```
> (filter #'oddp '(1 2 3))
(1 3)
```

Решение 4.235.2

```
(defun filter (p w)
  (when w  (if (funcall p (car w))
               (cons (car w) (filter p (cdr w)))
               (filter p (cdr w)))))
```

```
> (filter #'oddp '(1 2 3))
(1 3)
```

Решение 4.235.3

```
(defun filter (p w)
  (loop for a in w when (funcall p a) collect a))

> (filter #'oddp '(1 2 3))
(1 3)
```

Решение 4.235.4

```
(defun filter (p w)
  (remove-if-not #'(lambda (a) (funcall p a)) w))

> (filter #'oddp '(1 2 3))
(1 3)
```

Задача 4.236 inner-reverse.lisp

Сделать так, чтобы в каждом атоме списка цифры шли в обратном порядке, но буквы при этом оставались не тронуты.

Решение 4.236.1

```
(defun symbols (a)
  (mapcar #'read-from-string (map 'list #'string (string a))))

(defun rev-ns (w v)
  (cond ((null w) nil)
        ((numberp (car w))
         (cons (car v) (rev-ns (cdr w) (cdr v))))
        ((cons (car w) (rev-ns (cdr w) v)))))

(defun reverse-numbers (a &aux (v (symbols a)))
  (read-from-string
   (apply #'concatenate
          'string (mapcar #'write-to-string
                          (rev-ns v (reverse (remove-if-not
                                              #'numberp v)))))))

(defun inner-reverse (w)
  (mapcar #'reverse-numbers w))

> (inner-reverse '(a1b2c a10b23))
(A2B1C A32B01)
```

Задача 4.237 grades.lisp

Определить функцию, которая переводит оценки в баллы и обратно.

Решение 4.237.1

```
(defun grades (w)
  (mapcar
   #'(lambda (a)
       (case a
         (4 'A)
         (3 'B)
         (2 'C)
         (1 'D)
         (0 'F)
         (A 4)
         (B 3)
         (C 2)
         (D 1)
         (F 0)
         (otherwise 'unknown)))
   w))

> (grades '(A B 4 3))
(4 3 A B)
```

Задача 4.238 minusp-to-zero.lisp

Определить функцию, которая заменяет все отрицательные элементы списка значением 0.

Решение 4.238.1

```
(defun minusp-to-zero (w)
  (substitute-if 0 #'minusp w))

> (minusp-to-zero '(-1 0 1))
(0 0 1)
```

Задача 4.239 len&sum.lisp

Файл содержит список целых чисел, разделителем которых служат пробелы. Требуется создать модуль с функций, которая получает имя файла и возвращает список из 2-х элементов: количество чисел в файле и сумма этих чисел.

Решение 4.239.1

```
numbers.in:
11 11 11 11
12 12 12 12

(defun len&sum (&optional (path-in "d:/numbers.in"))
  (let ((u)) (with-open-file (s path-in :direction :input)
               (do ((c (read-line s nil :eof) (read-line s nil :eof)))
                   ((eql c :eof))
```

```
                (setq u (nconc (list-nums c) u))))
       (values (length u) (reduce #'+ u))))

(defun list-nums (s)
  (read-from-string (concatenate 'string "(" s ")")))

> (len&sum "d:/numbers.in")
8
92
> (len&sum)
8
92
```

Решение 4.239.2

```
numbers.in:
11 11 11 11
12 12 12 12

(defun len&sum (path-in)
  (let ((u)) (with-open-file (s path-in :direction :input)
               (do ((c (read-line s nil :eof) (read-line s nil :eof)))
                   ((eql c :eof))
                 (setq u (nconc (list-nums c) u))))
       (list (length u) (reduce #'+ u))))

(defun list-nums (s)
  (read-from-string (concatenate 'string "(" s ")")))

> (len&sum "d:/numbers.in")
(8 92)
```

Решение 4.239.3

```
numbers.in:
11 11 11 11
12 12 12 12

(defun len&sum (path-in &aux u)
  (with-open-file (s path-in :direction :input)
    (do ((c (read-line s nil :eof) (read-line s nil :eof)))
        ((eql c :eof))
      (setq u (nconc (list-nums c) u))))
  (list (length u) (reduce #'+ u)))

(defun list-nums (s)
  (read-from-string (concatenate 'string "(" s ")")))

> (len&sum "d:/numbers.in")
(8 92)
```

Решение 4.239.4

```
numbers.in:
11 11 11 11
12 12 12 12

(defun len&sum (path-in &aux u)
  (with-open-file (s path-in :direction :input)
    (do ((c (read-line s nil :eof) (read-line s nil :eof)))
        ((eql c :eof))
      (setq u (nconc (read-from-string
                      (concatenate 'string "(" c ")")) u))))
  (list (length u) (reduce #'+ u)))

(defun list-nums (s)
  (read-from-string (concatenate 'string "(" s ")")))

> (len&sum "d:/numbers.in")
(8 92)
```

Задача 4.240 co-insiders.lisp

Из числового списка построить список чисел, совпадающих со своими индексами.

Решение 4.240.1

```
(defun co-inciders (w)
  (loop for a in w
        for b upfrom 0
        when (= a b) collect a))

> (co-inciders '(0 1 1 3))
(0 1 3)
```

Задача 4.241 neg.lisp

Определите функцию, которая из одноуровневого числового списка создает новый список, меняя знак у каждого атома.

Решение 4.241.1

```
(defun neg (w)
  (cond ((null w) nil)
        (t (cons (- (car w)) (neg (cdr w))))))

> (neg '(0 -1 1))
(0 1 -1)
```

Решение 4.241.2

```
(defun neg (w)
  (when w (cons (- (car w)) (neg (cdr w)))))
```

```
> (neg '(0 -1 1))
(0 1 -1)
```

Решение 4.241.2

```
(defun neg (w)
  (mapcar #'- w))

> (neg '(0 -1 1))
(0 1 -1)
```

Задача 4.242 overs-n.lisp

Определить функцию, которая возвращает числа из одноуровневого числового списка w больше, чем n.

Решение 4.242.1

```
(defun overs-n (n w)
  (cond ((null w) nil)
        ((> (car w) n) (cons (car w) (overs-n n (cdr w))))
        (t (overs-n n (cdr w)))))

> (overs-n 3 '(1 2 3 4 5 6 7))
(4 5 6 7)
```

Решение 4.242.2

```
(defun overs-n (n w)
  (when w
    (if (> (car w) n)
        (cons (car w) (overs-n n (cdr w)))
        (overs-n n (cdr w)))))

> (overs-n 3 '(1 2 3 4 5 6 7))
(4 5 6 7)
```

Решение 4.242.3

```
(defun overs-n (n w)
  (loop for a in w when (> a n) collect a))

> (overs-n 3 '(1 2 3 4 5 6 7))
(4 5 6 7)
```

Решение 4.242.4

```
(defun overs-n (n w)
  (remove-if-not #'(lambda (a) (> a n)) w))

> (overs-n 3 '(1 2 3 4 5 6 7))
```

```
(4 5 6 7)
```

Задача 4.243 backward.lisp

Определить функцию, которая которая по линейныму списку вида (a b c d e ...) строит сложный список вида (a (b (c (d (e (...)))))).

Решение 4.243.1

```
(defun backward (w)
  (cond ((null (cdr w)) w)
        ((cons (car w) (list (backward (cdr w)))))))

> (backward '(a b c d e))
(A (B (C (D (E)))))
```

Решение 4.243.2

```
(defun backward (w)
  (if (cdr w) (cons (car w) (list (backward (cdr w)))) w))

> (backward '(a b c d e))
(A (B (C (D (E)))))
```

Задача 4.244 evens.lisp

Определить функцию, которая из линейного числового списка выбирает все четные числа.

Решение 4.244.1

```
(defun evens (w)
  (cond ((null w) nil)
        ((evenp (car w)) (cons (car w) (evens (cdr w))))
        (t (evens (cdr w)))))

> (evens '(0 1 2 3 4 5))
(0 2 4)
```

Решение 4.244.2

```
(defun evens (w)
  (when w (if (evenp (car w))
              (cons (car w) (evens (cdr w)))
              (evens (cdr w)))))

> (evens '(0 1 2 3 4 5))
(0 2 4)
```

Решение 4.244.3

```
(defun evens (w)
  (loop for a in w when (evenp a) collect a))

> (evens '(0 1 2 3 4 5))
(0 2 4)
```

Решение 4.244.4

```
(defun evens (w)
  (remove-if-not #'evenp w))

> (evens '(0 1 2 3 4 5))
(0 2 4)
```

Задача 4.245 del-thirds.lisp

Составить функцию с одним аргументом – сложным многоуровневым списком, которая превращала бы этот список таким образом, чтобы был удален каждый третий элемент во всех подсписках любого уровня вложенности.

Решение 4.245.1

```
(defun del-thirds (w)
  (cond ((atom w) w)
        (t (cons (del-thirds(car w))
                 (if (cadr w)
                     (cons (del-thirds (cadr w))
                           (del-thirds (cdddr w)))
                     nil)))))

> (del-thirds '(1 2 3 (4 5 6)))
(1 2 (4 5))
```

Решение 4.245.2

```
(defun del-thirds (w)
  (cond ((atom w) w)
        (t (cons (del-thirds(car w))
                 (when (cadr w)
                   (cons (del-thirds (cadr w))
                         (del-thirds (cdddr w))))))))

> (del-thirds '(1 2 3 (4 5 6)))
(1 2 (4 5))
```

Решение 4.245.3

```
(defun del-thirds (w)
  (if (atom w)
      w
      (cons (del-thirds(car w))
            (when (cadr w)
```

```
              (cons (del-thirds (cadr w))
                    (del-thirds (cdddr w)))))))

> (del-thirds '(1 2 3 (4 5 6)))
(1 2 (4 5))
```

Задача 4.246 make-assoc.lisp

Дан список ((a b) (b c) (c d) ...). Превратить его в ассоциативной список. В качестве ключей взять первые элементы вложенных списков. В качестве значений - вторые.

Решение 4.246.1

```
(defun make-assoc (w)
  (loop for (a b) in w collect (cons a b)))

> (make-assoc '((a 1) (b 2) (c 3)))
((A . 1) (B . 2) (C . 3))
```

Решение 4.246.2

```
(defun make-assoc (w)
  (when w (cons
           (apply #'cons (car w))
           (make-assoc (cdr w)))))

> (make-assoc '((a 1) (b 2) (c 3)))
((A . 1) (B . 2) (C . 3))
```

Задача 4.247 num-eventh.lisp

Определить функцию, которая перед каждым четным элементом списка вставляет его порядковый номер. Пример: (a b c d ...) ==> (a 2 b c 4 d ...).

Решение 4.247.1

```
(defun num-eventh (w &optional (n 1))
  (cond ((null w) nil)
        ((evenp n) (cons n (cons (car w)
                                 (num-eventh (cdr w) (1+ n)))))
        ((cons (car w) (num-eventh (cdr w) (1+ n))))))

> (num-eventh '(a b c d e))
(A 2 B C 4 D E)
```

Решение 4.247.2

```
defun num-eventh (w &optional (n 1))
  (when  w (if (evenp n)
```

```
               (cons n (cons (car w) (num-eventh (cdr w) (1+ n))))
               (cons (car w) (num-eventh (cdr w) (1+ n))))))

> (num-eventh '(a b c d e))
(A 2 B C 4 D E)
```

Решение 4.247.3

```
(defun num-eventh (w &optional (n 1))
  (when  w (if (evenp n)
               (nconc `(,n ,(car w)) (num-eventh (cdr w) (1+ n)))
               (cons (car w) (num-eventh (cdr w) (1+ n))))))

> (num-eventh '(a b c d e))
(A 2 B C 4 D E)
```

Решение 4.247.4

```
(defun num-eventh (w &optional (n 1))
  (when  w  (nconc (if (evenp n)`(,n ,(car w)) `(,(car w)))
                   (num-eventh (cdr w) (1+ n)))))

> (num-eventh '(a b c d e))
(A 2 B C 4 D E)
```

Задача 4.248 sum-pairs.lisp

Определить функцию, которая возвращает список сумм двух соседних элементов линейного числового списка четной длины.

Решение 4.248.1

```
(defun sum-pairs (w)
  (cond ((null w) nil)
        (t (cons (+ (car w) (cadr w)) (sum-pairs (cddr w))))))

> (sum-pairs '(1 2 3 4 5 6 7 8))
(3 7 11 15)
```

Решение 4.248.2

```
(defun sum-pairs (w)
  (when w (cons (+ (car w) (cadr w)) (sum-pairs (cddr w)))))

> (sum-pairs '(1 2 3 4 5 6 7 8))
(3 7 11 15)
```

Задача 4.249 sub-lengths.lisp

Дан линейный список списков, которые имеют произвольную длину. определить функциию, который считает длину каждого из вложенных списков. Ре-

зультат вернуть в виде линейного списка.

Решение 4.249.1

```
(defun sub-lengths (w)
  (cond ((null w) nil)
        (t (cons (length (car w)) (sub-lengths (cdr w))))))

> (sub-lengths '((a b) (c d e f) (g h i)))
(2 4 3)
```

Решение 4.249.2

```
(defun sub-lengths (w)
  (when w (cons (length (car w)) (sub-lengths (cdr w)))))

> (sub-lengths '((a b) (c d e f) (g h i)))
(2 4 3)
```

Решение 4.249.3

```
(defun sub-lengths (w) (mapcar #'length w))

> (sub-lengths '((a b) (c d e f) (g h i)))
(2 4 3)
```

Задача 4.250 sub-lengths.lisp

Определить функцю, которая каждые два соседних элемента списка превращает в двухэлементный подсписок.

Решение 4.250.1

```
(defun make-pairs (w)
  (when w (cons (list (car w) (cadr w)) (make-pairs (cddr w)))))

> (make-pairs '(a b c d e f))
((A B) (C D) (E F))
> (make-pairs '(a b c d e))
((A B) (C D) (E NIL))
```

Задача 4.251 abstract-sum.lisp

Дан список двухэлементных числовых списков произвольной длины. Определить функцию, которая считает разность двухэлементных списков с нечетными номерами и сумму - с нечетными.

Решение 4.251.1

```
(defun abstract-sum (w &optional (p #'-))
  (when w
```

```
    (cons (apply p (car w))
          (abstract-sum (cdr w)
                        (if (equalp p #'-)
                            #'+
                            #'-)))))

> (abstract-sum '((5 3) (8 6) (2 9)))
(2 14 -7)
```

Решение 4.251.2

```
(defun abstract-sum (w &optional f)
  (when w
    (cons (apply (if f #'+ #'-) (car w))
          (abstract-sum (cdr w) (not f)))))

> (abstract-sum '((5 3) (8 6) (2 9)))
(2 14 -7)
```

Задача 4.252 insert-nums.lisp

Определить функцию, которая после каждого элемента списка вставляет его порядковый номер.

Решение 4.252.1

```
(defun insert-nums (w &optional (n 1))
  (cond ((null w) nil)
        ((cons (car w) (cons n (insert-nums (cdr w) (1+ n)))))))

> (insert-nums '(a b c d))
(A 1 B 2 C 3 D 4)
```

Решение 4.252.2

```
(defun insert-nums (w &optional (n 1))
  (when w (cons (car w) (cons n (insert-nums (cdr w) (1+ n))))))

> (insert-nums '(a b c d))
(A 1 B 2 C 3 D 4)
```

Решение 4.252.3

```
(defun insert-nums (w &aux (n 0))
  (mapcan #'(lambda (a) (list a (incf n))) w))

> (insert-nums '(a b c d))
(A 1 B 2 C 3 D 4)
```

Решение 4.252.4

```
(defun insert-nums (w)
```

```
  (loop for a in w
        for n upfrom 1
        nconc (list a n)))

> (insert-nums '(a b c d))
(A 1 B 2 C 3 D 4)
```

Решение 4.252.5

```
(defun insert-nums (w)
  (mapcan #'list w (loop for a from 1 to (length w)
                         collect a)))

> (insert-nums '(a b c d))
(A 1 B 2 C 3 D 4)
```

Задача 4.253 lets-marks.lisp

Определить функцию, которая по заданному тексту, представленному списком символов, строит список пар: (<знак> <частота встречаемости в тексте>), а также выводит число точек, восклицательных знаков, вопросительных знаков и пробелов.

Решение 4.253.1

```
(defun lets-marks (w &aux (v (mapcan
                               #'(lambda (a)
                                   (concatenate 'list
                                                (string a)))
                               w)))
  (values (mapcar #'(lambda (a) (list a (count a v)))
                  (remove-duplicates v))
          (+ (reduce #'+ (mapcar #'(lambda (a) (list a (count a v)))
                                 '(#\. #\! #\?)) :key #'cadr)
             (1- (length w)))))

> (lets-marks '(aaa. bbb. ccc!))
((#\A 3) (#\B 3) (#\. 2) (#\C 3) (#\! 1))
5
```

Решение 4.253.2

```
(defun lets-marks (w &aux (v (mapcan
                               #'(lambda (a)
                                   (concatenate 'list
                                                (string a)))
                               w)))
  (values (mapcar #'(lambda (a) (list a (count a v)))
                  (remove-duplicates v))
          (+ (count-if #'(lambda (a) (or (eq a #\.)
                                         (eq a #\!)
                                         (eq a #\?)))
```

```
                        v)
             (1- (length w)))))

> (lets-marks '(aaa. bbb. ccc!))
((#\A 3) (#\B 3) (#\. 2) (#\C 3) (#\! 1))
5
```

Решение 4.253.3

```
(defun lets-marks (w &aux (v (mapcan
                               #'(lambda (a)
                                   (concatenate 'list
                                                (string a)))
                               w)))
  (values
   (loop for a in (remove-duplicates v)
         collect (list a (count a v)))
   (+ (count-if #'(lambda (a) (or (eq a #\.)
                                  (eq a #\!)
                                  (eq a #\?)))
                v)
      (1- (length w)))))

> (lets-marks '(aaa. bbb. ccc!))
((#\A 3) (#\B 3) (#\. 2) (#\C 3) (#\! 1))
5
```

Решение 4.253.3

```
(defun lets-marks (w &aux (v (mapcan
                               #'(lambda (a)
                                   (concatenate 'list
                                                (string a)))
                               w)))
  (values
   (loop for a in (remove-duplicates v)
         collect (list a (count a v)))
   (+ (count #\. v) (count #\! v) (count #\? v)
      (1- (length w)))))

> (lets-marks '(aaa. bbb. ccc!))
((#\A 3) (#\B 3) (#\. 2) (#\C 3) (#\! 1))
5
```

Задача 4.254 1-2-2-1.lisp

Дан список произвольной длины. Определить функцию, которая возвращает первый, второй, предпоследний и последний элементы этого списка.

Решение 4.254.1

```
(defun 1-2-2-1 (w)
```

```
  (cons (car w) (cons (cadr w) (last w 2))))

> (1-2-2-1 '(1 2 3 4 5 6 7))
(1 2 6 7)
> (1-2-2-1 '(1 2))
(1 2 1 2)
> (1-2-2-1 '(1))
(1 NIL 1)
> (1-2-2-1 '())
(NIL NIL)
```

Решение 4.254.2

```
(defun 1-2-2-1 (w)
  (when w (cons (car w) (if (cadr w)
                            (cons (cadr w)
                                  (last w 2))
                            (last w 2)))))

> (1-2-2-1 '(1 2 3))
(1 2 2 3)
> (1-2-2-1 '(1 2))
(1 2 1 2)
> (1-2-2-1 '(1))
(1 1)
> (1-2-2-1 '())
NIL
```

Задача 4.255 if-intp-abs-min-max-list-average.lisp

Определить функцию, которая принимает пять чисел. Найти минимальное и максимальное по модулю и если они целые то сформировать из этих двух чисел список, иначе подсчитать среднее арифметическое пяти чисед.

Решение 4.255.1

```
(defun if-intp-abs-min-max-list-average
    (&rest w &aux (v (mapcar #'abs w))
     (n (apply #'min v)) (x (apply #'max v)))
  (if (and (integerp n) (integerp x))
      (list n x)
      (/ (apply #'+ w) (length w))))

> (if-intp-abs-min-max-list-average  1 2 3 4 -5)
(1 5)
> (if-intp-abs-min-max-list-average  1 2 3 4 -5.5)
0.9
> (if-intp-abs-min-max-list-average  1 2 3 4 5.5)
3.1
```

Задача 4.256 class.lisp

Определить структуру списка студентов – имя, адрес, средний балл.

Решение 4.256.1

```
(defstruct sum name address average)

(defun res (n)
  (when (> n 0)
    (cons (make-sum :name (read) :address (read) :average (read))
          (res (1- n)))))

> (setf class (res 2))
ivanov
pushkina-10
4
petrov
gogolya-20
5
(#S(SUM :NAME IVANOV :ADDRESS PUSHKINA-10 :AVERAGE 4)
 #S(SUM :NAME PETROV :ADDRESS GOGOLYA-20 :AVERAGE 5))
```

Задача 4.257 rev-ev-lev.lisp

Создать функцию, которая делает реверс элементов во всех подсписках четных уровней, не затрагивая нечетные уровни. Принимает в качестве своего аргумента многоуровневый список, например (a b (c d (e f (g h)) i) (j (k (l m n)))) и возвращающую список (a b (i (e f (h g)) d c) ((k (n m l)) j)).

Решение 4.257.1

```
(defun rev-ev-lev (w &optional (v 1))
  (when w (mapcar
           #'(lambda (a)
               (if (listp a)
                   (rev-ev-lev a (1+ v))
                   a))
           (if (evenp v) (reverse w) w))))

> (rev-ev-lev '(a b (c d (e f (g h)) i) (j (k (l m n)))))
(A B (I (E F (H G)) D C) ((K (N M L)) J))
```

Решение 4.257.2

```
(defun rev-ev-lev (w &optional f)
  (when w (mapcar
           #'(lambda (a)
               (if (listp a)
                   (rev-ev-lev a (not f))
                   a))
```

```
           (if f (reverse w) w))))

> (rev-ev-lev '(a b (c d (e f (g h)) i) (j (k (l m n)))))
(A B (I (E F (H G)) D C) ((K (N M L)) J))
```

Решение 4.257.3

```
(defun rev-ev-lev (w &optional f)
  (when w (mapcar
           #'(lambda (a)
               (if (atom a) a (rev-ev-lev a (not f))))
               (if f (reverse w) w))))

> (rev-ev-lev '(a b (c d (e f (g h)) i) (j (k (l m n)))))
(A B (I (E F (H G)) D C) ((K (N M L)) J))
```

Задача 4.258 list-elm-num.lisp

Дан список. Определить функцию, возвращающую список списков, состоящих из элемента исходного списка, знака минус и порядкового номера элемента.

Решение 4.258.1

```
(defun list-elm-num (w &optional (n 1))
  (cond ((null w) nil)
        (t (cons (list (car w)'- n)
                 (list-elm-num (cdr w) (1+ n))))))

> (list-elm-num '(a b c d))
((A - 1) (B - 2) (C - 3) (D - 4))
```

Решение 4.258.2

```
(defun list-elm-num (w &optional (n 1))
  (when w (cons (list (car w)'- n)
                (list-elm-num (cdr w) (1+ n)))))

> (list-elm-num '(a b c d))
((A - 1) (B - 2) (C - 3) (D - 4))
```

Решение 4.258.3

```
(defun list-elm-num (w)
  (loop for a in w
        for b from 1
        collect (list a'- b)))

> (list-elm-num '(a b c d))
((A - 1) (B - 2) (C - 3) (D - 4))
```

Задача 4.259 lessen.lisp

Определить функцию, которая преобразует список целых чисел в список, каждый из элементов которого на единицу меньше соответствующего элемента.

Решение 4.259.1

```
(defun lessen (w)
  (cond ((null w) nil)
        ((cons (1- (car w)) (lessen (cdr w))))))

> (lessen '(1 2 3 4 5))
(0 1 2 3 4)
```

Решение 4.259.2

```
(defun lessen (w)
  (if w (cons (1- (car w)) (lessen (cdr w))) nil))

> (lessen '(1 2 3 4 5))
(0 1 2 3 4)
```

Решение 4.259.3

```
(defun lessen (w)
  (when w (cons (1- (car w)) (lessen (cdr w)))))

> (lessen '(1 2 3 4 5))
(0 1 2 3 4)
```

Решение 4.259.4

```
(defun lessen (w) (mapcar #'1- w))

> (lessen '(1 2 3 4 5))
(0 1 2 3 4)
```

Задача 4.260 drop-second.lisp

Из заданного списка удалить каждый второй элемент.

Решение 4.260.1

```
(defun drop-second (w)
  (cond ((null  w) nil)
        (t (cons (car w) (drop-second (cddr w))))))

> (drop-second '(1 2 3 4 5))
(1 3 5)
```

Решение 4.260.2

```
(defun drop-second (w)
  (if w (cons (car w) (drop-second (cddr w))) nil))

> (drop-second '(1 2 3 4 5))
(1 3 5)

Решение 4.260.3

(defun drop-second (w)
  (when w (cons (car w) (drop-second (cddr w)))))

> (drop-second '(1 2 3 4 5))
(1 3 5)

Решение 4.260.4

(defun drop-second (w)
  (loop for a in w by #'cddr collect a))

> (drop-second '(1 2 3 4 5))
(1 3 5)
```

Задача 4.261 flat-pair.lisp

```
Дан многоуровневый список. Определить функцию, группирующщую элементы
попарно

Решение 4.261.1

(defun flat (w &optional ac)
  (cond ((null w) ac)
        ((atom w) (cons w ac))
        ((flat (car w) (flat (cdr w) ac)))))

(defun pair (w)
  (if (cdr w)
      (cons (list (car w) (cadr w))
            (pair (cddr w)))
      (when w (list w))))

(defun flat-pair (w)
  (pair (flat w)))

> (flat-pair '(a b ((c d e) f g (h i)) j k))
((A B) (C D) (E F) (G H) (I J) (K))
> (flat-pair '(a b ((c d e) f g (h i)) j k l))
((A B) (C D) (E F) (G H) (I J) (K L))
```

Задача 4.262 minus-plus.lisp

Для заданного числового списка построить новый список, в котором сначала мдут все отрицательные элементы, а затем все неотрицательные.

Решение 4.262.1

```
(defun minus-plus (w &optional minus plus &aux (a (car w)))
  (cond ((null w) (nconc minus plus))
        ((minusp a) (minus-plus (cdr w) (cons a minus) plus))
        ((minus-plus (cdr w) minus (cons a plus)))))

> (minus-plus '(3 -7 0 5 -4))
(-4 -7 5 0 3)
```

Решение 4.262.2

```
(defun minus-plus (w)
  (loop for i in w
        if (minusp i)
        collect i into minus
        else collect i into plus
        finally (return (nconc minus plus))))

> (minus-plus '(3 -7 0 5 -4))
(-7 -4 3 0 5)
```

Решение 4.262.3

```
(defun minus-plus (w) (sort w #'<))

> (minus-plus '(3 -7 0 5 -4))
(-7 -4 0 3 5)
```

*Задача 4.263 odd*2-even*3.lisp*

Для заданного числового списка построить новый список, в котором сначала мдут все отрицательные элементы, а затем все неотрицательные.

Решение 4.263.1

```
(defun odd*2-even*3 (w)
  (when w (cons (* (car w) (if (oddp (car w)) 2 3))
                (odd*2-even*3 (cdr w)))))

> (odd*2-even*3 '(3 -7 0 5 -4))
(6 -14 0 10 -12)
```

Решение 4.263.2

```
(defun odd*2-even*3 (w &aux (a (car w)))
  (when w (cons (* a (if (oddp a) 2 3))
                (odd*2-even*3 (cdr w)))))
```

```
> (odd*2-even*3 '(3 -7 0 5 -4))
(6 -14 0 10 -12)

Решение 4.263.3

(defun odd*2-even*3 (w)
  (mapcar #'(lambda (a) (* a (if (oddp a) 2 3))) w))

> (odd*2-even*3 '(3 -7 0 5 -4))
(6 -14 0 10 -12)

Решение 4.263.4

(defun odd*2-even*3 (w)
  (loop for a in w collect (* a (if (oddp a) 2 3))))

> (odd*2-even*3 '(3 -7 0 5 -4))
(6 -14 0 10 -12)
```

Задача 4.264 oddth/3.lisp

Дан числовой список. Определить функцию, возвращающую список мз уменьшенных в три раза элементов исходного списка, которые имеют нечетные порядковые номера.

```
Решение 4.264.1

(defun oddth/3 (w)
  (cond ((null w) nil)
        (t (cons (/ (car w) 3) (oddth/3 (cddr w))))))

> (oddth/3 '(7 1 2 13 24))
(7/3 2/3 8)

Решение 4.264.2

(defun oddth/3 (w)
  (if w (cons (/ (car w) 3) (oddth/3 (cddr w))) nil))

> (oddth/3 '(7 1 2 13 24))
(7/3 2/3 8)

Решение 4.264.3

(defun oddth/3 (w)
  (when w (cons (/ (car w) 3) (oddth/3 (cddr w)))))

> (oddth/3 '(7 1 2 13 24))
(7/3 2/3 8)

Решение 4.264.4
```

```
(defun oddth/3 (w)
  (loop for a in w by #'cddr collect (/ a 3)))

> (oddth/3 '(7 1 2 13 24))
(7/3 2/3 8)
```

Задача 4.265 oddth/3.lisp

Дан одноуровневый список. Продублировать каждый элемент списка спомощью спискообразующих функций.

Решение 4.265.1

```
(defun twin-elm (w)
  (cond ((null w) nil)
        (t (cons (car w) (cons (car w) (twin-elm (cdr w)))))))

> (twin-elm '(a b c))
(A A B B C C)
```

Решение 4.265.2

```
(defun twin-elm (w)
  (when w (cons (car w) (cons (car w) (twin-elm (cdr w))))))

> (twin-elm '(a b c))
(A A B B C C)
```

Задача 4.266 plus-exp.lisp

Дан одноуровневый числовой список. Все положительные элементы списка возвести в квадрат.

Решение 4.266.1

```
(defun plus-exp (w)
  (cond ((null w) nil)
        ((plusp (car w)) (cons (expt (car w) 2)
                               (plus-exp (cdr w))))
        (t (plus-exp (cdr w)))))

> (plus-exp '(-2 2))
(-2 4)
```

Решение 4.266.2

```
(defun plus-exp (w)
  (when w (cons (if (plusp (car w)) (* (car w) (car w)) (car w))
                (plus-exp (cdr w)))))

> (plus-exp '(-2 2))
```

```
(-2 4)
```

Решение 4.266.3

```
(defun plus-exp (w)
  (when w (cons (* (car w) (if (plusp (car w)) (car w) 1))
                (plus-exp (cdr w)))))

> (plus-exp '(-2 2))
(-2 4)
```

Решение 4.266.4

```
(defun plus-exp (w)
  (mapcar #'(lambda (a) (if (plusp a) (* a a) a)) w))

> (plus-exp '(-2 2))
(-2 4)
```

Решение 4.266.5

```
(defun plus-exp (w)
  (loop for a in w collect (if (plusp a) (* a a) a)))

> (plus-exp '(-2 2))
(-2 4)
```

Решение 4.266.6

```
(defun plus-exp (w)
  (cond ((null w) nil)
        ((plusp (car w)) (cons (expt (car w) 2) (plus-exp (cdr w))))
        (t (cons (car w) (plus-exp (cdr w))))))

> (plus-exp '(-2 2))
(-2 4)
```

В общем виде:

Решение 4.266.7

```
(defun plus-exp (w p n)
  (cond ((null w) nil)
        ((funcall p (car w)) (cons (expt (car w) n)
                                   (plus-exp (cdr w) p n)))
        (t (cons (car w) (plus-exp (cdr w) p n)))))

> (plus-exp '(-2 2) #'plusp 2)
(-2 4)
```

Задача 4.267 snap.lisp

В каждой строке заданной матрицы вычислить сумму, количество и средне арифметическое положительных элементов.

Решение 4.267.1

```
(defun snap (w
             &aux
             (a (remove-if-not #'plusp (car w)))
             (m (length a))
             (s (reduce #'+ a)))
  (when w (cons (list s m (when (plusp m) (/ s m)))
                (snap (cdr w)))))

> (snap '((1 2 3 -1 -2 -3) (0 0 5 6 -7 -8) (0 0 0 -7 -9)))
((6 3 2) (11 2 11/2) (0 0 NIL))
```

Решение 4.267.2

```
(defun snap (w
             &aux
             (a (remove-if-not #'plusp (car w)))
             (m (length a))
             (s (reduce #'+ a)))
  (when w (cons `(,s ,m ,(when (plusp m) (/ s m)))
                (snap (cdr w)))))

> (snap '((1 2 3 -1 -2 -3) (0 0 5 6 -7 -8) (0 0 0 -7 -9)))
((6 3 2) (11 2 11/2) (0 0 NIL))
```

Решение 4.267.3

```
(defun snap (w
             &aux
             (a (remove-if-not #'plusp (car w)))
             (m (length a))
             (s (reduce #'+ a)))
  (when w (cons `(,s ,m ,(unless (zerop m) (/ s m)))
                (snap (cdr w)))))

> (snap '((1 2 3 -1 -2 -3) (0 0 5 6 -7 -8) (0 0 0 -7 -9)))
((6 3 2) (11 2 11/2) (0 0 NIL))
```

Задача 4.268 plus-minus.lisp

Определить функцию, переставляющую все отрицательные элементы числового списка в конец списка.

Решение 4.268.1

```
(defun plus-minus (w &optional acc)
  (cond ((null w) (reverse acc))
        ((minusp (car w)) (plus-minus (cdr w) (cons (car w) acc)))
```

```
        (t (cons (car w) (plus-minus (cdr w) acc)))))

> (plus-minus '(1 -2 3 -4 -7 4 5))
(1 3 4 5 -2 -4 -7)
```

Решение 4.268.2

```
(defun plus-minus (w)
  (loop for a in w
        if (minusp a)
        collect a into minus
        else collect a into plus
        finally (return (nconc plus minus))))

> (plus-minus '(1 -2 3 -4 -7 4 5))
(1 3 4 5 -2 -4 -7)
```

Задача 4.269 all-zero.lisp

Определить функцию, переставляющую все нулевые элементы числового списка в конец списка.

Решение 4.269.1

```
(defun all-zero (w &optional acc)
  (cond ((null w) (reverse acc))
        ((zerop (car w)) (all-zero (cdr w) (cons (car w) acc)))
        (t (cons (car w) (all-zero (cdr w) acc)))))

> (all-zero '(1 0 3 0 0 4 5))
(1 3 4 5 0 0 0)
```

Решение 4.269.2

```
(defun all-zero (w)
  (loop for a in w
        if (zerop a)
        collect a into zero
        else collect a into all
        finally (return (nconc all zero))))

> (all-zero '(1 0 3 0 0 4 5))
(1 3 4 5 0 0 0)
```

Задача 4.270 let>1.lisp

Определить функцию, которая каждую букву в списке '(1 (2 a) b 14 1 b) заменит цифрой 1.

Решение 4.270.1

```
(defun let>1 (w)
  (cond ((null w) nil)
        ((listp (car w)) (cons (let>1 (car w)) (let>1 (cdr w))))
        ((symbolp (car w)) (cons 1 (let>1 (cdr w))))
        (t (cons (car w) (let>1 (cdr w))))))

> (let>1 '(1 (2 a) b 14 1 b))
(1 (2 1) 1 14 1 1)
```

Решение 4.270.2

```
(defun let>1 (w)
  (when w (cons (cond ((listp (car w)) (let>1 (car w)))
                      ((symbolp (car w)) 1)
                      (t (car w)))
                (let>1 (cdr w)))))

> (let>1 '(1 (2 a) b 14 1 b))
(1 (2 1) 1 14 1 1)
```

Решение 4.270.3

```
(defun let>1 (w)
  (loop for a in w
        if (listp a) collect (let>1 a)
        else if (symbolp a) collect 1
        else collect a))

> (let>1 '(1 (2 a) b 14 1 b))
(1 (2 1) 1 14 1 1)
```

Задача 4.271 dig-over.lisp

Определить функцию, которая все цифры каждого элемента списка увеличивает на 1, а 9 заменяет на 0.

Решение 4.271.1

```
(defun dig-over (w)
  (mapcar
   #'(lambda (a)
       (parse-integer
        (apply #'concatenate
         'string
         (loop for e across (write-to-string a)
               collect (write-to-string
                        (over
                         (parse-integer
                          (string e))))))))
   w))

(defun over (n)
```

```
  (if (eq n 9) 0 (1+ n)))

> (dig-over '(123 567 896 878 102))
(234 678 907 989 213)
```

Решение 4.271.2

```
(defun dig-over (w)
  (mapcar
   #'(lambda (a)
       (parse-integer
        (apply #'concatenate
               'string
               (mapcar
                #'(lambda (e)
                    (write-to-string
                     (over
                      (parse-integer e))))
                (map 'list
                     #'string
                     (write-to-string a))))))
   w))

(defun over (n)
  (case n
    (0 1)
    (1 2)
    (2 3)
    (3 4)
    (4 5)
    (5 6)
    (6 7)
    (7 8)
    (8 9)
    (9 0)))

> (dig-over '(123 567 896 878 102))
(234 678 907 989 213)
```

Решение 4.271.3

```
(defun dig-over (w)
  (mapcar
   #'(lambda (a)
       (parse-integer
        (apply #'concatenate
               'string
               (mapcar
                #'over
                (map 'list
                     #'string
                     (write-to-string a))))))
   w))
```

```
(defun over (n)
  (cond ((equal n "0") "1")
        ((equal n "1") "2")
        ((equal n "2") "3")
        ((equal n "3") "4")
        ((equal n "4") "5")
        ((equal n "5") "6")
        ((equal n "6") "7")
        ((equal n "7") "8")
        ((equal n "8") "9")
        (t "0")))

> (dig-over '(123 567 896 878 102))
(234 678 907 989 213)
```

Решение 4.271.4

```
(defun dig-over (w)
  (mapcar
   #'(lambda (a)
       (parse-integer
        (apply #'concatenate
               'string
               (loop for e across (write-to-string a)
                     collect (over (string e))))))
   w))

(defun over (n)
  (cond ((equal n "0") "1")
        ((equal n "1") "2")
        ((equal n "2") "3")
        ((equal n "3") "4")
        ((equal n "4") "5")
        ((equal n "5") "6")
        ((equal n "6") "7")
        ((equal n "7") "8")
        ((equal n "8") "9")
        (t "0")))

> (dig-over '(123 567 896 878 102))
(234 678 907 989 213)
```

Задача 4.272 sieve-of-eratosthenes.lisp

Определить функцию нахождения всех простых чисел из заданного набора методом «решето Эратосфена»: задан список n целых чисел начиная с 2. Процедура нахождения простых чисел заключается в последовательной проверкt на делимость на каждое последовательно взятое оставшееся первое число всех оставшихся чисел списка и "вычёркивания" всех делящихся.

Решение 4.272.1

```
(defun sieve-of-eratosthenes (w &optional acc)
  (cond ((null w) (reverse acc))
        ((sieve-of-eratosthenes
          (remove-if #'(lambda (a) (zerop (rem a (car w)))) w)
          (cons (car w) acc)))))

> (sieve-of-eratosthenes (loop for a from 2 to 100 collect a))
(2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89
97)
```

Решение 4.272.2

```
(defun sieve-of-eratosthenes (w &optional acc &aux (a (car w)))
  (if w
      (sieve-of-eratosthenes
       (remove-if #'(lambda (e) (zerop (rem e a))) w)
       (cons a acc))
      (reverse acc)))

> (sieve-of-eratosthenes (loop for a from 2 to 100 collect a))
(2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89
97)
```

Решение 4.272.3

```
(defun sieve-of-eratosthenes (n m)
  (reverse (sieve (loop for a from n to m collect a))))

(defun sieve (w &optional acc &aux (a (car w)))
  (if w
      (sieve
       (delete-if #'(lambda (e) (zerop (rem e a))) w)
       (cons a acc))
      acc))

> (sieve-of-eratosthenes 2 100)
(2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83 89
97)
```

Задача 4.273 elm-occur.lisp

Определить функцию, преобразующую список w в новый список, элементы которого имеют вид: (<элемент списка w> <кол-во вхождений этого элемента в список w>).

Решение 4.273.1

```
(defun elm-occur (w)
  (loop for a in (remove-duplicates w)
        collect (list a (count a w))))
```

```
> (elm-occur '(a b a a c b))
((A 3) (C 1) (B 2))
```

Решение 4.273.2

```
(defun elm-occur (w)
  (mapcar #'(lambda (a) (list a (count a w)))
          (remove-duplicates w)))

> (elm-occur '(a b a a c b))
((A 3) (C 1) (B 2))
```

Решение 4.273.3

```
(defun elm-occur (w)
  (loop for a in (remove-duplicates w)
        collect `(,a ,(count a w))))

> (elm-occur '(a b a a c b))
((A 3) (C 1) (B 2))
```

Решение 4.273.4

```
(defun elm-occur (w)
  (mapcar #'(lambda (a) `(,a ,(count a w)))
          (remove-duplicates w)))

> (elm-occur '(a b a a c b))
((A 3) (C 1) (B 2))
```

Задача 4.274 prefix.lisp

Определить функцию, вычисляющую формулу с полной скобочной структурой.

Решение 4.274.1

```
(defun prefix (w)
  (if (atom w) w `(,(cadr w) ,(prefix (car w)) ,(prefix (caddr w)))))

> (prefix '((6 + (7 - 8)) * 6))
(* (+ 6 (- 7 8)) 6)
> (eval (prefix '((6 + (7 - 8)) * 6)))
30
> (prefix '((-8 + 10) * (1 + 2)))
(* (+ -8 10) (+ 1 2))
> (eval (prefix '((-8 + 10) * (1 + 2))))
6
```

Задача 4.275 atom-level.lisp

Определить функцию, вычисляющую формулу с полной скоочной структурой.

Решение 4.275.1

```
(defun atom-level (w &optional acc (n -1))
  (cond ((null w) acc)
        ((atom w) (cons (list w n) acc))
        ((atom-level (car w) (atom-level (cdr w) acc n) (1+ n)))))

> (atom-level '(a s (d f (g h (j) q) w) r))
((A 0) (S 0) (D 1) (F 1) (G 2) (H 2) (J 3) (Q 2) (W 1) (R 0))
```

Решение 4.275.2

```
(defun atom-level (w &optional acc (n -1))
  (cond ((null w) acc)
        ((atom w) (cons `(,w ,n) acc))
        ((atom-level (car w)
                     (atom-level (cdr w) acc n)
                     (1+ n)))))

> (atom-level '(a s (d f (g h (j) q) w) r))
((A 0) (S 0) (D 1) (F 1) (G 2) (H 2) (J 3) (Q 2) (W 1) (R 0))
```

Задача 4.276 matrix-max-min.lisp

Числовая матрица задана списком, в котором каждый элемент представляет собой подсписок, содержащий строку матрицы. Найти максимальный и минимальный элементы матрицы.

Решение 4.276.1

```
(defun matrix-max-min (w &aux (v (flat w)))
  (values (reduce #'max v) (reduce #'min v)))

(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

> (matrix-max-min '((1 2) (3 4) (5 6)))
6
1
```

Решение 4.276.2

```
(defun matrix-max-min (w)
  (values
   (loop for a in w maximizing (maxi a))
   (loop for a in w minimizing (mini a))))

(defun maxi (w)
```

```
  (loop for a in w maximizing a))

(defun mini (w)
  (loop for a in w minimizing a))

> (matrix-max-min '((1 2) (3 4) (5 6)))
6
1
```

Решение 4.276.3

```
(defun matrix-max-min (w)
  (values (edge w #'max) (edge w #'min)))

(defun edge (w p)
  (reduce p (mapcar #'(lambda (a) (reduce p a)) w)))

> (matrix-max-min '((1 2) (3 4) (5 6)))
6
1
```

Решение 4.276.4

```
(defun matrix-max-min (w)
  (values (edge w #'max) (edge w #'min)))

(defun edge (w p)
  (reduce p (apply #'mapcar (cons p w))))

> (matrix-max-min '((1 2) (3 4) (5 6)))
6
1
```

Решение 4.276.5

```
(defun matrix-max-min (w &aux (v (reduce #'nconc w)))
  (values (reduce #'max v) (reduce #'min v)))

> (matrix-max-min '((1 2) (3 4) (5 6)))
6
1
```

Задача 4.277 max-ab.lisp

В квадратной матрице, заданной списком строк, найти максимальный элемент и его координаты.

Решение 4.277.1

```
(defun max-ab (w
               &aux
               (a (length (car w)))
```

```
               (v (reduce #'nconc w))
               (m (reduce #'max v))
               (p (position m v)))
  `(,m ,(floor p a) ,(mod p a)))

> (max-ab '((1 2 3) (15 2 6) (7 8 9)))
(15 1 0)
```

Решение 4.277.2

```
(defun max-ab (w
               &aux
               (a (length (car w)))
               (v (reduce #'nconc w))
               (m (reduce #'max v))
               (p (position m v)))
  (values m (floor p a) (mod p a)))

> (max-ab '((1 2 3) (15 2 6) (7 8 9)))
15
1
0
```

Задача 4.278 max-min-elms.lisp

Дан список, состоящий из подсписков. Построить список, элементами которого являются наибольшие и наименьшие элементы подсписков данного списка, собранные в подсписки.

Решение 4.278.1

```
(defun edge (w p)
  (mapcar #'(lambda (a) (reduce p a)) w))

(defun max-min-elms (w)
  (values (edge w #'max) (edge w #'min)))

> (max-min-elms '((1 2 3) (4 5 6) (7 8 9)))
(3 6 9)
(1 4 7)
```

Задача 4.279 atom-sublist.lisp

Создать рекурсивную функцию, которая считает сколько в списке атомов и сколько подсписков.

Решение 4.279.1

```
(defun atom-sublist (w &optional (a 0) (s 0))
  (cond ((null w) (list a s))
        ((atom (car w)) (atom-sublist (cdr w)(1+ a) s))
```

```
        (t (atom-sublist (cdr w) a (1+ s)))))

> (atom-sublist '(a '(b) c '(d) e))
(3 2)
```

Задача 4.280 missed-nums.lisp

Определить функцию, возвращающую числа, пропущенные в списке идущих подряд целых чисел.

Решение 4.280.1

```
(defun missed-nums (w &optional (n (car w)))
  (cond ((null w) nil)
        ((= (car w) n) (missed-nums (cdr w) (1+ n)))
        (t (cons n (missed-nums w (1+ n))))))

> (missed-nums '(2 4 6 8 10))
(3 5 7 9)
> (missed-nums '(2 10))
(3 4 5 6 7 8 9)
```

Решение 4.280.2

```
(defun missed-nums (w &optional (n (car w)))
  (when w (if (= (car w) n)
              (missed-nums (cdr w) (1+ n))
              (cons n (missed-nums w (1+ n))))))

> (missed-nums '(2 4 6 8 10))
(3 5 7 9)
> (missed-nums '(2 10))
(3 4 5 6 7 8 9)
```

Решение 4.280.3

```
(defun missed-nums (w &optional acc (n (car w)))
  (cond ((null w) (reverse acc))
        ((= (car w) n) (missed-nums (cdr w) acc (1+ n)))
        (t (missed-nums w (cons n acc) (1+ n)))))

> (missed-nums '(2 4 6 8 10))
(3 5 7 9)
> (missed-nums '(2 10))
(3 4 5 6 7 8 9)
```

Решение 4.280.4

```
(defun missed-nums (w &aux (a (car w)) (b (car (last w))))
  (nset-difference (loop for n from a to b collect n) w))

> (missed-nums '(2 4 6 8 10))
```

```
(3 5 7 9)
> (missed-nums '(2 10))
(3 4 5 6 7 8 9)
```

Структура 5 (функция ATOM ATOM) > ATOM

Задача 5.1 average.lisp

Определить функцию, которая возвращает среднее арифметическое двух положительных чисел.

Решение 5.1.1

```
(defun average (a b)
  "Returns the average of the numbers a and b."
  ;; Doesn't round or truncate integers
  (float (/ (+ a b) 2)))

> (average 2 3)
2.5
```

Задача 5.2 div-abs.lisp

Решить задачу через хвостовую рекурсию: деление нацело через вычитание.

Решение 5.2.1

```
(defun div-abs (m n)
  (div m n 0))

(defun div (m n c)
  (cond ((< m n) c)
        ((div (- m n) n (1+ c)))))

> (div-abs 27 6)
4
```

Задача 5.3 string-right-n.lisp

Дана строка. Определить функцию, которая возвращает строку из n последних знаков исходной строки.

Решение 5.3.1

```
(defun string-right (s n)
  (subseq s (- (length s) n)))

> (string-right "abcde" 3)
```

"cde"

Задача 5.4 concord-mask-string.lisp

Определить функцию для сопоставления введенного шаблона (маски) и строки (слова). Под маской, в данном случае, подразумевается комбинация известных букв слова и знаков вопроса вместо неизвестных букв. Для замены неизвестного количества неизвестных букв используется знак звездочка *. Функция с двумя переменными: первая - собственно, шаблон, вторая - строка для проверки соответствия. Шаблон строится по следующим правилам: символ * заменяет любые символы, символ ? заменяет только один любой символ, количество * и ? в шаблоне не ограничено. Например, при введенном шаблоне L?*S**P строка LISP будет ему соответствовать.

Решение 5.4.1

```
(defun mems (s p)
  (member (car s) (cdr p) :test #'equal))

(defun check (p s)
  (cond ((null s) t)
        ((or (equal (car p) (car s))
             (equal (car p) "?"))
         (check (cdr p) (cdr s)))
        ((when (equal (car p) "*")
               (if (mems s p)
                   (if (equal (car s) (cadr s))
                       (check (cdr (mems s p)) (cddr s))
                       (check (cdr (mems s p)) (cdr s)))
                   (check p (cdr s)))))))

(defun squeeze (p &optional z)
  (when p (if (equal (car p) "*")
              (if z
                  (squeeze (cdr p) z)
                  (cons "*" (squeeze (cdr p) t)))
              (cons (car p) (squeeze (cdr p) nil)))))

(defun cleave (s)
  (map 'list #'string s))

(defun concord-mask-string (p s)
  (when (> (length s) 0) (check (squeeze (cleave p)) (cleave s))))

> (concord-mask-string "L?*S**P" "LISP")
T
> (concord-mask-string "L?*S**P" "LCISP")
T
> (concord-mask-string "L?*S**P" "LISRP")
T
> (concord-mask-string "L?*S**P" "LLISP")
```

```
T
> (concord-mask-string "L?*S**P" "LISPP")
T
> (concord-mask-string "L?*S**P" "CLISP")
NIL
> (concord-mask-string "L?*S**P" "LISPC")
NIL
> (concord-mask-string "L?*S**P" "")
NIL
```

Задача 5.5 disk-area.lisp

Написать функцию, которая считает площадь круга, если известны внутренний и внешний радиусы.

Решение 5.5.1

```
(defun disk-area (r2 r1)
  (- (* pi r2 r2) (* pi r1 r1)))

> (disk-area 3 2)
15.707963267948966193L0
```

Задача 5.6 solid-torus.lisp

Написать функцию, которая считает объем, ограниченный внутри тора, если известны внутренний и внешний радиусы.

Решение 5.6.1

```
(defun solid-torus (r2 r1)
  (* pi pi r1 (expt (- r2 r1) 2) 1/2))

> (solid-torus 3 2)
9.869604401089358619L0
```

Задача 5.7 a+a[a+1]+a*[a+1]*[a+2+...+a*[a+1]*...*[a+1].lisp*

Определите функцию (a+a*[a+1]+a*[a+1]*[a+2+...+a*[a+1]*...*[a+1] a n), которая от двух числовых аргументов вычисляет величину:
a + a * (a + 1) + a * (a + 1) * (a + 2) +...+ a * (a + 1) *...* (a + n).

Решение 5.7.1

```
(defun az (a n)
  (cond ((zerop n) a)
        ((* (+ a n) (az a (1- n))))))

(defun a+a*[a+1]+a*[a+1]*[a+2+...+a*[a+1]*...*[a+1] (a n)
  (cond ((zerop n) a)
```

```
        ((+ (a+a*[a+1]+a*[a+1]*[a+2+...+a*[a+1]*...*[a+1]
            a (1- n)) (az a n)))))

> (a+a*[a+1]+a*[a+1]*[a+2+...+a*[a+1]*...*[a+1] 1 2)
9
> (a+a*[a+1]+a*[a+1]*[a+2+...+a*[a+1]*...*[a+1] 1 3)
33
```

Решение 5.7.2

```
(defun az (a n)
  (if (zerop n) a (* (+ a n) (az a (1- n)))))

(defun bx (a n)
  (if (zerop n) a (+ (bx a (1- n)) (az a n))))

> (bx 1 3)
33
```

Решение 5.7.3 (автор – VH, www.hardforum.ru)

```
defun bx (a n)
  (if (zerop n) a (* a (1+ (bx (1+ a) (1- n))))))

> (bx 1 3)
33
```

Задача 5.8 insert-lets.lisp

Написать программу обработки текста естественного языка с использованием отображающих функционалов. Дан текст. В каждом слове вставить после заданного 3-буквенного сочетания заданное 2-буквенное.

Решение 5.8.1*

```
(defun insert-lets (s p r &aux (e (search p s)))
  (cond ((null e) (concatenate 'string nil s))
        ((concatenate 'string
                      (subseq s 0 e)
                      p
                      r
                      (insert-lets (subseq s (+ e 3)) p r)))))

> (insert-lets "abcde abcde abcde" "bcd" "zz")
"abcdzze abcdzze abcdzze"
```

По функциям:

```
> (concatenate 'string "a" "b" "c")
"abc"
> (subseq '(a b c d) 1 3)
(B C)
> (subseq "abcd" 1 3)
```

```
"bc"
> (subseq "abcd" 1)
"bcd"
```

Задача 5.9 calculate.lisp

Написать функцию, которая принимает ввод двух чисел и операции между ними и возвращает результат.

Решение 5.9.1

```
(defun  calculate (&aux cmd)
   (loop (print "Enter: Number Operator Number")
         (setq cmd (read-from-string
                     (concatenate 'string "(" (read-line) ")")))
         (when (eql (car cmd) 'quit) (return 'ok))
         (print (eval `(,(cadr cmd) ,(car cmd) ,(caddr cmd))))))

> (calculate)
"Enter: Number Operator Number" 2 + 3
5
"Enter: Number Operator Number" quit
OK
```

Решение 5.9.2

```
(defun  calculate (&aux cmd)
   (loop (print '>)
         (setq cmd (read-from-string
                     (concatenate 'string "(" (read-line) ")")))
         (when (eql (car cmd) 'quit) (return 'ok))
         (print (eval `(,(cadr cmd) ,(car cmd) ,(caddr cmd))))))

> (calculate)
> 2 + 3
5
> quit
OK
```

Задача 5.10 greatest-common-divisor.lisp

Определить функцию, вычисляющую наибольший общий делитель двух чисел - алгоритм Эвклида (Eukléides, около 365–300 г. до н. э.).

Решение 5.10.1

```
(defun greatest-common-divisor (a b)
  (if (zerop (rem a b)) b (greatest-common-divisor b (rem a b))))

> (greatest-common-divisor 70 105)
35
```

```
> (greatest-common-divisor 32453 345)
23
```

Решение 5.10.2

```
(defun greatest-common-divisor (a b)
  (if (zerop (mod a b)) b (greatest-common-divisor b (mod a b))))

> (greatest-common-divisor 70 105)
35
> (greatest-common-divisor 32453 345)
23
```

Решение 5.10.3

```
(defun greatest-common-divisor (a b)
  (if (zerop b) a (greatest-common-divisor b (mod a b))))

> (greatest-common-divisor 70 105)
35
> (greatest-common-divisor 32453 345)
23
```

Решение 5.10.4

```
(defun greatest-common-divisor (a b)
  (gcd a b))

> (greatest-common-divisor 70 105)
35
> (greatest-common-divisor 32453 345)
23
```

Задача 5.11 replace-vowels.lisp

Дана строка текста. Определить функцию, которая заменяет все гласные буквы на букву «z»

Решение 5.11.1

```
(defun replace-vowels (s r &optional (w '("a" "e" "i" "o" "u")))
  (if w (replace-vowels (replace-lets s (car w) r) r (cdr w)) s))

(defun replace-lets (s p r &aux (e (search p s)))
  (if e
      (concatenate
       'string
       (subseq s 0 e)
       r
       (replace-lets (subseq s (1+ e)) p r))
      s))
```

```
> (replace-vowels "aberikodubra" "z")
"zbzrzkzdzbrz"
```

Задача 5.12 co-prime.lisp

Написать функцию (предикат), определяющую, являются ли ее аргументы взаимно простыми числами.*

* Обычно число зубьев на звёздочках и число звеньев цепи в цепной передаче стремятся делать взаимно простыми, чтобы обеспечить равномерность износа: каждый зуб звёздочки будет поочередно работать со всеми звеньями цепи.

Решение 5.12.1

```
(defun co-prime (a b) (= (gcd a b) 1))

> (co-prime 15 25)
NIL
> (co-prime 14 25)
T
```

Решение 5.12.2

```
(defun co-prime (&rest w)
  (= (apply #'gcd w) 1))

> (co-prime 10 15 25)
NIL
> (co-prime 10 15 27)
T
```

Задача 5.13 sum-cubes.lisp

Написать функцию, вычисляющую сумму кубов чисел от n до m включительно.

Решение 5.13.1

```
(defun sum-cubes (n m)
  (if (> n m) 0 (+ (expt n 3) (sum-cubes (1+ n) m))))

> (sum-cubes 2 5)
224
```

Задача 5.14 sum-expt-n-m.lisp

Написать функцию, вычисляющую сумму кубов чисел от n до m включительно.

Решение 5.14.1

```
(defun sum-expt-n-m (n m)
  (reduce #'+ (loop for a from 1 to n collect (expt a m))))

> (sum-expt-n-m 10 2)
385
```

Задача 5.15 div.lisp

Определить функцию деления нацело через вычитание.

Решение 5.15.1

```
(defun div (a b)
  (cond ((zerop b) nil)
        ((< a b) 0)
        ((> a b) (1+ (div (- a b) b)))))

> (div 10 3)
3
```

Решение 5.15.2

```
(defun div (a b)
  (unless (zerop b) (if (< a b)
                        0
                        (1+ (div (- a b) b)))))

> (div 10 3)
3
```

Задача 5.16 -expt.lisp

Определить функцию возведения в степень.

Решение 5.16.1

```
(defun -expt (m n)
  (if (= n 1)
      m
      (* m (-expt m (1- n)))))

> (-expt 2 3)
8
```

Функция -expt является рекурсивной, то есть вызывает себя до тех пор, пока ее аргумент n (счетчик) не уменьшится до 1 (то есть n – 1 раз), после чего функция вернет значение m. При каждой передаче значения m на предыдущий уровень, результат умножается на m. Так m окажется перемноженным на себя n раз.

```
Решение 5.16.2

(defun -expt (n m)
  (reduce #'* (loop repeat m collect n)))

> (-expt 2 3)
8
```

Задача 5.17 add.lisp

```
Определить функцию, которая складывает два числа.

Решение 5.17.1

(defun dec (a) (- a 1))

(defun inc (a) (+ a 1))

(defun add (a b)
  (if (= a 0) b (+ (dec a) (inc b))))

> (add 2 3)
5

Решение 5.17.2

(defun dec (a) (- a 1))

(defun inc (a) (+ a 1))

(defun add (a b)
  (if (= a 0) b (inc (+ (dec a) b))))

> (add 2 3)
5
```

Задача 5.18 money-lost.lisp

```
Эта задача для второго класса церковно приходской школы была придумана
Львом Толстым. Продавец продает шапку, которая стоит 10 р. Подходит
покупатель, меряет и согласен взять, но у него есть только 25 р. Про-
давец отсылает мальчика с этими 25 р. к соседке, разменять. Мальчик
прибегает и отдает 10+10+5. Продавец отдает шапку и сдачу в 15 руб.
Через какое то время приходит соседка и говорит что 25 р. фальшивые,
требует отдать ей деньги. Что делать? Мужик лезет в кассу и возвращает
ей деньги. Вопрос: На сколько обманули продавца?

Решение 5.18.1

(defun money-lost (hat fake)
  (- fake hat (- fake hat) fake))
```

```
> (money-lost 10 25)
-25
```

Решение 5.18.2

Стоимость шапки можно не брать в расчет:

```
(defun money-lost (hat fake)
  (- (- fake fake) (- hat hat) fake))

> (money-lost 20 25)
-25
```

или

Решение 5.18.1

```
(defun money-lost (fake)
  (- fake))

> (money-lost 25)
-25
```

Можно по разному комбинировать числа в скобках, но иногда решающий учитывает лишь отдачу -10 (шапка покупателю), -15 (сдача покупателю), -25 (возврат соседке) и не учитывает настоящие деньги, полученные от соседки +25. Продавец получит +25 и вернет соселке полученные от нее настоящие деньги -25 и выйдет в ноль, по этому направлению не фомируются потери, а все его потери в направлении покупателя -25 (стоимость шапки -10 и сдача покупателю -15 или фальшивые деньги -25.

Или в столбик:
- 10 ушла шапка к покупателю
- 15 ушла сдача к покупателю
+ 25 пришли настоящие деньги от соседки
- 25 ушли настоящие деньги к соседке
Итого:
- 25

Структура 6 (функция ATOM ATOM) > (СПИСОК)

Задача 6.1 same-words.lisp

Даны два предложения. Вывести слова, которые входят в каждое из них, т.е. совпадающие.

Решение 6.1.1

```
(defun sw (s)
```

```
  (read-from-string (concatenate 'string "(" s ")")))

(defun same-words (a b)
  (intersection (sw a) (sw b)))

> (same-words "a b c" "b c d")
(B C)
```

Задача 6.2 quotient&remainder.lisp

Определите функцию, которая возвращает целое и остаток от деления положительных чисел. Не использовать функции деления и остатка.

Решение 6.2.1 (автор – VH, www.cyberforum.ru)

```
(defun quotient&remainder (x y)
  (cond ((zerop y) nil)
        ((< x y) (cons 0 x))
        (t ((lambda (res)
              (cons (1+ (car res)) (cdr res)))
            (quotient&remainder (- x y) y)))))

> (quotient&remainder 5 3)
(1 . 2)
```

Задача 6.3 random-row.lisp

Создайте функцию, порождающую по заданным числам m, k список, состоящий из случайного количества случайных чисел из промежутка от m до k.

Решение 6.3.1

```
(defun random-row (m k &optional (n (random 25)) &aux (a (random k)))
  (cond ((zerop n) nil)
        ((> a m) (cons a (random-row m k (1- n))))
        ((random-row m k n))))

> (random-row 40 50)
(49 48 48 48 46 49)
> (random-row 40 50)
(44 48 43 43 41 48 48 46 47 43 49 43 46 48 42 47 47 44 48)
> (random-row 40 50)
(48 45 46 48 46 43 45 47 42 42 49 48 42)
```

Задача 6.4 onion.lisp

Определите функцию, которая строит список "луковица" с уровнем вложенности n для параметра a. Например, при n=4, a=0 функция должна возвращать список ((((0)))).

Решение 6.4.1

```
(defun onion (a n)
  (if (zerop n) a (list (onion a (1- n)))))

> (onion 0 4)
((((0))))
```

Решение 6.4.2

```
(defun onion (a n)
  (loop repeat n
        for e = (list a) then (list e)
        finally (return e)))

> (onion 0 4)
((((0))))
```

Задача 6.5 primes.lisp

Определить функцию, которая принимает числа n и m и составляет список простых чисел в диапазоне от n до m.

Решение 6.5.1

```
(defun check (x)
  (when (> x 1) (loop for i from 2 to (isqrt x)
                      never (zerop (mod x i)))))

(defun primes (n m &optional w)
  (cond ((= n m) (nreverse w))
        ((or (= n 2) (check n)) (primes (1+ n) m (cons n w)))
        ((primes (1+ n) m w))))

> (primes 1 30)
(2 3 5 7 11 13 17 19 23 29)
> (primes 6 30)
(7 11 13 17 19 23 29)
```

Решение 6.5.2

```
(defun primes(x n &optional w)
  (cond ((= x n) (nreverse w))
        ((or (= x 2) (when (> x 1)
                       (loop for i from 2 to (isqrt x)
                             never (zerop (mod x i)))))
         (primes (1+ x) n (cons x w)))
        ((primes (1+ x) n w))))

> (primes 1 30)
(2 3 5 7 11 13 17 19 23 29)
> (primes 6 30)
```

```
(7 11 13 17 19 23 29)
```

Решение 6.5.3

```
(defun check (n)
  (when (> n 1) (loop for a from 2 to (isqrt n)
                      never (zerop (mod n a)))))

(defun nums(n m)
  (loop for a from n to m
        when (or (= a 2) (check a)) collect a))

> (primes 1 30)
(2 3 5 7 11 13 17 19 23 29)
> (primes 6 30)
(7 11 13 17 19 23 29)
```

Задача 6.6 numbers.lisp

Определить функцию, которая порождает все целые числа, отвечающие условию a <= x <= b.

Решение 6.6.1

```
(defun numbers (a b)
  (when (<= a b) (cons a (numbers (1+ a) b))))

> (numbers 4 8)
(4 5 6 7 8)
```

Решение 6.6.2

```
(defun numbers (a b)
  (loop for x from a to b collect x))

> (numbers 4 8)
(4 5 6 7 8)
```

Задача 6.7 count-dictionary.lisp

Написать программу вывода словарных слов в порядке их встречаемости в заданном тексте. Вывести частоту встречаемости словарных слов в тексте в виде в виде десятичной дроби. Текст и словарные слова вводить из отдельных файлов.

Решение 6.7.1

```
d:/text.txt:
aaaa bbbb cccc dddd
eeee ffff gggg aaaa
bbbb hhhh iiii aaaa
```

```
aaaa bbbb cccc

d:/dict.txt:
aaaa bbbb
cccc

(defun sw (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun count-dictionary (pt pd &aux text dict)
  (with-open-file (s pd :direction :input)
    (do ((line (read-line s nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
      (setf dict (nconc dict (sw line)))))
  (with-open-file (s pt :direction :input)
    (do ((line (read-line s nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
      (setf text (nconc text (sw line)))))
  (let ((m (length text)))
    (loop for a in dict
          collect (list a (float (/ (count a text) m))))))

> (count-dictionary "d:/text.txt" "d:/dict.txt")
((AAAA 0.26666668) (BBBB 0.2) (CCCC 0.13333334))
```

Решение 6.7.2

```
d:/text.txt:
aaaa bbbb cccc dddd
eeee ffff gggg aaaa
bbbb hhhh iiii aaaa
aaaa bbbb cccc aaa1
bbb2 ccc1 aaa2
d:/dict.txt:
aaaa aaa1 aaa2
bbbb bbb1 bbb2
cccc ccc1 ccc2

(defun sw (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun count-dictionary (pt pd &aux text dict)
  (with-open-file (s pd :direction :input)
    (do ((line (read-line s nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
      (setf dict (cons (sw line) dict))))
  (with-open-file (s pt :direction :input)
    (do ((line (read-line s nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
      (setf text (nconc text (sw line)))))
  (let ((m (length text)))
    (loop for a in (reverse dict)
          collect (list (car a)
```

```
                        (float (/ (loop for e in a
                                        sum (count e text))
                                  m))))))

> (count-dictionary "d:/text.txt" "d:/dict.txt")
((AAAA 0.31578946) (BBBB 0.21052632) (CCCC 0.15789473))
```

Решение 6.7.3 * с выводом содержания словаря и текста

```
d:/text.txt:
aaaa bbbb cccc dddd
eeee ffff gggg aaaa
bbbb hhhh iiii aaaa
aaaa bbbb cccc aaa1
bbb2 ccc1 aaa2

d:/dict.txt:
aaaa aaa1 aaa2
bbbb bbb1 bbb2
cccc ccc1 ccc2

defun sw (s)
  (read-from-string (concatenate 'string "(" s ")")))

(defun count-dictionary (pt pd &aux text dict)
  (with-open-file (s pd :direction :input)
    (do ((line (read-line s nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
      (setf dict (cons (sw line) dict))))
  (with-open-file (s pt :direction :input)
    (do ((line (read-line s nil :eof)(read-line s nil :eof)))
        ((eql line :eof))
      (setf text (nconc text (sw line)))))
  (print dict)
  (print text)
  (let ((m (length text)))
    (loop for a in (reverse dict)
          collect (list (car a)
                        (float (/ (loop for e in a
                                        sum (count e text))
                                  m))))))

> (count-dictionary "d:/text.txt" "d:/dict.txt")
((CCCC CCC1 CCC2) (BBBB BBB1 BBB2) (AAAA AAA1 AAA2))
(AAAA BBBB CCCC DDDD EEEE FFFF GGGG AAAA BBBB HHHH IIII AAAA AAAA BBBB
CCCC AAA1 BBB2 CCC1 AAA2)
((CCCC 0.15789473) (BBBB 0.21052632) (AAAA 0.31578946))
```

Задача 6.8 max-divided.lisp

Определить функцию, возвращающую числа из диапазона n m, имеющие наибольшее количество делителей.

Решение 6.8.1

```
(defun num-divisors (n)
  (loop for a from 2 to (/ n 2) when (zerop (rem n a)) count a))

(defun divided (n m ac &aux (z (num-divisors n)))
  (cond ((= n m) ac)
        ((> z (cadar ac)) (divided (1+ n) m `((,n ,z))))
        ((= z (cadar ac)) (divided (1+ n) m (cons `(,n ,z) ac)))
        ((divided (1+ n) m ac))))

(defun max-divided (n m)
  (divided (1+ n) m `((,n ,(num-divisors n)))))

> (max-divided 2 100)
((96 10) (90 10) (84 10) (72 10) (60 10))
> (max-divided 2 1000)
((840 30))
> (max-divided 2 10000)
((9240 62) (7560 62))
```

Решение 6.8.2

```
(defun num-divisors (n)
  (loop for a from 2 to (/ n 2) when (zerop (rem n a)) count a))

(defun divided (n m ac &aux (z (num-divisors n)))
  (if (= n m)
      ac
      (divided (1+ n)
               m
               (cond ((> z (cadar ac)) `((,n ,z)))
                     ((= z (cadar ac)) (cons `(,n ,z) ac))
                     (t ac)))))

(defun max-divided (n m)
  (divided (1+ n) m `((,n ,(num-divisors n)))))

> (max-divided 2 100)
((96 10) (90 10) (84 10) (72 10) (60 10))
> (max-divided 2 1000)
((840 30))
> (max-divided 2 10000)
((9240 62) (7560 62))
```

Решение 6.8.3

```
(defun num-divisors (n)
  (loop for a from 2 to (/ n 2) when (zerop (rem n a)) count a))

(defun divided (n m ac &aux (z (num-divisors n)) (d (cadar ac)))
  (if (= n m)
```

```
      (values (mapcar #'car ac) d)
      (divided (1+ n) m (cond ((> z d) `((,n ,z)))
                              ((= z d) (cons `(,n ,z) ac))
                              (t ac)))))

(defun max-divided (n m)
  (divided (1+ n) m `((,n ,(num-divisors n)))))

> (max-divided 2 100)
(96 90 84 72 60)
10
> (max-divided 2 1000)
(840)
30
> (max-divided 2 10000)
(9240 7560)
62
```

Задача 6.9 generate.lisp

Определить функцию, которая возвращает список-«маску» заданной длины.

Решение 6.9.1

```
(defun generate (n a)
  (cond ((zerop n) nil)
        (t (cons a (generate (1- n) a)))))

> (loaf 'x 4)
(X X X X)
```

Решение 6.9.2

```
(defun generate (n a)
  (if (zerop n) nil (cons a (generate (1- n) a))))

> (loaf 'x 4)
(X X X X)
```

Решение 6.9.3

```
(defun generate (n a)
  (when (plusp n) (cons a (generate (1- n) a))))

> (generate 5 'x)
(X X X X X)
```

Решение 6.9.4

```
(defun generate (n a)
  (make-list n :initial-element a))
```

```
R> (generate 5 'x)
(X X X X X)
```

Решение 6.9.5

```
(defun generate (n a)
  (loop for b from 1 to n collect a))

> (generate 5 'x)
(X X X X X)
```

Решение 6.9.6

```
(defun generate (n a)
  (loop repeat n collect a))

> (generate 5 'x)
(X X X X X)
```

Задача 6.10 onion-right.lisp

Определить фнкцию, которая по заданному уровню вложенности n генериру-
ет список вида (a (a (a (a (a (a)))))).

Решение 6.10.1

```
(defun onion-right (a n)
  (if (= n 1) (list a) (list a (onion-right a (1- n)))))

> (onion-right 'a 7)
(A (A (A (A (A (A (A)))))))
```

Решение 6.10.2

```
(defun onion-right (a n)
  (if (= n 1) `(,a) `(,a ,(onion-right a (1- n)))))

> (onion-right 'a 7)
(A (A (A (A (A (A (A)))))))
```

Задача 6.11 rand-add-pair.lisp

Построить два списка из пяти случайных чисел и получить список, попар-
но суммируя элементы полученных списков.

Решение 6.11.1

```
(defun rand (r n)
  (loop for i from 1 to n
        collect (random r)))
```

```
(defun rand-add-pair (n m)
  (mapcar #'+ (rand n m) (rand n m)))

> (rand-add-pair 100 5)
(74 59 123 85 29)
> (rand-add-pair 100 5)
(132 35 97 46 118)
```

Структура 7 (функция (СПИСОК) АТОМ) > АТОМ

Задача 7.1 elms-num.lisp

```
Определить функцию, подсчитывающую число вхождений эллемента в список.

Решение 7.1.1

defun elms-num (n w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 0)
        ((equal n a) (1+ (elms-num n d)))
        ((elms-num n d))))

> (elms-num 2 '(1 2 2 4 2 3))
3

Решение 7.1.2

(defun elms-num (n w)
  (loop for a in w count (equal n a)))

> (elms-num 2 '(1 2 2 4 2 3))
3

Решение 7.1.3

(defun elms-num (n w) (count n w))

> (elms-num 2 '(1 2 2 4 2 3))
3
```

Задача 7.2 check-number.lisp

```
Проверить числовой список на наличие числа.

Решение 7.2.1

(defun check-number (n w)
  (cond ((null w) nil)
        ((equal n (car w)) t)
        ((check-number n (cdr w)))))
```

```
> (check-number 3 '(1 2 3 4 5))
T
> (check-number 0 '(1 2 3 4 5))
NIL
```

Решение 7.2.2

```
(defun check-number (n w)
  (some #'(lambda (a) (eq n a)) w))

> (check-number 3 '(1 2 3 4 5))
T
> (check-number 0 '(1 2 3 4 5))
NIL
```

Решение 7.2.3

```
(defun check-number (n w)
  (loop for i in w thereis (eq n i)))

> (check-number 3 '(1 2 3 4 5))
T
> (check-number 0 '(1 2 3 4 5))
NIL
```

Решение 7.2.4

```
(defun check-number (n w)
  (member n w))

> (check-number 3 '(1 2 3 4 5))
(3 4 5)
> (check-number 0 '(1 2 3 4 5))
NIL
```

Задача 7.3 insider.lisp

Проверить, входит ли заданный элемент в многоуровневый список.

Решение 7.3.1

```
(defun insider (n w)
  (when w ((lambda (a d)
             (cond ((null w) nil)
                   ((listp a) (insider n a))
                   ((eql n a) t)
                   ((insider n d))))
           (car w) (cdr w))))

> (insider 0 '(1(2(4 5)3(6))))
NIL
> (insider 4 '(1(2(4 5)3(6))))
```

```
T
```

Задача 7.4 near-last.lisp

Написать функцию, которая возвращает предпоследний элемент списка.

Решение 7.4.1

```
(defun near-last (w)
  (car (last (nbutlast w))))

> (near-last '(1 2 3 4 5 6))
5
```

Решение 7.4.2

```
(defun near-last (w)
  (cadr (nreverse w)))

> (near-last '(1 2 3 4 5 6))
5
```

Задача 7.5 value.lisp

Прайс компьютерного магазина описывается списком (наименование фирма параметры цена): ((Терминал Acer 15 4500) (Терминал Sumsung 17 7000)(Матплата Asus 238 2500) (Процессор Intel 2.5 3000)). Определить функцию, возвращающую стоимость товаров заданной фирмы.

Решение 7.5.1

```
(defun value (b w &optional (p 0) &aux (a (car w)))
  (if w
      (if (eq (cadr a) b)
          (value b (cdr w) (+ p (cadddr a)))
          (value b (cdr w) p)) p))

> (value 'intel '((terminal acer 15 4500) (terminal samsung 17 7000)
(motherboard intel 238 2500) (processor intel 2.5 3000)))
5500
```

Решение 7.5.2

```
(defun value (b w &optional (p 0) &aux (a (car w)))
  (cond ((null w) p)
        ((eq (cadr a) b) (value b (cdr w) (+ p (cadddr a))))
        ((value b (cdr w) p))))

> (value 'intel '((terminal acer 15 4500) (terminal samsung 17 7000)
(motherboard intel 238 2500) (processor intel 2.5 3000)))
5500
```

Решение 7.5.3

```
(defun value (b w)
  (loop for a in w when (eq (cadr a) b) sum (cadddr a)))

> (value 'intel '((terminal acer 15 4500) (terminal samsung 17 7000)
(motherboard intel 238 2500) (processor intel 2.5 3000)))
5500
```

Задача 7.6 count-goods .lisp

Прайс компьютерного магазина описывается списком (наименование фирма параметры цена): ((Терминал Acer 15 4500) (Терминал Sumsung 17 7000)(Матплата Asus 238 2500) (Процессор Intel 2.5 3000)). Определить функцию возвращающую количество единиц выбранного наименования.

Решение 7.6.1

```
(defun count-goods (m w &optional (n 0))
  (cond ((null w) n)
        ((eq (caar w) m) (count-goods m (cdr w) (1+ n)))
        ((count-goods m (cdr w) n))))

> (count-goods 'terminal '((terminal acer 15 4500) (terminal samsung
17 7000) (motherboard intel 238 2500) (processor intel 2.5 3000)))
2
```

Решение 7.6.2

```
(defun count-goods (m w)
  (count m w :key #'car))

> (count-goods 'terminal '((terminal acer 15 4500) (terminal samsung
17 7000) (motherboard intel 238 2500) (processor intel 2.5 3000)))
2
```

Задача 7.7 last-n.lisp

Дан список. Определить функцию, которая выдавала бы элемент списка по заданному номеру с конца (функции length и nth не использовать).

Решение 7.7.1

```
(defun -nth (w n)
  (if (= n 1) (car w) (-nth (cdr w) (1- n))))

(defun len (w)
  (if (null w) 0 (1+ (len (cdr w)))))

(defun last-n (w n)
```

```
  (-nth w (- (len w) (1- n))))

> (last-n '(5 4 3 2 1) 5)
5
```

Решение 7.7.2

```
(defun len (w)
  (if (null w) 0 (1+ (len (cdr w)))))

(defun last-n (w n)
  (if (= (len w) n) (car w) (last-n (cdr w) n)))

> (last-n '(5 4 3 2 1) 5)
5
```

Решение 7.7.3

```
(defun last-n (w n)
  (-nth (reverse w) n))

(defun -nth (w n)
  (if (= n 1) (car w) (-nth (cdr w) (1- n))))

> (last-n '(5 4 3 2 1) 5)
5
```

Решение 7.7.4

```
(defun last-n (w n)
  (elt (reverse w) (1- n)))

> (last-n '(5 4 3 2 1) 5)
5
```

Решение 7.7.5

```
(defun last-n (w n)
  (car (last w n)))

> (last-n '(5 4 3 2 1) 2)
2
```

Решение 7.7.6

```
(defun last-n (w n)
  (if (null (nthcdr n w)) (car w) (last-n (cdr w) n)))

> (last-n '(5 4 3 2 1) 2)
2
```

Решение 7.7.7

```
(defun last-n (w n)
  (car (last (delete-if #'identity w :count (1- n) :from-end t))))

> (last-n '(5 4 3 2 1) 2)
2
```

Задача 7.8 last-a.lisp

Определить номер последнего вхождения атома в многоуровневый список.

Решение 7.8.1

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun last-a (w a)
  (when (member a w) (1+ (position a (flat w) :from-end t))))

> (last-a '(a (a b) a (b a) v) 'a)
6
```

Решение 7.8.2

```
(defun last-a (w n &aux (a (car w)) (d (cdr w)))
  (when w (if (listp a)
              (last-a (nconc a d) n)
              (let ((r (last-a d n)))
                (if r (1+ r) (when (equal a n) 1))))))

> (last-a '(a (a b) a (b a) v) 'a)
6
```

Решение 7.8.3

```
(defun last-a (w n &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((atom a) ((lambda (r)
                     (if r (1+ r) (when (equal a n) 1)))
                   (last-a d n)))
        ((last-a (nconc a d) n))))

> (last-a '(a (a b) a (b a) v) 'a)
6
```

Решение 7.8.4

```
(defun last-a (w n &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((atom a) (let ((r (last-a d n)))
                    (if r (1+ r) (when (equal a n) 1))))
```

```
        ((last-a (nconc a d) n))))

> (last-a '(a (a b) a (b a) v) 'a)
6
```

Решение 7.8.5

```
(defun last-a (w n &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((listp a) (last-a (nconc a d) n))
        ((let ((r (last-a d n)))
           (if r (1+ r) (when (equal a n) 1))))))

> (last-a '(a (a b) a (b a) v) 'a)
6
```

Задача 7.9 count-same.lisp

Написать функцию, аргументом которой является список атомов, которая выдает значение истина - t, если в списке есть n и более одинаковых атома, и ложь nil в противном случае.

Решение 7.9.1

```
(defun count-same (w n &optional (v w))
  (cond ((null w) nil)
        ((>= (count (car w) v) n) t)
        ((count-same (cdr w) n v))))

> (count3 '(a b b b) 3)
T
```

Задача 7.10 count-elm.lisp

Найти число вхождений заданного элемента в список (на любом уровне вложенности).

Решение 7.10.1

```
(defun count-elm (w n &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 0)
        ((atom a) (if (eql n a)
                      (1+ (count-elm d n))
                      (count-elm d n)))
        ((+ (count-elm a n) (count-elm d n)))))

> (count-elm '((1 2) 2 (4 2 3)) 2)
3
```

Решение 7.10.2

```
(defun count-elm (n w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) 0)
        ((listp a)(+ (count-elm n a) (count-elm n d)))
        ((eql n a) (1+ (count-elm n d)))
        ((count-elm n d))))

> (count-elm 2 '((1 2) 2 (4 2 3)))
3
```

Решение 7.10.3 (автор – Lambdik, www.cyberforum.ru)

```
(defun count-elm (n w)
  (cond ((eql n w) 1)
        ((atom w) 0)
        (t (+ (count-elm n (car w)) (count-elm n (cdr w))))))

> (count-elm 2 '((1 2) 2 (4 2 3)))
3
```

Решение 7.10.4

```
(defun count-elm (n w)
  (if (eql n w) 1 (if (atom w) 0 (+ (count-elm n (car w))
                                    (count-elm n (cdr w))))))

> (count-elm 2 '((1 2) 2 (4 2 3)))
3
```

Задача 7.11 add&length.lisp

Добавить элемент в конец списка, если его нет в этом списке. Найти длину полученного списка. Если элемент есть в списке или список пустой, то значение функции nil.

Решение 7.11.1

```
(defun add&length (n w)
  (cond ((or (null w) (member n w)) nil)
        ((length (nconc w (list n))))))

> (add&length 4 '(0 1 2 3))
5
```

Задача 7.12 insider.lisp

Проверить, входит ли элемент в список.

Решение 7.12.1 (для одноуровневого списка)

```
(defun insider (w z)
  (cond ((null w) nil)
```

```
        ((eql (car w) z) t)
        ((insider (cdr w) z))))

> (insider '(a b) 'b)
T
```

Решение 7.12.2 (для многоуровневого списка)

```
(defun insider (w z &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((consp a) (or (insider a z) (insider d z)))
        ((eql a z) t)
        ((insider d z))))

> (insider '((a (b)) c) 'b)
T
```

Задача 7.13 count-atoms.lisp

Напишите функцию (count p w), которая подсчитывает, сколько атомов в многоуровневом списке w удовлетворяет предикату p.

Решение 7.13.1

```
(defun ft (w &optional ac)
(cond ((null w) ac)
      ((atom w) (cons w ac))
      ((ft (car w) (ft (cdr w) ac)))))

(defun fn (w p &optional (n 0))
  (cond ((null w) n)
        ((funcall p (car w)) (fn (cdr w) p (+ n 1)))
        ((fn (cdr w) p n))))

(defun count-atoms (w p)
  (fn (ft w) p))

> (count-atoms '(1 (2 3) 4) #'evenp)
2
```

Задача 7.14 elm.lisp

Определить функцию, которая принимает два аргумента – число и список, и возвращает элемент из списка, который соответствует номеру переданного числа.

Решение 7.14.1

```
(defun elm (n w)
  (cond ((null w) nil)
        ((= n 1) (car w))
```

```
        (t (elm (- n 1) (cdr w)))))

> (elm 2 '(a b c d e))
B
```

Решение 7.14.2

```
(defun elm (n w)
  (when w (if (= n 1) (car w) (elm (- n 1) (cdr w)))))

> (elm 2 '(a b c d e))
B
```

Решение 7.14.3

```
(defun elm (n w)
  (when w (if (= n 1) (car w) (elm (1- n) (cdr w)))))

> (elm 2 '(a b c d e))
B
```

Решение 7.14.4

```
(defun elm (n w)
  (when w (if (> n 1) (elm (1- n) (cdr w)) (car w))))

> (elm 2 '(a b c d e))
B
```

Задача 7.15 tail-minus.lisp

Определить функцию, которая принимает один или больше аргументов и возвращает число, которое получается, отнимая от первого аргумента остальные.

Решение 7.15.1

```
(defun tail-minus (&rest w) (reduce #'- w))

> (tail-minus 1 2 3)
-4
```

Решение 7.15.2

```
(defun tail-minus (&rest w)
  (- (car w) (apply #'+ (cdr w))))

> (tail-minus 1 2 3)
-4
```

Решение 7.15.3

```
(defun tail-minus (&rest w) (apply #'- w))

> (tail-minus 1 2 3)
-4
```

Задача 7.16 member-test.lisp

Функция возвращает t если элемент входит в список, в другом случае возвращает nil.

Решение 7.16.1

```
(defun member-test (a w)
  (cond ((null w) nil)
        ((eql (car w) a) t)
        ((member-test a (cdr w)))))

> (member-test 'd '(c a d))
T
> (member-test 'b '(c a d))
NIL
```

Задача 7.17 pos.lisp

Написать функцию, которая определяет на каком месте стоит заданный элемент в заданном списке.

Решение 7.17.1

```
(defun pos (a w &optional (p 0))
  (cond ((null w) nil)
        ((eql (car w) a) p)
        (t (pos a (cdr w) (+ p 1)))))

> (pos 2 '(0 1 2 3 4))
2
```

Решение 7.17.2

```
(defun pos (n w)
  (when (member n w) (os n w)))

(defun os (n w)
  (if (eq (car w) n) 0 (1+ (os n (cdr w)))))

> (pos 2 '(0 1 2 3 4))
2
> (pos 2 '(0 1 3 4))
NIL
```

Решение 7.17.3

```
(defun pos (n w)
  (when (member n w)
    (labels ((os (n w)
               (if (eq (car w) n) 0 (1+ (os n (cdr w))))))
      (os n w))))

> (pos 2 '(0 1 3 4))
NIL
> (pos 2 '(0 1 2 3 4))
2
```

Задача 7.18 cnt.lisp

Написать функцию, которая вычисляет количество вхождений элемента на верхнем уровне.

Решение 7.18.1

```
(defun cnt (a w)
  (cond ((null w) 0)
        ((equal  (car w) a) (1+ (cnt a (cdr w))))
        ((cnt a (cdr w)))))

> (cnt 'e '(a e e c))
2
```

Задача 7.19 sum-ns.lisp

Определить функцию, которая суммирует первые n чисел одноуровневого числового списка.

Решение 7.19.1

```
(defun sum-ns (n w)
  (if (zerop n) 0 (+ (car w) (sum-ns (1- n) (cdr w)))))

> (sum-ns 4 '(1 1 1 1 1))
4
```

Задача 7.20 check-console-primes.lisp

Определить функцию, которая проверяет, являются ли элементы списка seq простыми числами. Если элемент является простым числом, функция возвращает в результирующем списке значение T. Если элемент не является натуральными числом, функция возвращает в результирующем списке NIL. Ввод аргументов с консоли, вывод значения в файл.

Решение 7.20.1

```
(defun check (x)
```

```
  (when (> x 1) (loop for i from 2 to (isqrt x)
                      never (zerop (mod x i)))))

(defun check-comsole-primes (path)
  (with-open-file
      (r path :direction :output :if-exists :supersede)
    (format r  "~{~s ~}"
            (mapcar #'check
                    (loop for n = (read)
                          if (eq n 'quit) return numbers
                          else  collect n into numbers)))))
```

```
> (check-console-primes "d:/new.txt")
11
12
13
14
quit
NIL

d:/new.txt:
T NIL T NIL
```

Задача 7.21 check-file-primes.lisp

Определить функцию, которая проверяет, являются ли элементы списка seq простыми числами. Если элемент является простым числом, функция возвращает в результирующем списке значение T. Если элемент не является натуральными числом, функция возвращает в результирующем списке NIL. Ввод из файла, вывод значения на экран.

Решение 7.21.1

```
(defun check (x)
  (when (> x 1) (loop for i from 2 to (isqrt x)
                      never (zerop (mod x i)))))

(defun check-file-primes (path)
  (mapcar #'check
          (read-from-string
           (concatenate
            'string "("
            (with-open-file (s path :direction :input)
              (read-line s  nil :eof))
             ")"))))
```

```
> (check-fie-primes "d:/old.txt")
(T NIL T NIL)

d:/old.txt:
11 12 13 14
```

Задача 7.22 exceed-number.lisp

Создать программу, вычисляющую количество элементов больших n из введенного списка.

Решение 7.22.1

```
(defun exceed-number (n w)
  (cond ((null w) 0)
        ((> (car w) n) (+ 1 (exceed-number n (cdr w))))
        ((exceed-number n (cdr w)))))

> (exceed-number 5 '(0 5 6 7 8))
3
```

Задача 7.23 dive-count.lisp

Написать программу определения количества атомов на заданном уровне в произвольном исходном списке. Атомы, содержащиеся в исходном списке, имеют уровень 0, атомы, содержащиеся в следующем вложенном списке, имеют уровень 1 и т.д.

Решение 7.23.1

```
(defun dive-elms (n w &aux (a (car w)) (d (cdr w)))
   (cond ((null w) nil)
         ((= n 0) w)
         ((atom a) (dive-elms n d))
         ((nconc (dive-elms (1- n) a) (dive-elms n d)))))

(defun dive-count (n w)
  (count-if #'atom (dive-elms n w)))

> (dive-count 0 '(a(b(c 1)d e)f((2(g))3)))
2
> (dive-count 1 '(a(b(c 1)d e)f((2(g))3)))
4
> (dive-count 2 '(a(b(c 1)d e)f((2(g))3)))
3
```

Решение 7.23.2

```
(defun dive-count (n w &aux (a (car w)) (d (cdr w)))
   (cond ((null w) 0)
         ((= n 0) (count-if #'atom w))
         ((atom a) (dive-count n d))
         ((+ (dive-count (1- n) a) (dive-count n d)))))

> (dive-count 0 '(a(b(c 1)d e)f((2(g))3)))
2
> (dive-count 1 '(a(b(c 1)d e)f((2(g))3)))
4
```

```
> (dive-count 2 '(a(b(c 1)d e)f((2(g))3)))
3
```

Задача 7.24 any.lisp

Определить функционал, возвращающий истину, если определенным свойством обладает хотя бы один элемент списка.

Решение 7.24.1

```
(defun any (p w)
  (cond ((null w) nil)
        ((funcall p (car w)) t)
        ((any p (cdr w)))))

> (any #'minusp '(-1 1))
T
> (any #'minusp '(1 1))
NIL
```

Решение 7.24.2

```
(defun any (p w)
  (when w (or (funcall p (car w)) (any p (cdr w)))))

> (any #'minusp '(-1 1))
T
> (any #'minusp '(1 1))
NIL
```

Решение 7.24.3

```
(defun any (p w)
  (loop for a in w thereis (funcall p a)))

> (any #'minusp '(-1 1))
T
> (any #'minusp '(1 1))
NIL
```

Решение 7.24.4

```
(defun any (p w) (find-if p w))

> (any #'minusp '(1 1 ))
NIL
> (any #'minusp '(-1 1))
-1
```

Решение 7.24.5

```
(defun any (p w) (some p w))
```

```
> (any #'minusp '(-1 1))
T
> (any #'minusp '(1 1))
NIL
```

Задача 7.25 count-after.lisp

Определить функцию, возвращающую количество элементов списка, следующих после заданного символа.

Решение 7.25.1

```
(defun count-after (m w &optional f)
  (cond ((null w) 0)
        ((eq (car w) m) (count-after m (cdr w) t))
        (f (1+ (count-after m (cdr w) t)))
        ((count-after m (cdr w)))))

> (count-after '* '(a b c * 1 2 3 4))
4
> (count-after '+ '(a b c * 1 2 3 4))
0
```

Решение 7.25.2

```
(defun count-after (m w &aux (n (position m w)))
  (if n (length (nthcdr (1+ (position m w)) w)) 0))

> (count-after '* '(a b c * 1 2 3 4))
4
> (count-after '+ '(a b c * 1 2 3 4))
0
```

Решение 7.25.3

```
(defun count-after (m w &aux (n (position m w)))
  (if n (count-if #'identity w :start (1+ n)) 0))

> (count-after '* '(a b c * 1 2 3 4))
4
> (count-after '+ '(a b c * 1 2 3 4))
0
```

Решение 7.25.4

```
(defun count-after (m w &aux (v (member m w)))
  (if v (1- (length v)) 0))

> (count-after '* '(a b c * 1 2 3 4))
4
> (count-after '+ '(a b c * 1 2 3 4))
```

```
0
```

Задача 7.26 rhymes.lisp

Определить функцию, удаляющую каждый n-й элемента в списке, пока не останется один элемент (считалка).

Решение 7.26.1

```
(defun rhymes (w n &optional acc)
  (if (nth n w)
      (rhymes (nthcdr n w) n (nconc acc (subseq w 0 (1- n))))
      (drop-thin (nconc w acc) n)))

(defun drop-thin (w n)
  (if (cdr w)
      (drop-thin (drop-n w (rem n (length w))) n)
      (car w)))

(defun drop-n (w n)
  (print w)
  (if (> n 0)
      (nconc (nthcdr n w ) (subseq w 0 (1- n)))
      (butlast w)))

> (rhymes '(a b c d e f g h) 5)
(F G H A B C D)
(C D F G H A)
(A C D F G)
(A C D F)
(C D F)
(F C)
C
> (rhymes '(a b c d e f g h i j k) 7)
(H I J K A B C D E F)
(D E F H I J K A B)
(A B D E F H I J)
(J A B D E F H)
(J A B D E F)
(A B D E F)
(D E F A)
(A D E)
(D E)
E
```

Решение 7.26.2

```
(defun rhymes (w n &optional acc)
  (if (nth n w)
      (rhymes (nthcdr n w) n (nconc acc (subseq w 0 (1- n))))
      (drop-thin (nconc w acc) n)))
```

```
(defun drop-thin (w n &aux (m (rem n (length w))))
  (print w)
  (if (cdr w)
      (drop-thin (if (> m 0)
                     (nconc (nthcdr m w ) (subseq w 0 (1- m)))
                     (butlast w))
                 n)
      (car w)))

> (rhymes '(a b c d e f g h) 5)
(F G H A B C D)
(C D F G H A)
(A C D F G)
(A C D F)
(C D F)
(F C)
(C)
C
> (rhymes '(a b c d e f g h i j k) 7)
(H I J K A B C D E F)
(D E F H I J K A B)
(A B D E F H I J)
(J A B D E F H)
(J A B D E F)
(A B D E F)
(D E F A)
(A D E)
(D E)
(E)
E
```

Решение 7.26.3

```
(defun rhymes (w n &optional acc)
  (if (nth n w)
      (rhymes (subseq w n) n (nconc acc (subseq w 0 (1- n))))
      (drop-thin (nconc w acc) n)))

(defun drop-thin (w n &aux (m (rem n (length w))))
  (print w)
  (if (cdr w)
      (drop-thin (if (> m 0)
                     (nconc (subseq w m) (subseq w 0 (1- m)))
                     (butlast w))
                 n)
      (car w)))

> (rhymes '(a b c d e f g h) 5)
(F G H A B C D)
(C D F G H A)
(A C D F G)
(A C D F)
(C D F)
```

```
(F C)
(C)
C
> (rhymes '(a b c d e f g h i j k) 7)
(H I J K A B C D E F)
(D E F H I J K A B)
(A B D E F H I J)
(J A B D E F H)
(J A B D E F)
(A B D E F)
(D E F A)
(A D E)
(D E)
(E)
E
```

Решение 7.26.4

```
(defun rhymes (w n &optional acc)
  (if (nth n w)
      (rhymes (subseq w n) n (nconc acc (subseq w 0 (1- n))))
      (drop (nconc w acc) n)))

(defun drop (w n &aux (m (rem n (length w))))
  (if (cdr w)
      (drop (if (> m 0)
                (nconc (subseq w m) (subseq w 0 (1- m)))
                (butlast w))
            n)
      w))
```

```
> (rhymes '(a b c d e f g h i j k) 7)
(E)
> (rhymes '(a b c d e f g h) 5)
(C)
```

Решение 7.26.5

```
(defun rhymes (w n &optional acc)
  (if (nth n w)
      (rhymes (subseq w n) n (nconc acc (subseq w 0 (1- n))))
      (drop (nconc w acc) n)))

(defun drop (w n &aux (m (rem n (length w))))
  (if (cdr w)
      (drop (if (> m 0)
                (nconc (subseq w m) (subseq w 0 (1- m)))
                (butlast w))
            n)
      (car w)))
```

```
> (rhymes '(a b c d e f g h) 5)
C
```

```
> (rhymes '(a b c d e f g h i j k) 7)
E
```

Задача 7.27 n-elem-n.lisp

Определить функцию, которая выбирает на верхнем уровне списка n-ый элемент, отсчитанный от начала списка, если n > 0, или от конца списка, если n < 0.

Решение 7.27.1

```
(defun n-elm-n (w n)
  (nth (1- (abs n)) (if (plusp n) w (reverse w))))

> (n-elm-n '(1 2 3 4 5) 1)
1
> (n-elm-n '(1 2 3 4 5) -1)
5
```

Структура 8 (функция (СПИСОК) АТОМ) > (СПИСОК)

Задача 8.1 _mapcar.lisp

Определить функцию применения другой переданной в качестве параметра функции ко всем элементам заданного списка.

Решение 8.1.1

```
(defun _mapcar (p w)
  (when w (cons (funcall p (car w)) (_mapcar p (cdr w)))))

> (_mapcar #'1- '(1 2 3))
(0 1 2)
```

До тех пор - when, пока не опустел (существует) список w включаем - cons результат применения функции p, переданной в качестве параметра определяемой функции _mapcar, к первому элементу (car w) списка w в начало списка, полученного в результате вызова определяемой функции _mapcar с параметрами: p и хвостом (cdr w) списка w.

Задача 8.2 multiply.lisp

Определить функцию, которая умножает элементы списка (числа) на заданное число.

Решение 8.2.1

Далее для краткости, единообразия и сходства по начертанию: w - больший список, v – меньший.

```
(defun multiply (w n)
  (cond ((null w) nil)
        ((cons (* (car w) n) (multiply (cdr w) n)))))

> (multiply '(1 2) 2)
(2 4)
```

Комментарии (построчно):
1. Определяем функцию multiply с параметрами: список **w и число n**.
2. Если (**cond**) пустой (**null**) список **w,** функция multiply возвращает **nil**
3. При невыполнении условия **в**ставляет (**cons**) произведение первого элемента (**car w**) и числа n в результат вызова multiply с параметрами: списком без первого элемента (**cdr w**) и числом n.

Решение 8.2.2

```
(defun multiply (w n)
  (if (null w) nil (cons (* (car w) n) (multiply (cdr w) n))))

> (multiply '(1 2) 2)
(2 4)
```

Решение 8.2.3

```
(defun multiply (w n)
  (when w (cons (* (car w) n) (multiply (cdr w) n))))

> (multiply '(1 2) 2)
(2 4)
```

Решение 8.2.4

```
(defun multiply (w n)
  (mapcar #'(lambda (a) (* a n)) w))

> (multiply '(1 2) 2)
(2 4)
```

Решение 8.2.5

```
(defun multiply (w n)
  (loop for a in w collect (* a n)))

> (multiply '(1 2) 2)
(2 4)
```

Задача 8.3 drop-elm.lisp

Определить функцию, которая удаляет из списка все совподающие с данным атомом элементы и возвращает в качестве значения список из остатка элементов.

Решение 8.3.1

```
(defun drop-elm (n w)
  (cond ((null w) nil)
        ((eql (car w) n) (drop-elm n (cdr w)))
        ((cons (car w) (drop-elm n (cdr w))))))

> (drop-elm 3 '(1 2 3))
(1 2)
```

Комментарии (построчно):
1. Определяем функцию **drop-elm** с двумя параметрами: атом **n** и список **w**.
2. Если (**cond**) пустой (**null**) список **w** функция **drop-elm** возвращает **nil**
3. Если первый элемент списка (**car w**) равен **n** вызывается функция **drop-elm** с параметрами: **n** и список без первого элемента (**cdr w**).
4. При невыполнении 1 и 2 условий вставляет (**cons**) первый элемент (**car w**) в результат вызова **drop-elm** с параметрами: **n** и списком без первого элемента (**cdr w**).

Решение 8.3.2

```
(defun drop-elm (a w)
  (when w (if (equal a (car w))
              (drop-elm a (cdr w))
              (cons (car w) (drop-elm a (cdr w))))))

> (drop-elm 3 '(1 2 3))
(1 2)
```

Задача 8.4 add-elt.lisp

Проверить есть ли символ в списке, если нет, то добавить его в список.

Решение 8.4.1

```
(defun add-elt (e w)
  (cond ((member e w) w)
        ((cons e w))))

> (add-elt '1 '(2 3))
(1 2 3)
> (add-elt 'a '(b c))
(A B C)
```

Задача 8.5 group2.lisp

Написать функцию, которая разбивает исходный список на пары (a1 a2)(a3 a4)...

Решение 8.5.1

```
(defun group2 (w)
  (cond ((cadr w) (cons (list (car w) (cadr w)) (group2 (cddr w))))
        (w (cons w nil))))

> (group2 '(a b c d e f))
((A B) (C D) (E F))
```

Решение 8.5.2

```
(defun group2 (w)
  (loop for v on w by #'cddr collect
        (if (cadr v)
            (subseq v 0 2)
            (subseq v 0))))

> (group '(a b c d e f))
((A B) (C D) (E F))
```

Задача 8.6 blast-els.lisp

Необходимо удалить n последних элементов из списка и вывести первоначальный список без удаленных элементов.

Решение 8.6.1

```
(defun blast-els (w n)
  (delete-if #'identity w :count n :from-end t))

> (blast-els '(1 2 3 4 5) 2)
(1 2 3)
```

Решение 8.6.2

```
(defun blast-els (w n)
  (butlast w n))

> (blast-els '(1 2 3 4 5) 2)
(1 2 3)
```

Задача 8.7 delete-first.lisp

Определить функцию, которая удаляет из списка первое вхождение заданного атома на верхнем уровне.

Решение 8.7.1

```
(defun delete-first (a w)
  (cond ((null w) nil)
        ((equal a (car w)) (cdr w))
        ((cons (car w) (delete-first a (cdr w))))))
```

```
> (delete-first 'a '(a b c (a b c)))
(B C (A B C))
> (delete-first 'a '((a b) c a b c))
((A B) C B C)
```

1. Если (null w), то достигли конца списка, вернули nil - пустой список (к нему в глубине рекурсии будет присоединен результат выполнения третьего условия, если элемент не будет найден);
2. Если (equal a (car w)) - найден нужный элемент, то вернули (cdr w) хвост списка;
3. Если не выполнилось ни одно из условий, то (cons - создаем список из (car w) - головы списка и (delete-first a (cdr w)) результата вызова функции с параметрами элемент и хвост списка.

Решение 8.7.2

```
(defun delete-first (a w)
  (cond ((or (null w) (eql a (car w))) (cdr w))
        ((cons (car w) (delete-first a (cdr w))))))

> (delete-first 'a '((a b) c a b c))
((A B) C B C)
> (delete-first 'a '(a b c (a b c)))
(B C (A B C))
> (delete-first 'a '((a b) c (a b c)))
((A B) C (A B C))
```

Решение 8.7.3

```
(defun delete-first (a w)
  (if (or (null w) (eql a (car w)))
      (cdr w)
      (cons (car w) (delete-first a (cdr w)))))

> (delete-first 'a '((a b) c (a b c)))
((A B) C (A B C))
> (delete-first 'a '(a b c (a b c)))
(B C (A B C))
> (delete-first 'a '((a b) c a b c))
((A B) C B C)
```

Задача 8.8 frequent.lisp

Дан список. Определить функцию, находящую элементы в списке встречающиеся ровно n раз.

Решение 8.8.1

```
(defun frequent (w n)
  (remove-if-not
   #'(lambda (a) (= (count a w) n)) (remove-duplicates w)))
```

```
> (time (frequent '(1 2 3 4 5 5 4 5 1 2 1 2) 3))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 520 Bytes
(5 1 2)
```

Решение 8.8.2

```
(defun frequent (w n)
  (delete-if-not
   #'(lambda (a) (= (count a w) n)) (remove-duplicates w)))

> (time (frequent '(1 2 3 4 5 5 4 5 1 2 1 2) 3))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 496 Bytes
(5 1 2)
```

Решение 8.8.3

```
(defun frequent (w n)
  (loop for a in (remove-duplicates w)
        when (= (count a w) n) collect a))

> (time (frequent '(1 2 3 4 5 5 4 5 1 2 1 2) 3))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 468 Bytes
(5 1 2)
```

Решение 8.8.4

```
(defun frequent (w n &optional (v (remove-duplicates w))
                 &aux (a (car v)))
  (cond ((null v) nil)
        ((= (count a w) n) (cons a (frequent w n (cdr v))))
        ((frequent w n (cdr v)))))

> (time (frequent '(1 2 3 4 5 5 4 5 1 2 1 2) 3))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 468 Bytes
(5 1 2)
```

Решение 8.8.5

```
(defun frequent (w n &optional (v (remove-duplicates w))
                 &aux (a (car v)))
  (when v (if (= (count a w) n)
              (cons a (frequent w n (cdr v)))
              (frequent w n (cdr v)))))
```

```
> (time (frequent '(1 2 3 4 5 5 4 5 1 2 1 2) 3))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 468 Bytes
(5 1 2)
```

Решение 8.8.6

```
(defun quent (v w n)
  (loop for a in v when (= (count a w) n) collect a))

(defun frequent (w n)
  (quent (remove-duplicates w) w n))
```

```
> (time (frequent '(1 2 3 4 5 3 3 3 5 4 5 1 2 1 2) 3))
Real time: 0.0010001 sec.
Run time: 0.0 sec.
Space: 468 Bytes
(5 1 2)
```

Решение 8.8.7

```
(defun frequent (w n)
  (cond ((null w) nil)
        ((= n (count (car w) w))
         (cons (car w) (frequent (remove (car w) (cdr w)) n)))
        ((frequent (remove (car w) (cdr w)) n))))
```

```
> (time (frequent '(1 2 3 4 5 3 3 3 5 4 5 1 2 1 2) 3))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 316 Bytes
(1 2 5)
```

Решение 8.8.8

```
(defun frequent (w n)
  (cond ((null w) nil)
        ((= n (count (car w) w))
         (cons (car w) (frequent (delete (car w) (cdr w)) n)))
        ((frequent (delete (car w) (cdr w)) n))))
```

```
> (time (frequent '(1 2 3 4 5 3 3 3 5 4 5 1 2 1 2) 3))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 84 Bytes
(1 2 5)
```

Задача 8.9 mapping.lisp

Определить функцию, аргументами которой являются функция и список, рассматриваемый как множество, а результат применения - множество

(список без повторяющихся элементов), которое можно получить применением функции-аргумента, к каждому элементу списка. Порядок элементов не имеет значения.

Решение 8.9.1

```
(defun mapping (f w)
  (remove-duplicates (mapcar f w)))

> (mapping #'(lambda (z) (/ (- z (rem z 10)) 10)) '(1 57 101 102 53
6))
(10 5 0)
```

Решение 8.9.2

```
(defun sine-duos (w)
  (cond ((null w) nil)
        ((member (car w) (cdr w)) (sine-duos (cdr w)))
        ((cons (car w) (sine-duos (cdr w))))))

(defun mapping (f w)
  (reverse
   (sine-duos (mapcar #'(lambda (a) (funcall f a)) w))))

> (mapping (lambda (z) (/ (- z (rem z 10)) 10)) '(1 57 101 102 53 6))
(0 5 10)
```

Решение 8.9.3

```
(defun -member (a w)
       (cond ((null w) nil)
       ((equalp a (car w)) (cons (car w) (cdr w)))
       ((member a (cdr w)))))

(defun duplicates- (w)
  (cond ((null w) nil)
        ((-member (car w) (cdr w)) (duplicates- (cdr w)))
        ((cons (car w) (duplicates- (cdr w))))))

(defun mapping (f w)
  (reverse
   (duplicates- (mapcar #'(lambda (a) (funcall f a)) w))))

> (mapping #'(lambda (z) (/ (- z (rem z 10)) 10)) '(1 57 101 102 53
6))
(0 5 10)
```

Задача 8.10 group-n.lisp

Определить функцию, которая группирует элементы списка.

Решение 8.10.1

```
(defun group-n (w n &optional ac (m n) &aux (a (car w)) (d (cdr w)))
  (cond ((null w) (when (> m 0) (reverse ac)))
        ((= m 1) (cons (reverse (cons a ac)) (group-n d n nil n)))
        ((group-n d n (cons a ac) (1- m)))))

> (group-n '(1 2 3 4 5 6 7 8 9 10 11) 3)
((1 2 3) (4 5 6) (7 8 9) 10 11)
```

Задача 8.11 quadros.lisp

Дан список и число n. Разбить список на подсписки длины n.

Решение 8.11.1

```
(defun quadros (w n &optional ac acc (m n))
  (cond ((null w)  (reverse (cons (reverse ac) (reverse acc))))
        ((= m 1) (quadros (cdr w)
                          n
                          nil
                          (cons (reverse (cons (car w) ac)) acc)
                          n))
        ((quadros (cdr w) n (cons (car w) ac) acc (1- m)))))

> (quadros '(A B (C D) 1 2 NIL) 4)
((A B (C D) 1) (2 NIL))
```

Задача 8.12 _remove-if-not.lisp

Определите фильтр (удалить-если-не предикат список), удаляющий из списка элементы, которые не обладают свойством, наличие которого проверяет предикат.

Решение 8.12.1

```
(defun _remove-if-not (p w)
  (cond ((null w) nil)
        ((funcall p (car w))
         (cons (car w) (_remove-if-not p (cdr w))))
        ((_remove-if-not p (cdr w)))))

> (_remove-if-not #'evenp '(1 2))
(2)
```

Решение 8.12.2

```
(defun _remove-if-not (p w)
  (mapcan #'(lambda (a) (when (funcall p a) (list a))) w))

> (_remove-if-not #'evenp '(1 2))
(2)
```

Решение 8.12.3

```
(defun _remove-if-not (p w)
  (loop for a in w when (funcall p a) collect a))

> (_remove-if-not #'evenp '(1 2))
(2)
```

Задача 8.13 dive-elms.lisp

Написать программу с двумя параметрами (n - целое число, w - список), чтобы получить из списка w новый список из элементов, находящихся на уровне n. n=1 самый верхний уровень, возвращается сам список.

Решение 8.13.1

```
(defun dive-elms (n w)
  (if (= n 1) w (dive-elms (1- n) (elms w))))

(defun elms (w)
  (cond ((null w) nil)
        ((atom (car w)) (elms (cdr w)))
        ((nconc (car w) (elms (cdr w))))))

> (dive-elms 1 '((a ((b (c) d))) (a ((b (c) d)))))
((A ((B (C) D))) (A ((B (C) D))))
> (dive-elms 2 '((a ((b (c) d))) (a ((b (c) d)))))
(A ((B (C) D)) A ((B (C) D)))
> (dive-elms 3 '((a ((b (c) d))) (a ((b (c) d)))))
((B (C) D) (B (C) D))
> (dive-elms 4 '((a ((b (c) d))) (a ((b (c) d)))))
(B (C) D B (C) D)
> (dive-elms 5 '((a ((b (c) d))) (a ((b (c) d)))))
(C C)
> (dive-elms 6 '((a ((b (c) d))) (a ((b (c) d)))))
NIL
```

Решение 8.13.2

```
(defun dive-elms (n w)
  (labels ((elms (w)
             (cond ((null w) nil)
                   ((atom (car w)) (elms (cdr w)))
                   ((nconc (car w) (elms (cdr w)))))))
    (if (= n 1) w (dive-elms (1- n) (elms w)))))

> (dive-elms 1 '((a ((b (c) d))) (a ((b (c) d)))))
((A ((B (C) D))) (A ((B (C) D))))
> (dive-elms 2 '((a ((b (c) d))) (a ((b (c) d)))))
(A ((B (C) D)) A ((B (C) D)))
> (dive-elms 3 '((a ((b (c) d))) (a ((b (c) d)))))
```

```
((B (C) D) (B (C) D))
> (dive-elms 4 '((a ((b (c) d))) (a ((b (c) d)))))
(B (C) D B (C) D)
> (dive-elms 5 '((a ((b (c) d))) (a ((b (c) d)))))
(C C)
> (dive-elms 6 '((a ((b (c) d))) (a ((b (c) d)))))
NIL
```

Решение 8.13.3

```
(defun dive-elms (n w &aux (a (car w)) (d (cdr w)))
   (cond ((null w) nil)
         ((= n 1) w)
         ((atom a) (dive-elms n d))
         ((nconc (dive-elms (1- n) a) (dive-elms n d)))))
```

```
> (dive-elms 1 '((a ((b (c) d))) (a ((b (c) d)))))
((A ((B (C) D))) (A ((B (C) D))))
> (dive-elms 2 '((a ((b (c) d))) (a ((b (c) d)))))
(A ((B (C) D)) A ((B (C) D)))
> (dive-elms 3 '((a ((b (c) d))) (a ((b (c) d)))))
((B (C) D) (B (C) D))
> (dive-elms 4 '((a ((b (c) d))) (a ((b (c) d)))))
(B (C) D B (C) D)
> (dive-elms 5 '((a ((b (c) d))) (a ((b (c) d)))))
(C C)
> (dive-elms 6 '((a ((b (c) d))) (a ((b (c) d)))))
NIL
```

Задача 8.14 get-while.lisp

Напишите функцию (function w p), которая в качестве результата выдает список, состоящий из всех элементов списка w, с начала списка и до первого элемента, не удовлетворяющего данному предикату p. Например: (function '(2 4 6 7 8 9) #'evenp) возвращает (2 4 6).

Решение 8.14.1

```
(defun get-while (w p)
  (when w (when (funcall p (car w))
            (cons (car w) (get-while (cdr w) p)))))
```

```
> (get-while '(2 4 6 5 7 4) #'evenp)
(2 4 6)

> (get-while nil #'evenp)
NIL
```

Решение 8.14.2

```
(defun get-while (w p)
  (when (and w (funcall p (car w)))
```

```
    (cons (car w) (get-while (cdr w) p))))

> (get-while '(2 4 6 5 7 4) #'evenp)
(2 4 6)
> (get-while nil #'evenp)
NIL
```

Решение 8.14.3

```
(defun get-while (w p)
  (cond ((null w) nil)
        ((funcall p (car w)) (cons (car w) (get-while (cdr w) p)))
        (nil)))

> (get-while '(2 4 6 5 7 4) #'evenp)
(2 4 6)
> (get-while nil #'evenp)
NIL
```

Решение 8.14.4

```
(defun get-while (w p)
  (let ((a (car w)))
    (cond ((null w) nil)
          ((funcall p a) (cons a (get-while (cdr w) p)))
          (nil))))

> (get-while '(2 4 6 5 7 4) #'evenp)
(2 4 6)
> (get-while nil #'evenp)
NIL
```

Решение 8.14.5

```
(defun get-while (w p &aux (a (car w)))
  (cond ((null w) nil)
        ((funcall p a) (cons a (get-while (cdr w) p)))
        (nil)))

> (get-while '(2 4 6 5 7 4) #'evenp)
(2 4 6)
> (get-while nil #'evenp)
NIL
```

Задача 8.15 dump-level.lisp

Написать функцию, удаляющую из исходного списка подсписки заданной глубины. При описании функций нельзя использовать функции set, setq, setf и циклы.

Решение 8.15.1

```
(defun dump-level (w n)
  (when w ((lambda (a d)
             (cond ((atom a) (cons a (dump-level d n)))
                   ((= n 0) (dump-level d n))
                   ((cons (dump-level a (1- n)) (dump-level d n)))))
           (car w) (cdr w))))

> (dump-level '(5 (4 (6)) (1 7) (8 (3 (5)))) 0)
(5)
> (dump-level '(5 (4 (6)) (1 7) (8 (3 (5)))) 1)
(5 (4) (1 7) (8))
> (dump-level '(5 (4 (6)) (1 7) (8 (3 (5)))) 2)
(5 (4 (6)) (1 7) (8 (3)))
> (dump-level '(5 (4 (6)) (1 7) (8 (3 (5)))) 3)
(5 (4 (6)) (1 7) (8 (3 (5))))
```

Задача 8.16 addn.lisp

Определите функцию (f s n), которая из списка чисел s создает новый список, прибавляя к каждому атому число n. Исходный список не предполагается одноуровневым.

Решение 8.16.1

```
(defun addn (w n &aux (a (car w)))
  (cond ((null w) nil)
        ((consp a) (cons (addn a n) (add (cdr w) n)))
        ((cons (+ a n) (addn (cdr w) n)))))

> (addn '(1 (2) (3 (4)) 5) 1)
(2 (3) (4 (5)) 6)
```

Задача 8.17 select-p.lisp

Определите функцию (f n v p), которая выдает список всех элементов списка v, удовлетворяющих некоторому предикату p и встречающихся в исходном списке более n раз.

Решение 8.17.1

```
(defun select-p (n w p &optional v &aux (a (car w)))
  (cond ((null w) (reverse v))
        ((and (not (member a v)) (funcall p a)(> (count a w) n))
         (select-p n (cdr w) p (cons a v)))
        ((select-p n (cdr w) p v))))

> (selectv-p 1 '(0 1 0 1) #'oddp)
(1)
```

Задача 8.18 plunge-in.lisp

Добавить число в упорядоченный в порядке возрастания список без нарушения порядка. Например: >(функция 7 '(1 3 5 8 11)) результат: (1 3 5 7 8 11).

Решение 8.18.1

```
(defun plunge-in (n w)
  (when w (let ((a (car w)) (d (cdr w)))
            (cond ((atom d) (list a n))
                  ((> n a) (cons a (plunge-in n d)))
                  ((cons n w))))))

> (plunge-in 7 '(1 3 5 8 11))
(1 3 5 7 8 11)
```

Решение 8.18.2

```
(defun plunge-in (n w)
  (when w ((lambda (a d)
             (cond ((atom d) (list a n))
                   ((> n a) (cons a (plunge-in n d)))
                   ((cons n w))))
           (car w) (cdr w))))

> (plunge-in 7 '(1 3 5 8 11))
(1 3 5 7 8 11)
```

Решение 8.18.3

```
(defun squeeze-in (n w)
  (let ((e))
    (loop for i from 0 below (length w) do
          ((lambda (u)
             (when  (and  (<= n u) (or (= i 0) (> n (nth (1- i) w))))
               (push n e))
             (push u e)
             (when (and (> n u) (= (length w) (1+ i)))
               (push n e)))
           (nth i w)))
    (nreverse e)))

> (squeeze-in 7 '(1 3 5 8 11))
(1 3 5 7 8 11)
```

Решение 8.18.4

```
(defun cram-in (n w)
  (sort (cons n w) #'<))

> (cram-in 7 '(1 3 5 8 11))
(1 3 5 7 8 11)
```

Решение 8.18.5

```
(defun set-in (n w)
  (cond ((null w) (cons n nil))
        ((> n (car w)) (cons (car w) (set-in n (cdr w))))
        ((cons n w))))

> (set-in 7 '(1 3 5 8 11))
(1 3 5 7 8 11)
```

Решение 8.18.6

```
(defun set-in (n w)
  (cond ((null w) `(,n))
        ((> n (car w)) (cons (car w) (set-in n (cdr w))))
        ((cons n w))))

> (set-in 7 '(1 3 5 8 11))
(1 3 5 7 8 11)
```

Решение 8.18.7

```
(defun set-in (n w &aux (a (car w)))
  (cond ((null w) `(,n))
        ((> n a) (cons a (set-in n (cdr w))))
        ((cons n w))))

> (set-in 7 '(1 3 5 8 11))
(1 3 5 7 8 11)
```

Решение 8.18.8

```
(defun set-in (n w &aux (a (car w)))
  (if w (if (> n a) (cons a (set-in n (cdr w))) (cons n w)) `(,n)))

> (set-in 7 '(1 3 5 8 11))
(1 3 5 7 8 11)
```

Задача 8.19 substitute-first-or-last.lisp

Есть список и некоторый объект. Написать функцию, которая возвращает новый список, в котором объект заменяет первый элемент списка, если первый элемент списка и объект является атомами, иначе заменяет последний элемент списка.

Решение 8.19

```
(defun substitute-first-or-last (w v)
  (if (and (atom (car w)) (atom v)) (cons v (cdr w))
      (append (butlast w) (list v))))

> (substitute-first-or-last '(a b c) 'z)
(Z B C)
```

Задача 8.20 mask-list.lisp

Задан список. Вывести все элементы по заданной маске с указанием индекса элемента. Пример - дан: (1 a b 3 a c f a), маска (a), результат: (2 a 5 a 8 a).

Решение 8.20.1

```
(defun mask-list
    (w z &optional (n 1) &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((eq a z) (cons n (cons z (mask-list d z (1+ n)))))
        ((mask-list d z (1+ n)))))

> (mask-list '(1 a b 3 a c f a) 'a)
(2 A 5 A 8 A)
```

Решение 8.20.2

```
(defun ne (w a n)
  (cond ((null w) nil)
        ((eq a (car w)) (cons n (cons a (ne (cdr w) a (1+ n)))))
        ((ne (cdr w) a (1+ n)))))

(defun mask-list (w a)
  (ne w a 1))

> (mask-list '(1 a b 3 a c f a) 'a)
(2 A 5 A 8 A)
```

Задача 8.21 repeat-elts.lisp

Напишите функцию (f w n), которая выдает список, получающийся из списка w увеличением вхождения каждого элемента на n, например: (f '(1 2 3) 4) = (1 1 1 1 2 2 2 2 3 3 3 3).

Решение 8.21.1

```
(defun repeat-elts (w n &optional (m n))
  (cond ((null w) nil)
        ((zerop m) (repeat-elts (cdr w) n n))
        ((cons (car w) (repeat-elts w n (1- m))))))

>  (repeat-elts '(1 2 3) 4)
(1 1 1 1 2 2 2 2 3 3 3 3)
```

Задача 8.22 del-last-ns.lisp

Напишите функцию (f w n), которая удаляет n элементов с конца списка w.

Решение 8.22.1

```
(defun del-last-ns (w n)
  (delete-if #'identity w :count n :from-end t))

> (del-last-ns '(a b) 1)
(A)
```

Решение 8.22.2

```
(defun del-last-ns (w n)
  (butlast w n))

> (del-last-ns '(a b) 1)
(A)
```

Задача 8.23 shift-right.lisp

Осуществить циклический сдвиг элементов списка вправо, количество шагов задаётся.

Решение 8.23.1

```
(defun r> (w n)
  (append (nthcdr n w) (subseq w 0 n)))

> (r> '(a b c d e) 2)
(C D E A B)
```

Задача 8.24 remove-i+n.lisp

Дан список w и число n. Реализовать функцию, которая удаляет все i+n элементы списка.

Решение 8.24.1

```
(defun remove-i+n (w n)
  (when (and w (plusp n)) (cons (car w) (wx (cdr w) (1- n)))))

> (remove-i+n '(1 2) 1)
(1)
```

Задача 8.25 insert-@.lisp

Написать функционал, который ставит символ @ перед каждым элементом списка отвечающего критерию, например каждым элементом, который больше 0 или каждым нечетным элементом.

Решение 8.25.1

```
(defun insert-@ (w f &aux (a (car w)) (d (cdr w)))
```

```
  (cond ((null w) nil)
        ((funcall f a) (cons '@ (cons a (insert-@ d f))))
        ((cons a (insert-@ d f)))))

> (insert-@ '(1 0 -2 3 0 -4 5) #'evenp)
(1 @ 0 @ -2 3 @ 0 @ -4 5)

> (insert-@ '(1 0 -2 3 0 -4 5) #'(lambda (x) (>= x 0)))
(@ 1 @ 0 -2 @ 3 @ 0 -4 @ 5)
```

Переменная w - список; переменная f - функция; &aux предваряет, вспомогательные параметры (a (car w)) (d (cdr w)), от "auxiliary" - вспомогательный, вместо него можно применить let в теле функции, или просто заменить все a на (car w) и все d на (cdr w).

*Задача 8.26 hide****.lisp*

Определить функционал, заменяющий все элементы списка, не обладающие определенным свойством, на символ *. Функционал для предикатов:
- число;
- неположительное число (при вызове используйте лямбда-функцию).

Решение 8.26.1

```
(defun hide**** (w f &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((funcall f a) (cons a (hide**** d f)))
        ((cons '* (hide**** d f)))))

> (hide**** '(a 1) #'numberp)
(* 1)

> (hide**** '(1 0) #'(lambda (x) (< x 0)))
(* *)
```

Задача 8.27 del-over-n.lisp

Удалить из подсписков числа больше n, с использовнием спискоразрушающих функций. Например:

```
> (f '(1 2 (3 4 7) 8 9 (5 1)) 5)
(1 2 (3 4) 8 9 (5 1))
```

Решение 8.27.1

```
(defun del-over-n (w m &aux (a (car w)) (d (cdr w)))
  (cond((null w) nil)
       ((atom a) (cons a (del-over-n d m)))
       ((cons (delete-if #'(lambda (z) (> z m)) a) (del-over-n d
m)))))
```

```
> (del-over-n '(1 2 (3 4 7) 8 9 (5 1)) 5)
(1 2 (3 4) 8 9 (5 1))
```

Задача 8.28 delete-elt.lisp

Определите функцию, удаляющую заданный элемент из списка.

Решение 8.28.1.

```
(defun delete-elt (w a)
  (cond ((null w) nil)
        ((eq (car w) a) (delete-elt (cdr w) a))
        ((cons (car w)(delete-elt (cdr w) a)))))

> (delete-elt '(a b a c) 'a)
(B C)
```

Задача 8.29 left-shift.lisp

Осуществить циклический сдвиг элементов списка влево, количество шагов задаётся.

Решение 8.29.1

```
(defun <r (w n)
  (append (last w n) (butlast w n)))

> (<r '(a b c d e) 2)
(D E A B C)
```

Задача 8.30 rotate>.lisp

Осуществить n циклических сдвигов вправо элементов исходного списка.

Решение 8.30.1

```
(defun rotate> (w n)
  (if (zerop n) w (rotate> (nconc (last w) (butlast w)) (1- n))))

> (rotate> '(1 2 3 4 5) 2)
(4 5 1 2 3)
```

Решение 8.30.2

```
(defun rotate> (w n)
  (if (zerop n) w (rotate> (cons (car (last w)) (butlast w)) (1- n))))

> (rotate> '(1 2 3 4 5) 2)
(4 5 1 2 3)
```

Решение 8.30.3

```
(defun rotate> (w n)
  (nconc  (last w n) (butlast w n)))

> (rotate> '(1 2 3 4 5) 2)
(4 5 1 2 3)
```

Задача 8.31 cut-n-elms.lisp

Написать программу формирующую список, состоящий из не более чем n элементов исходного списка.

Решение 8.31.1

```
(defun cut-n-elms (n w)
  (loop for a in w
        for b downfrom n unless (zerop b) collect a))

> (cut-n-elms 2 '(1 2 3))
(1 2)
```

Задача 8.32 ante-del.lisp

Удалите из списка перед каждым вхождением x один элемент, если такой имеется и отличен от x.

Решение 8.32.1

```
(defun ante-del (a w)
  (cond ((null w) nil)
        ((and (equalp (cadr w) a) (not (equalp (car w) a)))
         (ante-del a (cdr w)))
        ((cons (car w) (ante-del a (cdr w))))))

> (ante-del 7 '(7 2 7 7 3))
(7 7 7 3)
```

Решение 8.32.2

```
(defun ante-del (a w)
  (when w (if (and (equalp (cadr w) a) (not (equalp (car w) a)))
              (ante-del a (cdr w))
              (cons (car w) (ante-del a (cdr w))))))

> (ante-del 7 '(7 2 7 7 3))
(7 7 7 3)
```

Задача 8.33 list-a .lisp

Дан список (1 2 3 4) и число 5. Определить рекурсивную функцию, которая помещает 5 в конец списка (1 2 3 4).

Решение 8.33.1

```
(defun list-a (w a)
   (if (null w) `(,a) (cons (car w) (list-a (cdr w) a))))

> (list-a '(1 2 3 4) 5)
(1 2 3 4 5)
```

Задача 8.34 downsize.lisp

Дан список (1 2 3 4 5 6 7). Необходимо составить новый список из элементов, имеющих нечетные порядковые номера, уменьшенных в n-раз.

Решение 8.34.1

```
(defun downsize (n w)
  (when w (cons (/ (car w) n) (downsize n (cddr w)))))

> (downsize 3 '(1 2 3 4 5 6 7))
(1/3 1 5/3 7/3)
```

Задача 8.35 drop-num.lisp

Дан список и позиция элемента. Удалить элемент с заданным номером.

Решение 8.35.1

```
(defun drop-num (n w)
  (cond ((null w) nil)
        ((zerop n) (cdr w))
        ((cons (car w) (drop-num (1- n) (cdr w))))))

> (drop-num 3 '(0 1 2 3 4))
(0 1 2 4)
```

Решение 8.35.2

```
(defun drop-num (n w)
  (when w (if (zerop n)
              (cdr w)
              (cons (car w) (drop-num (1- n) (cdr w))))))

> (drop-num 3 '(0 1 2 3 4))
(0 1 2 4)
```

Задача 8.36 dive-atoms.lisp

Написать программу выделения атомов на заданном уровне в произвольном введеном исходном списке. Сформировать из них список вывести его. Атомы, содержащиеся в исходном списке, имеют уровень 0, атомы, содержа-

щиеся в следующем, имеют уровень 1 и т.д.

Решение 8.36.1

```
(defun dive (n w &aux (a (car w)) (d (cdr w)))
   (cond ((null w) nil)
         ((= n 0) w)
         ((atom a) (dive n d))
         ((nconc (dive (1- n) a) (dive n d)))))

(defun dive-atoms (n w)
  (remove-if #'listp (dive n w)))

> (dive-atoms 0 '(a (b (c 1) d e) f ((2 (g)) 3)))
(A F)
> (dive-atoms 1 '(a (b (c 1) d e) f ((2 (g)) 3)))
(B D E 3)
> (dive-atoms 2 '(a (b (c 1) d e) f ((2 (g)) 3)))
(C 1 2)
> (dive-atoms 3 '(a (b (c 1) d e) f ((2 (g)) 3)))
(G)
> (dive-atoms 4 '(a (b (c 1) d e) f ((2 (g)) 3)))
NIL
```

Решение 8.36.2

```
(defun dive (n w)
   (cond ((null w) nil)
         ((= n 0) w)
         ((atom (car w)) (dive n (cdr w)))
         ((append (dive (1- n) (car w)) (dive n (cdr w))))))

(defun atoms (w)
  (cond ((null w) nil)
        ((atom (car w)) (cons (car w) (atoms (cdr w))))
        ((atoms (cdr w)))))

(defun dive-atoms (n w)
  (atoms (dive n w)))

> (dive-atoms 0 '(a (b)))
(A)
> (dive-atoms 1 '(a (b)))
(B)
```

Решение 8.36.3

```
(defun dive-atoms (n w)
  (labels ((atoms (w)
             (cond ((null w) nil)
                   ((atom (car w))
                    (cons (car w) (atoms (cdr w))))
                   ((atoms (cdr w))))))
```

```
    (atoms (labels ((dive (n w)
                      (cond ((null w) nil)
                            ((= n 0) w)
                            ((atom (car w)) (dive n (cdr w)))
                            ((append (dive (1- n) (car w))
                                     (dive n (cdr w)))))))
             (dive n w)))))

> (dive-atoms 0 '(a (b)))
(A)
> (dive-atoms 1 '(a (b)))
(B)
```

Решение 8.36.4

```
(defun dive-atoms (n w)
  (labels ((atoms (w)
             (remove-if #'listp w)))
    (atoms (labels ((dive (n w)
                      (cond ((null w) nil)
                            ((= n 0) w)
                            ((atom (car w)) (dive n (cdr w)))
                            ((append (dive (1- n) (car w))
                                     (dive n (cdr w)))))))
             (dive n w)))))

> (dive-atoms 0 '(a (b)))
(A)
> (dive-atoms 1 '(a (b)))
(B)
```

Решение 8.36.5

```
(defun dive-atoms (n w)
  (remove-if #'listp
             (labels ((dive (n w)
                        (cond ((null w) nil)
                              ((= n 0) w)
                              ((atom (car w)) (dive n (cdr w)))
                              ((append (dive (1- n) (car w))
                                       (dive n (cdr w)))))))
               (dive n w))))

> (dive-atoms 0 '(a (b)))
(A)
> (dive-atoms 1 '(a (b)))
(B)
```

Решение 8.36.6

```
(defun dive-atoms (n w)
  (cond ((null w) nil)
        ((= n 0) (labels ((atoms (w)
```

```
                              (cond ((null w) nil)
                                    ((atom (car w))
                                     (cons (car w) (atoms (cdr w))))
                                    ((atoms (cdr w))))))
                      (atoms w)))
        ((atom (car w)) (dive-atoms n (cdr w)))
        ((append (dive-atoms (1- n) (car w))
                 (dive-atoms n (cdr w))))))

> (dive-atoms 0 '(a (b)))
(A)
> (dive-atoms 1 '(a (b)))
(B)
```

Решение 8.36.7

```
(defun dive-atoms (n w)
  (cond ((null w) nil)
        ((= n 0) (remove-if #'listp w))
        ((atom (car w)) (dive-atoms n (cdr w)))
        ((nconc (dive-atoms (1- n) (car w))
                (dive-atoms n (cdr w))))))

> (dive-atoms 0 '(a (b)))
(A)
> (dive-atoms 1 '(a (b)))
(B)
```

Задача 8.37 member-down.lisp

Определить функцию (f a w), где a – s-выражение, a w – список чисел, которая уменьшает числа списка w на единицу, если число a является элементом списка w, и возвращает исходный список в противном случае.

Решение 8.37.1

```
(defun member-down (a x)
  (if (member a x) (mapcar #'1- x) x))

> (member-down 2 '(1 2 3 4 5 6))
(0 1 2 3 4 5)
> (member-down 0 '(1 2 3 4 5 6))
(1 2 3 4 5 6)
```

Задача 8.38 expt-elms.lisp

Определить функцию возведения в квадрат списка чисел.

Решение 8.38.1

```
(defun expt-elms (w f)
```

```
  (when w (cons (funcall f (car w)) (expt-elms (cdr w) f))))

(defun x^2 (x) (* x x))

> (expt-elms '(1 2 3) #'x^2)
(1 4 9)
```

Решение 8.38.2

```
(defun expt-elms (w)
  (when w (cons (expt (car w) 2) (expt-elms (cdr w)))))

> (expt-elms '(1 2 3))
(1 4 9)
```

Решение 8.38.3

```
(defun expt-elms (w)
  (mapcar #'(lambda (a) (expt a 2)) w))

> (expt-elms '(1 2 3))
(1 4 9)
```

Задача 8.39 swing-elms.lisp

Написать программу, которая позволит увеличить значение всех положительных элементов одноуровневого числового списка на заданное число и уменьшить значения всех отрицательных элементов того же массива на то же число.

Решение 8.39.1

```
(defun swing-elms (w n &aux (a (car w)))
  (cond ((null w) nil)
        ((plusp a) (cons (+ a n) (swing-elms (cdr w) n)))
        ((minusp a) (cons (- a n) (swing-elms (cdr w) n)))
        ((cons a (swing-elms (cdr w) n)))))

> (swing-elms '(-1 0 1) 1)
(-2 0 2)
```

Решение 8.39.2

```
(defun swing-elms (w n &aux (a (car w)))
  (when w (cons (cond ((zerop a) a)
                      ((plusp a) (+ a n))
                      ((- a n)))
                (swing-elms (cdr w) n))))

> (swing-elms '(-1 0 1) 1)
(-2 0 2)
```

Задача 8.40 knapsack.lisp

Дан упорядоченный по убыванию список целых положительных чисел и некоторое число. Необходимо найти последовательности списка, сумма элементов которых равна этому числу, т.н. задача о ранце - название своё получила от максимизационной задачи укладки как можно большего числа ценных вещей в рюкзак при условии, что общий объём (или вес) всех предметов, способных поместиться в рюкзак, ограничен.

Решение 8.40.1

```
(defun knapsack (w n)
  (mapcar #'reverse (remove-duplicates (knap w n) :test #'equal)))

(defun knap (w n &optional acc ac
             &aux (a (car w)) (d (cdr w)) (z (ap-ply #'+ ac)))
  (cond ((null w) acc)
        ((= (+ a z) n)(knap d n (cons (cons a ac) acc) ac))
        ((knap d n (knap d n acc ac)
               (if (< (+ a z) n) (cons a ac) ac)))))
```

```
> (knapsack '(9 8 7 6 5 4 3 2 1) 13)
((9 3 1) (9 4) (8 4 1) (8 3 2) (8 5) (7 5 1) (7 4 2) (7 3 2 1) (7 6)
(6 5 2) (6 4 2 1) (6 4 3) (5 4 3 1))
```

Решение 8.40.2

```
(defun knapstack (w n &optional acc ac
                  &aux (a (car w)) (d (cdr w)) (b (cons a ac))
                  (z (apply #'+ ac)))
  (cond ((null w) acc)
        ((= (+ a z) n)
         (knapstack d n (if (member (reverse b) acc
                                    :test #'equal)
                            acc
                            (cons (reverse b) acc))
                    ac))
        ((knapstack d n (knapstack d n acc ac)
                    (if (< (+ a z) n) b ac)))))
```

```
> (knapstack '(9 8 7 6 5 4 3 2 1) 13)
((9 3 1) (9 4) (8 4 1) (8 3 2) (8 5) (7 5 1) (7 4 2) (7 3 2 1) (7 6)
(6 5 2) (6 4 2 1) (6 4 3) (5 4 3 1))
```

Решение 8.40.3

```
(defun knapstack (w n &optional acc ac
                  &aux (a (car w)) (d (cdr w)) (b (cons a ac))
                  (z (apply #'+ ac)))
  (cond ((null w) acc)
        ((= (+ a z) n)
         (knapstack d n (let ((c (reverse b)))
```

```
                          (if (member c acc :test #'equal)
                              acc
                              (cons c acc)))
                          ac))
         ((knapstack d n (knapstack d n acc ac)
                     (if (< (+ a z) n) b ac)))))

> (knapstack '(9 8 7 6 5 4 3 2 1) 13)
((9 3 1) (9 4) (8 4 1) (8 3 2) (8 5) (7 5 1) (7 4 2) (7 3 2 1) (7 6)
(6 5 2) (6 4 2 1) (6 4 3) (5 4 3 1))
```

Решение 8.40.4

```
(defun knapstack (w n &optional acc ac
                  &aux (a (car w)) (d (cdr w)) (b (cons a ac))
                  (z (apply #'+ ac)))
  (if w
      (if (= (+ a z) n)
          (knapstack d n (let ((c (reverse b)))
                           (if (member c acc :test #'equal)
                              acc
                              (cons c acc)))
                     ac)
          (knapstack d n (knapstack d n acc ac)
                     (if (< (+ a z) n) b ac)))
      acc))

> (knapstack '(9 8 7 6 5 4 3 2 1) 13)
((9 3 1) (9 4) (8 4 1) (8 3 2) (8 5) (7 5 1) (7 4 2) (7 3 2 1) (7 6)
(6 5 2) (6 4 2 1) (6 4 3) (5 4 3 1))
```

Решение 8.40.5

```
(defun knapstack (w n &optional acc ac
                  &aux (a (car w)) (d (cdr w)) (b (cons a ac))
                  (z (apply #'+ ac)))
  (if w (knapstack d n (if (= (+ a z) n)
                           (let ((c (reverse b)))
                             (if (member c acc :test #'equal)
                                 acc
                                 (cons c acc)))
                           (knapstack d n acc ac))
                    (if (< (+ a z) n) b ac))
      acc))

> (knapstack '(9 8 7 6 5 4 3 2 1) 13)
((9 3 1) (9 4) (8 4 1) (8 3 2) (8 5) (7 5 1) (7 4 2) (7 3 2 1) (7 6)
(6 5 2)(6 4 2 1) (6 4 3) (5 4 3 1))
```

Задача 8.41 insert-atom.lisp

Реализовать функцию (insert atom list), возвращающую список list, в котором в начало каждого подсписка добавлен атом atom.
Пример: (insert-atom 'a '((b c) (b (c d)) (c d) nil))
((A B C) (A B (A C D)) (A C D) (A))

Решение 8.41.1

```
(defun insert-atom (a w)
  (mapcar #'(lambda (e) (if (atom e)
                            (if e e (list a))
                            (cons a (insert-atom a e)))) w))

> (insert-atom 'a '((b c) (b (c d)) (c d) nil))
((A B C) (A B (A C D)) (A C D) (A))
```

Решение 8.41.2

```
(defun insert-atom (a w)
  (mapcar #'(lambda (e) (if (listp e)
                            (cons a (insert-atom a e))
                            e))
                    w))

> (insert-atom 'a '((b c) (b (c d)) (c d) nil))
((A B C) (A B (A C D)) (A C D) (A))
```

Решение 8.41.3

```
(defun insert-atom (a w)
  (cond ((null w) nil)
        ((listp (car w))
         (cons (cons a (car w)) (insert-atom a (cdr w))))
        ((cons (car w) (insert-atom a (cdr w))))))

> (insert-atom 'a '((b c) (b (c d)) (c d) nil))
((A B C) (A B (A C D)) (A C D) (A))
```

Задача 8.42 purgatory.lisp

Удалите из списка перед каждым вхождением n один элемент, если такой имеется и отличен от n.

Решение 8.42.1

```
(defun purgatory (n w)
  (cond ((null w) nil)
        ((and (eq (cadr w) n) (not (eq (car w) (cadr w))))
         (purgatory n (cdr w)))
        ((cons (car w) (purgatory n (cdr w))))))

> (purgatory 1 '(1 2 1 1 3))
(1 1 1 3)
```

Задача 8.43 not-dictionary.lisp

Дано предложение и словарь. Определить функцию, возвращающую слова предложения, которые не входят в словарь.

Решение 8.43.1

```
(defun not-dictionary (s v)
  (remove-duplicates
   (loop for a in
         (read-from-string
          (concatenate
           'string "("
           (delete-if-not #'(lambda (x)
                              (or (alpha-char-p x)
                                  (equal x #\space)
                                  (equal x #\-)))
                          s) ")"))
         unless (member a v) collect a)))
```

```
> (not-dictionary "He has plundered our seas, ravaged our coasts, burnt our towns, and destroyed the lives of our people." '(plundered coasts towns destroyed lives people))
(HE HAS SEAS RAVAGED BURNT AND THE OF OUR)
```

Задача 8.44 zeka.lisp

Определить функцию (f n m), где n < m, которая сначала выводит строку чисел без пробелов n,n+1,n+2…m, а затем на другой строке выводит значение в виде списка чисел (n n+1 n+2…m).

Решение 8.44.1

```
(defun zeka (n m)
  (mapcar #'princ (loop for a from n to m collect a)))
```

```
> (zeka 1 5)
12345
(1 2 3 4 5)
```

Решение 8.44.1

```
(defun zeka (n m)
  (loop for a from n to m do (princ a) collect a))
```

```
> (zeka 1 5)
12345
(1 2 3 4 5)
```

Задача 8.45 cska.lisp

Определить функцию (f n m), где n < m, которая сначала выводит числа в колонку n,n+1,n+2…m, а затем с новой строки выводит значение в виде списка чисел (n n+1 n+2…m).

Решение 8.45.1

```
(defun cska (n m)
  (loop for a from n to m
        do (format t "~a~%" a)
        collect a))

> (cska 1 5)
1
2
3
4
5
(1 2 3 4 5)
```

Решение 8.45.1

```
(defun cska (n m)
  (loop for a from n to m do (print a) collect a))

> (cska 1 5)
1
2
3
4
5
(1 2 3 4 5)
```

Задача 8.46 inserta.lisp

Используя функцию DEFUN определить функцию f (a w), где a – s-выражение, a w - список, состоящий из 2-3 элементов, которая вставляет A между первым и вторым элементом списка w, если третий элемент равен нулю. В остальных случаях возвращает исходный список w.

Решение 8.46.1

```
(defun inserta (a w)
  (if (zerop (caddr w)) (cons (car w) (cons a (cdr w))) w))

> (inserta 'v '(1 2 0))
(1 V 2 0)
```

Задача 8.47 dump-levels.lisp

Написать функцию, удаляющую из исходного списка заданное число подсписков заданной глубины.

Решение 8.47.1

```
(defun dump-levels (w n m)
  (labels ((next (w n)
             (cond ((null w) nil)
                   ((atom (car w)) (cons (car w) (next (cdr w) n)))
                   ((and (= n 0) (plusp m))
                    (setf m (1- m)) (next (cdr w) n))
                   ((cons (next (car w) (1- n)) (next (cdr w) n))))))
    (next w n)))

> (dump-levels '(5 (4 (6)) (1 7) (8 (3 (5)))) 0 1)
(5 (1 7) (8 (3 (5))))
> (dump-levels '(5 (4 (6)) (1 7) (8 (3 (5)))) 1 1)
(5 (4) (1 7) (8 (3 (5))))
```

Задача 8.48 keys+.lisp

Написать функцию, выбирающую из ассоциативного списка все значения, которые больше введенного числа. Собрать в список их ключи.

Решение 8.48.1

```
(defun keys+ (w n)
  (mapcar #'cdr (remove-if #'(lambda (a) (<= a n)) w :key #'cdr)))

> (keys+ '((a . 1) (b . 2) (c . 3) (d . 4) (e . 5) (f . 6) (g . 7)) 4)
(5 6 7)
```

Решение 8.48.2

```
(defun keys+ (w n)
  (mapcar #'cdr (remove-if-not #'(lambda (a) (> a n)) w :key #'cdr)))

> (keys+ '((a . 1) (b . 2) (c . 3) (d . 4) (e . 5) (f . 6) (g . 7)) 4)
(5 6 7)
```

Задача 8.49 drop-n.lisp

Удалить из списка n-й элемент.

Решение 8.49.1

```
(defun drop-n (n w)
  (cond  ((null w) nil)
         ((= n 2) (cons (car w) (cddr w)))
         ((cons (car w) (drop-n (1- n) (cdr w))))))

> (drop-n 14 '(1 2 3 4 5 6 7 8 9 10 11 12 13))
(1 2 3 4 5 6 7 8 9 10 11 12 13)
```

```
> (drop-n 14 '(1 2 3 4 5 6 7 8 9 10 11 12 13 14))
(1 2 3 4 5 6 7 8 9 10 11 12 13)
R> (drop-n 14 '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15))
(1 2 3 4 5 6 7 8 9 10 11 12 13 15)
```

Задача 8.50 with-pairs.lisp

Дан одноуровневый числовой список с четным числом элементов. Определить функционал, который применяет функцию последовательно к каждым двум соседним элементам списка.

Решение 8.50.1

```
(defun with-pairs (f w)
  (when w (cons (funcall f (car w) (cadr w))
                (with-pairs f (cddr w)))))

> (with-pairs #'list '(1 2 3 4 5 6))
((1 2) (3 4) (5 6))
> (with-pairs #'+ '(1 2 3 4 5 6))
(3 7 11)
> (with-pairs #'- '(1 2 3 4 5 6))
(-1 -1 -1)
> (with-pairs #'* '(1 2 3 4 5 6))
(2 12 30)
> (with-pairs #'expt '(1 2 3 4 5 6))
(1 81 15625)
```

Задача 8.51 drop-zerop-rem-a-n.lisp

Дан одноуровневый числовой список с четным числом элементов. Определить функционал, который применяет функцию последовательно к каждым двум соседним элементам списка.

Решение 8.51.1

```
(defun drop-zerop-rem-a-n (w n &aux (a (car w)))
  (cond ((null w) nil)
        ((zerop (rem a n)) (drop-zerop-rem-a-n (cdr w) n))
        ((cons a (drop-zerop-rem-a-n (cdr w) n)))))

> (drop-zerop-rem-a-n '(1 2 3 4 5 6 7 8 9 10) 3)
(1 2 4 5 7 8 10)
```

Решение 8.51.2

```
(defun drop-zerop-rem-a-n (w n &aux (a (car w)))
  (when w (if (zerop (rem a n))
              (drop-zerop-rem-a-n (cdr w) n)
              (cons a (drop-zerop-rem-a-n (cdr w) n)))))
```

```
> (drop-zerop-rem-a-n '(1 2 3 4 5 6 7 8 9 10) 3)
(1 2 4 5 7 8 10)
```

Решение 8.51.3

```
(defun drop-zerop-rem-a-n (w n)
  (loop for a in w
        unless (zerop (rem a n))
        collect a))

> (drop-zerop-rem-a-n '(1 2 3 4 5 6 7 8 9 10) 3)
(1 2 4 5 7 8 10)
```

Решение 8.51.4

```
(defun drop-zerop-rem-a-n (w n)
  (remove-if #'(lambda (a) (zerop (rem a n))) w))

> (drop-zerop-rem-a-n '(1 2 3 4 5 6 7 8 9 10) 3)
(1 2 4 5 7 8 10)
```

Задача 8.52 del-upto-a.lisp

Определить функцию, удаляющую начало списка до заданного элемента (включительно).

Решение 8.52.1

```
(defun del-upto-a (a w)
  (cond ((null w) nil)
        ((equal a (car w)) (cdr w))
        ((del-upto-a a (cdr w)))))

> (del-upto-a 'a '((a b) c a b c))
(B C)
```

Решение 8.52.2

```
(defun del-upto-a (a w)
  (cond ((null w) nil)
        ((equal a (car w)) (cdr w))
        ((del-upto-a a (cdr w)))))

> (del-upto-a 'a '((a b) c a b c))
(B C)
```

Задача 8.53 n-elm.lisp

Создать список, в котором каждый из элементов исходного списка повторяется n раз.

Решение 8.53.1

```
(defun n-elm (n w &optional (m n))
  (cond ((null w) nil)
        ((= m 1) (cons (car w) (n-elm n (cdr w) n)))
        (t (cons (car w) (n-elm n w (1- m))))))

> (n-elm 3 '(a b c))
(A A A B B B C C C)
```

Задача 8.54 _every.lisp

Определите функциональный предикат, который истинен в том и только в том случае, когда являющийся функциональным аргументом предикат истинен для всех элементов списка.

Решение 8.54.1

```
(defun _every (p w)
  (cond ((null w) t)
        ((funcall p (car w)) (_every p (cdr w)))
        (t nil)))

> (_every #'numberp '(1 2 3))
T
> (_every #'numberp '(1 2 a))
NIL
> (_every #'numberp '())
T
```

Решение 8.54.2

```
(defun _every (p w)
  (if w (when (funcall p (car w)) (_every p (cdr w))) t))

> (_every #'numberp '(1 2 3))
T
> (_every #'numberp '(1 2 a))
NIL
> (_every #'numberp '())
T
```

Решение 8.54.3

```
CL-USER> (defpackage :good (:use :common-lisp))
#<PACKAGE GOOD>
CL-USER> (in-package :good)
#<PACKAGE GOOD>
GOOD> (defun every (p w)
  (if w (when (funcall p (car w)) (every p (cdr w))) t))
EVERY
GOOD> (every #'numberp '(1 2 3))
```

```
T
```

Задача 8.55 sum-n-elms.lisp

Найти сумму первых n положительных элементов одномерного числового массива.

Решение 8.55.1

```
(defun sum-n-elms (w n)
  (cond ((zerop n) 0)
        ((plusp (car w)) (+ (car w) (sum-n-elms  (cdr w) (1- n))))
        (t (sum-n-elms (cdr w) n))))

> (sum-n-elms '(-10 -10 1 2 3 4) 2)
3
> (sum-n-elms '(-10 -10 1 2 3 4) 4)
10
```

Решение 8.55.1 (автор - nullxdth, www.cyberforum.ru)

```
(defun sum-n-elms (w n)
  (reduce #'+ (remove-if-not #'plusp w) :end n))

> (sum-n-elms '(-10 -10 1 2 3 4) 2)
3
> (sum-n-elms '(-10 -10 1 2 3 4) 4)
10
```

Задача 8.56 cut-n-elms.lisp

Напишите функцию, "обрывающуя" список, если он состоит более чем из n элементов.

Решение 8.56.1

```
(defun cut-n-elms (n w)
  (cond ((or (zerop n) (null w)) nil)
        (t (cons (car w) (cut-n-elms (1- n) (cdr w))))))

> (cut-n-elms 4 '(1 -2 3 -4 -7 4 5))
(1 -2 3 -4)
```

Решение 8.56.2

```
(defun cut-n-elms (n w)
  (when (and w (> n 0))
    (cons (car w) (cut-n-elms (1- n) (cdr w)))))

> (cut-n-elms 4 '(1 -2 3 -4 -7 4 5))
(1 -2 3 -4)
```

Решение 8.56.3

```
(defun cut-n-elms (n w)
  (when (and (plusp n) w)
    (cons (car w) (cut-n-elms (1- n) (cdr w)))))

> (cut-n-elms 4 '(1 -2 3 -4 -7 4 5))
(1 -2 3 -4)
```

Задача 8.57 drop<n.lisp

Определите функцию, зависящую от двух аргументов u и n, которая по данному списку строит список его элементов, встречающихся в нем не менее n раз.

Решение 8.57.1

```
(defun drop<n (w n)
  (remove-duplicates
   (remove-if #'(lambda (a) (< (count a w) n)) w)))

> (drop<n '(40 2 8 9 4 2 8 9 4 1 4 1 2 8 7 9 7) 2)
(4 1 2 8 9 7)
```

Решение 8.57.2

```
(defun drop<n (w n &aux (v (remove-duplicates w)))
  (remove-if #'(lambda (a) (< (count a w) n)) v))

> (drop<n '(40 2 8 9 4 2 8 9 4 1 4 1 2 8 7 9 7) 2)
(4 1 2 8 9 7)
```

Задача 8.58 streak.lisp

Из произвольного списка выделить серии стоящих подряд одинаковых элементов.

Решение 8.58.1

```
(defun streak (w n &optional (b (car w)) ac &aux (a (car w)))
  (cond ((null w) (when (>= (length ac) n) (list ac)))
        ((eql a b) (streak (cdr w) n b (cons b ac)))
        ((< (length ac) n) (streak (cdr w) n a (list a)))
        ((cons ac (streak (cdr w) n a (list a))))))

> (streak '(1 2 2 2 3 4 5 5 5 6 7 7 7 7 7 8) 2)
((2 2 2) (5 5 5) (7 7 7 7 7))
> (streak '(1 2 2 2 3 4 5 5 5 6 7 7 7 7 7 8) 4)
((7 7 7 7 7))
```

Решение 8.58.2

```
(defun streak (w n &optional (b (car w)) ac
               &aux (a (car w)) (p (>= (length ac) n)))
  (cond ((null w) (when p (list ac)))
        ((eql a b) (streak (cdr w) n b (cons b ac)))
        (p (cons ac (streak (cdr w) n a (list a))))
        ((streak (cdr w) n a (list a)))))
```

```
> (streak '(1 2 2 2 3 4 5 5 5 6 7 7 7 7 7 8) 2)
((2 2 2) (5 5 5) (7 7 7 7 7))
> (streak '(1 2 2 2 3 4 5 5 5 6 7 7 7 7 7 8) 4)
((7 7 7 7 7))
```

Структура 9 (функция (СПИСОК) (СПИСОК)) > АТОМ

Задача 9.1 count-xpr.lisp

Значение функции равно количеству вхождений заданного s-выражения в список. Если s-выражение не принадлежит списку или список пустой, то значение функции nil.

Решение 9.1.1

```
(defun count-xpr (e w)
  (when w (cond ((equal e (car w)) (1+ (or (count-xpr e (cdr w)) 0)))
                ((count-xpr e (cdr w))))))
```

```
> (cnt '(a 1) '((a 1) (a 1)))
2
```

Задача 9.2 equalsetp.lisp

Написать функию, которая проверяет равны ли два множества, представленные двумя списками v и w, которые заданы в качестве параметров.

Решение 9.2.1

```
(defun equalsetp (v w)
  (and (subsetp v w) (subsetp w v)))
```

```
> (equalsetp '(1 1 2 3) '(1 2 2 3))
T
```

Решение 9.2.2

```
(defun equalsetp (v w)
  (and (subp v w) (subp w v)))

(defun subp (v w)
```

```
  (cond ((null v))
        ((member (car v) w) (subp (cdr v) w))
        (nil)))

> (equalsetp '(1 1 2 3) '(1 2 2 3))
T
```

Задача 9.3 every>.lisp

Решить задачу на параллельную рекурсию: определить предикат, проверяющий, что все элементы первого числового списка больше элементов второго списка.

Решение 9.3.1

```
defun every> (w v)
  (cond ((null w) t)
        ((not (ev (car w) v)) nil)
        ((every> (cdr w) v))))

(defun ev (a v)
  (cond ((null v) t)
        ((< a (car v)) nil)
        ((ev a (cdr v)))))

> (every> '(4 5 6 7) '(1 2 3))
T
> (every> '(4 1 6 7) '(1 2 3))
NIL
```

Решение 9.3.2

```
(defun every> (w v)
  (cond ((null w) t)
        ((some #'(lambda (e) (< (car w) e)) v) nil)
        ((every> (cdr w) v))))

> (every> '(4 5 6 7) '(1 2 3))
T
> (every> '(4 1 6 7) '(1 2 3))
NIL
```

Решение 9.3.3

```
(defun every> (w v)
  (> (apply #'min w) (apply #'max v)))

> (every> '(4 5 6 7) '(1 2 3))
T
> (every> '(4 1 6 7) '(1 2 3))
NIL
```

Задача 9.4 same-elm&pos.lisp

Даны два списка, проверить стоят ли одинаковые элементы списков на одинаковых позициях. Выводить T, если все одинаковые элементы стоят на одинаковых позициях, и N, если хотя бы одна пара одинаковых элементов стоит на разных позициях.

Решение 9.4.1

```
(defun same-elm&pos (v w)
  (cond ((null v) t)
        ((or (member (car v) (cdr w))(member (car w) (cdr v))) nil)
        ((same-elm&pos (cdr v) (cdr w)))))

> (same-elm&pos '(1 2 3 4) '(1 7 8 5))
T
> (same-elm&pos '(1 2 3 4) '(7 1 8 5))
NIL
> (same-elm&pos '(1 2 3 4) '(1 7 1 5))
NIL
> (same-elm&pos '(1 2 3 4) '(2 5 6 7))
NIL
```

Комментарии (построчно):
1. Функция tw (список список);
2. Условие (добрались до конца списка, т.е. список v - пуст) > возвращаем истина;
3. Условие (голова первого списка входит в хвост второго списка или наоборот) > возвращаем ложь;
4. Если не выполнено ни одно из условий > вызываем функцию tw с хвостами списков.

Задача 9.5 over-atom.lisp

Определить функцию с двумя списками в качестве аргументов. Расположение скобок в s-выражениях этих списков одинаковое, т.е. (3 2 (6) 1) и (4 5 (7) 2). Значение функции - t, если каждый атом из первого списка больше соответствующего атома из второго списка, и nil в противном случае.

Решение 9.5.1

```
(defun over-atom (w v)
  (cond ((null w) t)
        ((listp (car w)) (over-atom (car w) (car v)))
        ((> (car w) (car v)) (over-atom (cdr w) (cdr v)))))

> (over-atom '(5 6 (7) 8) '(1 2 (3) 4))
T
```

Решение 9.5.2

```
(defun over-atom (w v)
  (every #'(lambda (m n)
             (cond ((listp m)(over-atom m n))
                   ((> m n))))
         w v))

> (over-atom '(5 6 (7) 8) '(1 2 (3) 4))
T
```

Решение 9.5.3

```
(defun over-atom (w v)
  (every #'(lambda (m n) (if (listp m) (over-atom m n) (> m n))) w v))

> (over-atom '(5 6 (7) 8) '(1 2 (3) 4))
T
```

Задача 9.6 same-order.lisp

Реализовать функцию, возвращающую T при идентичности порядка расположения одинаковых атомов в исходных списках

Решение 9.6.1

```
(defun same-order (w v)
  (cond ((null w) t)
        ((eq (car w) (car w)) (same-order (cdr w) (cdr v)))
        ((member (car w) v) nil)
        ((same-order (cdr w) (cdr v)))))

> (same-order '(a b) '(a c))
T
```

Задача 9.7 sublistp.lisp

Написать функцию аргументов w v, возвращающую t, если v является подсписком w. Элементами списков могут быть атомы и списки любой вложенности.

Решение 9.7.1

```
(defun sublistp (w v)
  (when w ((lambda (a d)
             (if (atom a)
                 (sublistp d v)
                 (or (equal a v) (sublistp a v) (sublistp d v))))
           (car w) (cdr w))))

> (sublistp '(((j k) c (r p)) (u (r (m t) (a z) d) z)) '(a z))
T
```

Решение 9.7.2

```
(defun sublistp (w v)
  (when w ((lambda (a d)
             (cond ((atom a)(sublistp d v))
                   ((or (equal a v) (sublistp a v) (sublistp d v)))))
           (car w) (cdr w))))

> (sublistp '(((j k) c (r p)) (u (r (m t) (a z) d) z)) '(a z))
T
```

Решение 9.7.3

```
(defun sublistp (v w)
  (cond ((null w) nil)
        ((equal v (car w)) t)
        ((atom (car w)) (sublistp v (cdr w)))
        ((or (sublistp v (car w)) (sublistp v (cdr w))))))

> (sublistp '(z z) '(a (a (a a) (z z))a))
T
```

Задача 9.8 more-elements.lisp

Написать программу, которая сравнивает два списка по количеству элементов.

Решение 9.8.1

```
(defun more-elements (w v)
  (cond ((and w (null v)) 'first)
        ((and v (null w)) 'second)
        ((null w) 'equal)
        ((more-elements (cdr w) (cdr v)))))

> (more-elements '(1) '(1 2))
SECOND
```

Задача 9.9 more-atoms.lisp

Написать программу, которая сравнивает два списка по количеству атомов.

Решение 9.9.1

```
(defun more-elements (w v)
  (cond ((and w (null v)) 'first)
        ((and v (null w)) 'second)
        ((null w) 'equal)
        ((more-elements (cdr w) (cdr v)))))
```

```
(defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun more-atoms (w v)
  (more-elements (flat w) (flat v)))

> (more-atoms '(1 2) '((1 2) 3))
SECOND
```

Задача 9.10 subset-next.lisp

Даны два списка элементов типа symbol. Если v есть подсписок w, то определить следующий после v элемент списка w, иначе ответом является слово "no".

Решение 9.10.1

```
(defun subset-next (v w)
  (cond ((null w) 'no)
        ((equal v (car w)) (cadr w))
        ((subset-next v (cdr w)))))

> (subset-next '(b c) '(a (b c) d))
D
```

Задача 9.11 maxum.lisp

Даны два списка чисел произвольной длины. Определить функцию которая вычисляет сумму элементов каждого списка и выводит наибольшую из них.

Решение 9.11.1

```
(defun maxum (w v)
  (max (reduce #'+ w) (reduce #'+ v)))

> (maxum '() '())
0
> (maxum '() '(1))
1
> (maxum '(1 1) '(1))
2
```

Решение 9.11.1

```
(defun sum (w)
  (if (null w) 0 (+ (car w) (sum (cdr w)))))

(defun mx (a b)
  (if (>= a b) a b))
```

```
(defun maxum (w v)
  (mx (sum w) (sum v)))

> (maxum '() '())
0
> (maxum '() '(1))
1
> (maxum '(1 1) '(1))
2
```

Задача 9.12 miss.lisp

Определите функцию, возвращающую первый совпавший в двух списках элемент, либо nil, если таких элементов нет.

Решение 9.12.1

```
(defun miss (w v)
  (cond ((null w) nil)
        ((eq (car w) (car v)) (car w))
        ((miss (cdr w) (cdr v)))))

> (miss '(1 2) '(0 2))
2
```

Задача 9.13 reversed.lisp

Определить функцию, проверяющую является ли один список результатом реверсирования другого списка.

Решение 9.13.1

```
(defun reversed (w v &optional z)
  (if w (reversed (cdr w) v (cons (car w) z)) (equal z v)))

> (reversed '(1 2 3) '(3 2 1))
T
> (reversed '(1 2 3) '(1 2 3))
NIL
```

Задача 9.14 max-sum-pair.lisp

Определить функцию, находящую наибольшее значение xi+yi – суммы элементов произвольных массивов одинаковой размерности.

Решение 9.14.1

```
(defun max-pair (w v)
  (reduce #'max (map 'array #'+ w v)))
```

```
> (max-pair #(1 2 3) #(1 1 1))
4
```

Решение 9.14.2 * для списков

```
(defun max-sum-pair (w v)
  (reduce #'max (mapcar #'+ w v)))

> (max-sum-pair '(0 1) '(1 1))
2
```

Задача 9.15 birthday-earlier.lisp

Известны даты рождения двух человек. Определить, кто из них младше.

Решение 9.15.1

```
(defun encode-date (n d m y)
  (list n (encode-universal-time 0 0 1 d m y)))

(defun birthday-earlier (v w)
  (if (<= (cadr (apply #'encode-date v))
          (cadr (apply #'encode-date w)))
      (car v)
      (car w)))

> (birthday-earlier '(ann 1 1 1990) '(rob 1 1 1991))
ANN
```

Структура 10 (функция (СПИСОК) (СПИСОК)) > (СПИСОК)

Задача 10.1 uni.lisp

Написать рекурсивную функцию, реализующую объединение двух множеств w и v. Пример: (объединение '(1 2 3) '(1 2 4) есть (1 2 3 4).

Решение 10.1.1

```
(defun uni (w v)
  (cond ((null w) v)
        ((member (car w) v) (uni (cdr w) v))
        ((cons (car w) (uni (cdr w) v)))))

> (uni '(1 2 3) '(1 2 4))
(3 1 2 4)
```

Решение 10.1.2

```
(defun uni (w v)
  (sort (union w v) #'<))
```

```
> (uni '(1 2 3) '(1 2 4))
(1 2 3 4)
```

Решение 10.1.3

```
(defun uni (w v)
  (delete-duplicates (merge 'list w v #'<)))
```

```
> (uni '(1 2 3) '(1 2 4))
(1 2 3 4)
```

Задача 10.2 priority-sort.lisp

Даны 8 троек - 0 0 0, 0 0 1, 0 1 0, 0 1 1, 1 0 0, 1 0 1, 1 1 0, 1 1 1. Рассортировать их с разными приоритетами для столбцов.

Решение 10.2.1

```
(defun priority-sort (w prs &optional (m (reduce #'max prs)))
  (if (zerop m)
      w
      (priority-sort
       (stable-sort w #'<
                    :key #'(lambda (p)
                             (nth (position m prs) p))) prs (1- m))))
```

```
> (priority-sort '((0 0 0) (0 0 1) (0 1 0) (0 1 1) (1 0 0) (1 0 1) (1
1 0) (1 1 1)) '(1 2 3))
((0 0 0) (0 0 1) (0 1 0) (0 1 1) (1 0 0) (1 0 1) (1 1 0) (1 1 1))
> (priority-sort '((0 0 0) (0 0 1) (0 1 0) (0 1 1) (1 0 0) (1 0 1) (1
1 0) (1 1 1)) '(3 2 1))
((0 0 0) (1 0 0) (0 1 0) (1 1 0) (0 0 1) (1 0 1) (0 1 1) (1 1 1))
```

Задача 10.3 mix-elements.lisp

Написать функцию, которая чередует элементы двух списков: (a b c…)(1 2 3…)>(a 1 b 2 c 3…).

Решение 10.3.1

```
(defun mix-elms (w v)
  (cond ((null w) v) ((null v) w) ((mix w v))))

(defun mix (w v)
  (when w (cons (car w) (mix v (cdr w)))))
```

```
> (mix-elms '(1 2 3) '(a b c))
(1 A 2 B 3 C)
```

Решение 10.3.2 (автор - Catstail, www.cyberforum.ru)

```
(defun mix-elms (w v)
  (cond ((null w) v) ((null v) w)
        ((cons (car w) (cons (car v) (mix-elms (cdr w) (cdr v)))))))

> (mix-elms '(1 2 3) '(a b c))
(1 A 2 B 3 C)
> (mix-elms '(1 2 3) '(a b c d))
(1 A 2 B 3 C D)
> (mix-elms '(1 2 3 4 5) '(a b c d))
(1 A 2 B 3 C 4 D 5)
```

Решение 10.3.3 (автор - Nameless One, www.cyberforum.ru) *корректное лишь при равном числе элементов в списках

```
(defun mix-elms (w v)
  (mapcan #'list w v))

> (mix-elms '(1 2 3) '(a b c))
(1 A 2 B 3 C)
> (mix-elms '(1 2 3) '(a b c d))
(1 A 2 B 3 C)
> (mix-elms '(1 2 3 4 5) '(a b c d))
(1 A 2 B 3 C 4 D)
```

Задача 10.4 alter-mix.lisp

Используя рекурсию преобразовать два исходных списка ((a b) c d e (f) g h (i j k)...) и ((1) 2 3 (4 5 6 7) 8 (9 10)...). Получить ((a b 2 3) (1 c d e) (f 8) (4 5 6 7 g h) (i j k...) (9 10...)...).

Решение 10.4.1

```
(defun alter-mix (w v)
  (glue (thick w) (thick v)))

(defun thick (w &optional ac)
  (cond ((null w) (if ac (cons (nreverse ac) nil) nil))
        ((listp (car w))
         (if ac
             (nconc (list (nreverse ac) (car w))
                    (thick (cdr w) nil))
             (cons (car w) (thick (cdr w) nil))))
        ((push (car w) ac) (thick (cdr w) ac))))

(defun glue (w v)
  (cond ((null w) nil)
        ((nconc
          (list
           (nconc (car w) (cadr v))
           (nconc (car v) (cadr w)))
          (glue (cddr w) (cddr v))))))
```

```
> (alter-mix '((a b) c d e (f) g h (i j k) l m) '((1) 2 3 (4 5 6 7) 8
(9 10) 11 12))
((A B 2 3) (1 C D E) (F 8) (4 5 6 7 G H) (I J K 11 12) (9 10 L M))
```

Решение 10.4.2

```
(defun alter-mix (w v)
  (glue (thick w) (thick v)))

(defun thick (w &optional ac)
  (cond ((null w) (if ac (cons (nreverse ac) nil) nil))
        ((listp (car w))
         (if ac
             (list* (nreverse ac) (car w) (thick (cdr w) nil))
             (cons (car w) (thick (cdr w) nil))))
        ((push (car w) ac) (thick (cdr w) ac))))

(defun glue (w v)
  (cond ((null w) nil)
        ((list* (nconc (car w) (cadr v))
                (nconc (car v) (cadr w))
                (glue (cddr w) (cddr v))))))

> (alter-mix '((a b) c d e (f) g h (i j k) l m) '((1) 2 3 (4 5 6 7) 8
(9 10) 11 12))
((A B 2 3) (1 C D E) (F 8) (4 5 6 7 G H) (I J K 11 12) (9 10 L M))
```

Задача 10.5 cartesian-product.lisp

Написать на Лиспе программу смешивания двух списков. Например вызывать ее так: (mix '(a b c d) '(1 2 3)), и чтобы результат был такой: ((a 1)(a 2)(a 3)(b 1)(b 2)...(d 2)(d 3)). Решить задачу без циклов и дополнительных переменных, пользуясь только рекурсией.

Решение 10.5.1

```
(defun cartesian-product (w v)
  (cond ((null w) nil)
        ((append (mi (car w) v) (cartesian-product (cdr w) v)))))

(defun mi (a w)
  (cond ((null w) nil)
        ((cons (list a (car w)) (mi a (cdr w))))))

> (cartesian-product '(a b c) '(1 2 3))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.5.2

```
(defun cartesian-product (w v)
  (when w (nconc (mi (car w) v) (cartesian-product (cdr w) v))))
```

```
(defun mi (a v)
  (when v (cons (list a (car v)) (mi a (cdr v)))))

> (cartesian-product '(a b c) '(1 2 3))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.5.3

```
(defun cartesian-product (w v)
  (when w
    (append
     (maplist
      #'(lambda (e) (list (car w) (car e))) v)
     (cartesian-product (cdr w) v))))

> (cartesian-product '(a b c) '(1 2 3))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.5.4

```
(defun cartesian-product (w v)
  (apply
   #'nconc
   (mapcar
    #'(lambda (e) (mapcar #'(lambda (a) (list e a)) v)) w)))

> (cartesian-product '(a b c) '(1 2 3))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.5.5 (автор – Max Vasin, www.linux.org.ru)

```
(defun cartesian-product (w v)
  (mapcan (lambda (e) (mapcar (lambda (a) `(,a ,e)) w)) v))

> (cartesian-product '(a b c) '(1 2 3))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Задача 10.6 exxp.lisp

Функция возводит элемент списка w в степень равную элементу из списка v, записывая каждый результат в выходной список. При этом:
Если элемент списка v не является целым числом, а элемент списка w – является, функция возвращает в результирующем списке запись «E1».
Если элемент списка w не является целым числом, а элемент списка v – является, функция возвращает в результирующем списке запись «E2».
Если оба элемента не являются целыми числами, функция возвращает в результирующем списке запись «E3».

Решение 10.6.1

```
(defun exxp (w v)
```

```
  (mapcar #' (lambda (aw av)
               (cond ((and (zerop (mod aw 1)) (/= (mod av 1) 0)) 'e1)
                     ((and (/= (mod aw 1) 0) (zerop (mod av 1))) 'e2)
                     ((and (/= (mod aw 1) 0) (/= (mod av 1) 0)) 'e3)
                     ((expt aw av))))
             w v))

> (exxp '(2 2.1 2.1 2) '(2.1 2 2.1 2))
(E1 E2 E3 4)
```

Задача 10.7 num-v-carvcdrw.lisp

Есть два списка. Если первый элемент списка есть натуральное число, то возвратить второй список, иначе возвратить список из головы второго и хвоста первого.

Решение 10.7.1

```
defun num-v-carvcdrw (w v)
  (cond ((and (integerp (car w)) (plusp (car w))) v)
        ((cons (car v) (cdr w)))))

> (num-v-carvcdrw '(1 2 3) '(a b c))
(A B C)
> (num-v-carvcdrw '(0 2 3) '(a b c))
(A 2 3)
```

Задача 10.8 compare.lisp

Определите функцию, которая из двух списков одинаковой длины строит список из t и nil в результате попарного сравнения элементов, например, в результате вызова (f '(5 3 9 7 4) '(5 2 9 7 8)) выдается ответ (T Nil T T Nil).

Решение 10.8.1

```
(defun compare (w v)
  (cond ((null w) nil)
        ((cons (eq (car w) (car v))
               (compare (cdr w) (cdr v))))))

> (compare '(5 3 9 7 4) '(5 2 9 7 8))
(T NIL T T NIL)
```

Решение 10.8.2

```
(defun compare (w v)
  (mapcar #'eq w v))

> (compare '(5 3 9 7 4) '(5 2 9 7 8))
(T NIL T T NIL)
```

Задача 10.9 mix-wvwv.lisp

Есть два списка w (a b) и v (1 2). Определить функцию, чтобы на выходе получалось (a 1 b 2).

Решение 10.9.1

```
(defun mix-wvwv (w v)
  (cond ((null w) v) ((null v) w) ((mi w v))))

(defun mi (w v)
  (cond (w (cons (car w) (mi v (cdr w))))
        ((when v (nconc v nil)))))
```

```
> (mix-wvwv '(1 2 3 4) '(a b c d))
(1 A 2 B 3 C 4 D)
> (mix-wvwv '(1 2 3 4) '(a b c))
(1 A 2 B 3 C 4)
> (mix-wvwv '(1 2 3 4) '(a b))
(1 A 2 B 3 4)
> (mix-wvwv '(1 2 3 4) '(a))
(1 A 2 3 4)
> (mix-wvwv '(1 2 3 4) '())
(1 2 3 4)
> (mix-wvwv '(1 2 3) '(a b c d))
(1 A 2 B 3 C D)
> (mix-wvwv '(1 2) '(a b c d))
(1 A 2 B C D)
> (mix-wvwv '(1) '(a b c d))
(1 A B C D)
> (mix-wvwv '() '(a b c d))
(A B C D)
```

Задача 10.10 mult-pairs.lisp

Перемножить поэлементно 2 числовых списка и вернуть результирующий список.

Решение 10.10.1

```
(defun mult-pairs (w v)
  (when (and w v)
    (cons (* (car w) (car v)) (mult-pairs (cdr w) (cdr v)))))
```

```
> (mult-pairs '(1 2 3) '(2 2 2))
(2 4 6)
```

Решение 10.10.2

```
(defun mult-pairs (w v)
  (mapcar #'* w v))
```

```
> (mult-pairs '(1 2 3) '(2 2 2))
(2 4 6)
```

Задача 10.11 our-set-difference.lisp

Разработать функцию, находящую теоретико-множественную разность двух списков.

Решение 10.11.1

```
(defun our-set-difference (w v)
  (cond ((null w) w)
        ((member (car w) v) (our-set-difference (cdr w) v))
        ((cons (car w) (our-set-difference (cdr w) v)))))

> (our-set-difference '(1 2 3 4 5) '(4 5 6 7))
(1 2 3)
```

Решение 10.11.2

```
(defun our-set-difference (w v)
  (when w (if (member (car w) v)
              (our-set-difference (cdr w) v)
              (cons (car w) (our-set-difference (cdr w) v)))))

> (our-set-difference '(1 2 3 4 5) '(4 5 6 7))
(1 2 3)
```

Задача 10.12 drop-suffixes.lisp

Написать программу, исключающую из слов их окончания по словарю. Словарь окончаний представлять списком строк.

Решение 10.12.1

```
(defun string-right (s n)
  (subseq s (- (length s) n)))

(defun drop-suffix (s v &aux (m (length s)) (n (length (car v))))
  (cond ((null v) s)
        ((string= (string-right s n) (car v)) (subseq s 0 (- m n)))
        ((drop-suffix s (cdr v)))))

(defun drop-suffixes (w v &aux (v> (sort v #'string>)))
  (mapcar #'(lambda (s) (drop-suffix s v>)) w))

> (drop-suffixes '("nokia" "nokia") '("kia" "okia"))
("n" "n")
```

Задача 10.13 -union.lisp

Определить функцию объединения двух множеств.

Решение 10.13.1

```
(defun -union (w v)
  (cond ((null w)  v)
        ((member (car w) v) (union (cdr w) v))
        ((cons (car w) (union (cdr w) v)))))

> (-union '(1 2 3) '(2 3 4))
(1 2 3 4)
```

Задача 10.14 apply-list.lisp

Определить функцию, которая применяет каждую функцию первого списка к соответствующему элементу второго списка и возвращает список, сформированный из результатов.

Решение 10.14.1

```
(defun apply-list (v w)
  (mapcar #'(lambda (a b) (funcall a b)) v w))

> (apply-list '(1- 1+ 1+) '(1 1 1))
(0 2 2)
```

Задача 10.15 inter-evenp.lisp

Найти четные числа в результирующем множестве, полученном из пересечения двух множеств.

Решение 10.15.1

```
(defun inter-evenp (w v)
  (remove-if #'oddp (intersection w v)))

> (inter-evenp '(1 2 3 4 5 6) '(4 5 6 7 8))
(4 6)
```

Задача 10.16 sort-exclusive.lisp

Реализовать функцию, которая осуществляет поиск и сортировку по возрастанию всех положительных чисел в результирующем множестве, полученном из симметричной разности двух множеств. В качестве множеств выступают списки из чисел.

Решение 10.16.1

```
(defun sort-exclusive (w v)
  (sort (set-exclusive-or w v) #'<))

> (sort-exclusive '(5 4 3) '(4 3 2))
(2 5)
```

Задача 10.17 unite-unique.lisp

Определить функцию создания списка, который является объединением двух списков, за исключением элементов, встречающихся в обоих списках.

Решение 10.17.1

```
(defun unite-unique (w v)
  (set-exclusive-or w v))

> (unite-unique '(1 2 3 4 5) '( 3 4 5 6 7))
(1 2 6 7)
```

Задача 10.18 cartesian-sort.lisp

Реализовать функцию, которая сортирует по возрастанию все числа в множестве, которое мы получаем в результате Декартового произведения двух множеств. Множества в виде списков чисел.

Решение 10.18.1

```
(defun cartesian-sort (w v)
  (mapcar #'(lambda (u) (sort u #'<))
          (mapcan (lambda (e) (mapcar (lambda (a) `(,a ,e)) w)) v)))

> (cartesian-sort '(2 2 2) '(1 1 1))
((1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2))
```

Решение 10.18.2

```
(defun cartesian-sort (w v)
  (mapcar #'(lambda (u) (sort u #'<))
          (when w (nconc (mi (car w) v) (cartesian-sort (cdr w) v)))))

(defun mi (a v)
  (when v (cons (list a (car v)) (mi a (cdr v)))))

> (cartesian-sort '(2 2 2) '(1 1 1))
((1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2))
```

Решение 10.18.3

```
(defun sort-cartesian (w v)
  (mapcar #'(lambda (u) (sort u #'<))
          (cond ((null w) nil)
```

```
                (t (append (mi (car w) v)
                           (sort-cartesian (cdr w) v))))))

(defun mi (a v)
  (cond ((null v) nil)
        (t (cons (list a (car v)) (mi a (cdr v))))))

> (sort-cartesian '(2 2 2) '(1 1 1))
((1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2) (1 2))
```

Задача 10.19 our-set-exclusive-or.lisp

Определить функцию высокого порядка для создания списка, который является объединением двух списков, за исключением элементов, встречающихся в обоих списках, т. е. симметричную разность.

Решение 10.19.1

```
(defun our-set-exclusive-or (w v)
  (nconc (unique w v) (unique v w)))

(defun unique (w v)
  (cond ((null w) nil)
        ((member (car w) v) (unique (cdr w) v))
        ((cons (car w) (unique (cdr w) v)))))

> (our-set-exclusive-or '(a b) '(b c))
(A C)
```

Задача 10.20 _intersection.lisp

Определить функцию высшего порядка для создания списка из элементов, которые встречаются в обоих списках.

Решение 10.20.1

```
(defun _intersection (w v)
  (cond ((null w) nil)
        ((member (car w) v) (cons (car w) (_intersection (cdr w) v)))
        ((_intersection (cdr w) v))))

> (_intersection '(a s d f g h ) '(q a w s e d r))
(A S D)
```

Решение 10.20.2

```
(defun _intersection (w v &aux (a (car w)))
  (when w (if (member a v)
              (cons a (_intersection (cdr w) v))
              (_intersection (cdr w) v))))
```

```
> (_intersection '(a s d f g h ) '(q a w s e d r))
(A S D)
```

Решение 10.20.3

```
(defun _intersection (w v)
  (loop for a in w when (member a v) collect a))

> (_intersection '(a s d f g h ) '(q a w s e d r))
(A S D)
```

Решение 10.20.4

```
(defun _intersection (w v)
  (remove-if-not #'(lambda (a) (member a v)) w))

> (_intersection '(a s d f g h ) '(q a w s e d r))
(A S D)
```

Задача 10.21 shuffle.lisp

Даны два списка, построить третий по условию: даны списки L1 и L2 элементов типа char, построить список L3 по следующему правилу:
 - если L1 является префиксом L2, то получить L3, исключив этот префикс и добавив его в качестве суффикса;
 - в противном случае получить L3, присоединив к L1 список L2.

Решение 10.21

```
(defun shuffle (v w &aux (n (mismatch v w)))
  (if (zerop n) (nconc v w) (nconc (subseq w n) v)))

> (shuffle '(#\a #\b #\c) '(#\a #\b #\c #\d))
(#\d #\a #\b #\c)

> (shuffle '(#\z #\z #\z) '(#\a #\b #\c #\d))
(#\z #\z #\z #\a #\b #\c #\d)
```

Задача 10.22 sum-elms.lisp

Даны два списка одинаковой длины, элементы которых - числа. Найти список с элементами - суммами соответствующих элементов исходных списков.

Решение 10.22.1

```
(defun sum-elms (w v)
  (cond ((null w) nil)
        ((cons (+ (car w) (car v)) (sum-elms (cdr w) (cdr v))))))

> (sum-elms '(1 2) '(1 2))
(2 4)
```

```
Решение 10.22.2

(defun sum-elms (w v)
  (mapcar #'+ w v))

> (sum-elms '(1 2) '(1 2))
(2 4)
```

Задача 10.23 drop-set.lisp

Даны списки w и v. Реализовать рекурсивную функцию, которая удаляет из w все элементы-списки, соответствующие тому же множеству, что и v. Пример: w: '(1 (2 3) 5 (2 3 3 2) 4 (2 2 3) 6) , v: '(3 2 3 2) результат: '(1 5 4 6).

```
Решение 10.23.1

(defun drop-set (w v)
  (when w (if (or (atom (car w)) (set-difference (car w) v))
              (cons (car w) (drop-set (cdr w) v))
              (drop-set (cdr w) v))))

> (drop-set '((2 3 2 3) 8 (2 3) (2 7) (2) (2 2 3) 5) '(3 2 3 2))
(8 (2 7) 5)

Решение 10.23.2

(defun drop-set (w v)
  (when w ((lambda (a d)
             (if (or (atom a) (set-difference a v))
                 (cons a (drop-set d v))
                 (drop-set d v)))
           (car w) (cdr w))))

> (drop-set '((2 3 2 3) 8 (2 3) (2 7) (2) (2 2 3) 5) '(3 2 3 2))
(8 (2 7) 5)

Решение 10.23.3

(defun drop-sub (w v)
  (remove-if #'(lambda (s)
                 (when (listp s)
                   (subsetp s v)))
             w))

> (drop-set '((2 3 2 3) 8 (2 3) (2 7) (2) (2 2 3) 5) '(3 2 3 2))
(8 (2 7) 5)

Решение 10.23.4

(defun drop-set (w v)
```

```
  (delete-if #'(lambda (s)
                 (and (listp s) (subsetp s v)))
             w))

> (drop-set '((2 3 2 3) 8 (2 3) (2 7) (2) (2 2 3) 5) '(3 2 3 2))
(8 (2 7) 5)
```

Решение 10.23.5*

* Переделать решение задачи, чтобы использовалась только форма let.

```
(defun drop-set (w v)
  (when w (let ((a (car w)) (d (cdr w)))
            (if (or (atom a) (nset-difference a v))
                (cons a (drop-set d v))
                (drop-set d v)))))

> (drop-set '((2 3 2 3) 8 (2 3) (2 7) (2) (2 2 3) 5) '(3 2 3 2))
(8 (2 7) 5)
```

Задача 10.24 bowl.lisp

> Some day we will build a thinking machine.
> It will be a truly intelligent machine. One
> that can see and hear and speak. A machine
> that will be proud of us.
>
> From a Thinking Machines brochure

Имеются два кувшина объемом 9 литров и 4 литра. Можно заполнять кувшины из источника, выливать и переливать воду из одного кувшина в другой. Найти последовательность действий, при которой в большем кувшине останется 6 литров воды.

Решение 10.24.1 (автор – Alexey Dejneka, www.progz.ru)

```
(defun mv (w s c &aux (n (copy-seq s)))
  (destructuring-bind (p f) c
       (case p
        ((m) (let* ((a (- 1 f))
                    (d (min (- (svref w a) (svref n a))
                            (svref n f))))
               (decf (svref n f) d) (incf (svref n a) d)))
        ((f) (setf (svref n f) (svref w f)))
        ((e) (setf (svref n f) 0)))
      n))

(defun gn (w s l)
  (loop for c in '((m 0) (m 1) (f 0) (f 1) (e 0) (e 1))
        for n = (mv w s c) unless (eq c l) collect (cons n c)))

(defun fg (w v s r l)
  (cond ((equalp s v) (list s))
```

```
        ((zerop r) nil)
        ((loop for (n . m) in (gn w s l) do
               (let ((z (fg w v n (1- r) m)))
                 (when z (return (cons s z))))))))

(defun bowl (w v &optional (s #(0 0)))
  (loop for r from 0 thereis (fg w v s r nil)))

> (bowl #(9 4) #(6 0))
(#(0 0) #(9 0) #(5 4) #(5 0) #(1 4) #(1 0) #(0 1) #(9 1) #(6 4) #(6
0))
> (bowl #(8 5) #(3 0))
(#(0 0) #(8 0) #(3 5) #(3 0))
> (bowl #(5 3) #(1 0))
(#(0 0) #(0 3) #(3 0) #(3 3) #(5 1) #(0 1) #(1 0))
> (bowl #(8 3) #(7 0))
(#(0 0) #(8 0) #(5 3) #(5 0) #(2 3) #(2 0) #(0 2) #(8 2) #(7 3) #(7
0))
> (bowl #(5 2) #(1 0))
(#(0 0) #(5 0) #(3 2) #(3 0) #(1 2) #(1 0))
> (bowl #(8 5) #(6 0))
(#(0 0) #(8 0) #(3 5) #(3 0) #(0 3) #(8 3) #(6 5) #(6 0))
```

Задача 10.25 remove-same-els.lisp

Даны списки lst1 и lst2. Реализовать функцию, которая удаляет из lst1 все элементы-списки, которые соответствуют тому же множеству, что и lst2. Пример: для списков: lst1 = (1 (2 2 3) 4 (3 2 3) 5), lst2 = (2 2 3) результатом будет (1 4 (3 2 3) 5).

Решение 10.25.1

```
(defun remove-same-els (w v)
  (cond ((null w) nil)
        ((equal (car w) v) (remove-same-els (cdr w) v))
        ((cons (car w) (remove-same-els (cdr w) v)))))

> (remove-same-els '(1 (2 2 3) 4 (3 2 3) 5) '(2 2 3))
(1 4 (3 2 3) 5)
```

Задача 10.26 add-pairs.lisp

Сложить попарно порядковые элементы двух списков, например (1 2 3) и (1 2 3), чтобы получился список (2 4 6).

Решение 10.26.1

```
(defun add-pairs (w v)
  (cond ((or (null w) (null v)) nil)
        ((cons (+ (car w) (car v)) (add-pairs (cdr w) (cdr v))))))
```

```
> (add-pairs '(1 2 3) '(1 2 3))
(2 4 6)
```

Решение 10.26.2

```
(defun add-pairs (w v)
  (mapcar #'+ w v))

> (add-pairs '(1 2 3) '(1 2 3))
(2 4 6)
```

Задача 10.27 intersect.lisp

Определите функцию, зависящую от двух аргументов w и v, являющихся списками, которая вычисляет список всех элементов, содержащихся одновременно и в списке w, и в списке v.

Решение 10.27.1

```
(defun intersect (w v &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((member a v) (cons a (intersect d v)))
        ((intersect d v))))

> (intersect '(a b c) '(b c d))
(B C)
```

Задача 10.28 product-elms.lisp

Определите функцию, зависящую от двух аргументов w и v, являющихся списками, которая вычисляет список всех элементов, содержащихся одновременно и в списке w, и в списке v.

Решение 10.28.1

```
(defun product-elms (w v)
  (loop for a in w
        for b in v collect (* a b)))

> (product-elms '(1 2 3) '(2 2 2))
(2 4 6)
```

Решение 10.28.2

```
(defun product-elms (w v) (mapcar #'* w v))

> (product-elms '(1 2 3) '(2 2 2))
(2 4 6)
```

Задача 10.29 insert-ordered.lisp

Определите функцию, которая вставляет элементы одного упорядоченного списка в другой упорядоченный список, не нарушая упорядоченности.

Решение 10.29.1

```
(defun insert-ordered (w v)
  (cond ((null w) v)
        ((null v) w)
        ((< (car w) (car v))
         (cons (car w) (insert-ordered (cdr w) v)))
        ((cons (car v) (insert-ordered w (cdr v))))))

> (insert-ordered '(1 2 4 5 7) '(0 1 2 3 4 5 6 7))
(0 1 1 2 2 3 4 4 5 5 6 7 7)
```

Решение 10.29.2

```
> (merge 'list '(1 2 4 5 7) '(0 1 2 3 4 5 6 7) #'<)
(0 1 1 2 2 3 4 4 5 5 6 7 7)
```

Задача 10.30 exclusive-atoms.lisp

Объединить два одноуровневых списка, из которых нужно выбрать только атомы, которые не повторяются.

Решение 10.30.1

```
(defun exclusive-atoms (w v)
  (cond ((null w) v)
    ((member (car w) v) (exclusive-atoms (cdr w) v))
    ((cons (car w) (exclusive-atoms (cdr w) v)))))

> (exclusive-atoms '(a b c) '(c b d))
(A C B D)
> (union '(a b c) '(c b d))
(A C B D)
```

Задача 10.31 exclusive-atomsxx.lisp

Объединить два многоуровневых списка, из которых нужно выбрать только атомы, которые не повторяются.

Решение 10.31.1

```
(defun atoms (w)
  (cond ((null w) nil)
        ((atom (car w)) (cons (car w) (atoms (cdr w))))
        ((atoms (cdr w)))))

(defun exclusive-atomsxx (w v)
  (cond ((null w) (atoms v))
```

```
        ((listp (car w)) (exclusive-atomsxx (cdr w) v))
        ((member (car w) v) (exclusive-atomsxx (cdr w) v))
        ((cons (car w) (exclusive-atomsxx (cdr w) v)))))

> (exclusive-atomsxx '(1 2 (3 4) 5 6) '(7 (8 9 10) 11 12))
(1 2 5 6 7 11 12)
```

Решение 10.31.2

```
(defun atoms (w)
  (cond ((null w) nil)
        ((atom (car w)) (cons (car w) (atoms (cdr w))))
        ((atoms (cdr w)))))

(defun exclusive-atoms (w v)
  (cond ((null w) v)
    ((member (car w) v) (exclusive-atoms (cdr w) v))
    ((cons (car w) (exclusive-atoms (cdr w) v)))))

(defun  exclusive-atomsxx (w v)
  (exclusive-atoms (atoms w) (atoms v)))

> (exclusive-atomsxx '(1 2 (3 4) 5 6) '(7 (8 9 10) 11 12))
(1 2 5 6 7 11 12)
```

Решение 10.31.3

```
(defun atoms (w)
  (delete-if-not #'atom w))

(defun exclusive-atomsxx-atoms (w v)
  (remove-duplicates (union (atoms w) (atoms v))))

> (exclusive-atomsxx '(1 2 3 3 (4 5 6) 7) '(7 (10 11 12) 13 14))
(1 2 3 13 14)
```

Задача 10.32 max-pair.lisp

Дано два списка: y1, y2. Определить функцию, создающую список y3, элементы которого определяются как y3 = max (y1, y2).

Решение 10.32.1

```
(defun max-pair (w v)
  (cond ((null w) nil)
        ((>= (car w) (car v))
         (cons (car w) (max-pair (cdr w) (cdr v))))
        ((cons (car v) (max-pair (cdr w) (cdr v))))))

> (max-pair '(1 2 3 4 5) '(2 1 33 11 0))
(2 2 33 11 5)
```

Решение 10.32.2

```
(defun max-pair (w v)
  (cond ((null w) nil)
        ((cons (if (>= (car w) (car v)) (car w) (car v))
               (max-pair (cdr w)  (cdr v))))))

> (max-pair '(1 2 3 4 5) '(2 1 33 11 0))
(2 2 33 11 5)
```

Решение 10.32.3

```
(defun max-pair (w v)
  (when w (cons (if (>= (car w) (car v)) (car w) (car v))
                (max-pair (cdr w)  (cdr v)))))

> (max-pair '(1 2 3 4 5) '(2 1 33 11 0))
(2 2 33 11 5)
```

Решение 10.32.4

```
(defun max-pair (w v)
  (when w (cons (max (car w) (car v)) (max-pair (cdr w) (cdr v)))))

> (max-pair '(1 2 3 4 5) '(2 1 33 11 0))
(2 2 33 11 5)
```

Решение 10.32.5

```
(defun max-pair (w v)
  (loop for a in w for b in v collect (max a b)))

> (max-pair '(1 2 3 4 5) '(2 1 33 11 0))
(2 2 33 11 5)
```

Решение 10.32.6

```
(defun max-pair (w v)
  (mapcar #'max w v))

> (max-pair '(1 2 3 4 5) '(2 1 33 11 0))
(2 2 33 11 5)
```

Задача 10.33 _set-difference.lisp

Определите функцию, зависящую от двух аргументов w и v, являющихся списками, которая вычисляет список всех элементов wa, не содержащихся в v.

Решение 10.33.1

```
(defun _set-difference (a b)
```

```
  (cond ((null a) nil)
        ((member (car a) b) (_set-difference (cdr a) b))
        ((cons (car a) (_set-difference (cdr a) b)))))

> (_set-difference '(1 2 3 4 5) '(3 4 5 6 7 8))
(1 2)
```

Решение 10.33.2

```
(defun _set-difference (w v)
  (loop for a in w unless (member a v) collect a))

> (_set-difference '(1 2 3 4 5) '(3 4 5 6 7 8))
(1 2)
```

Задача 10.34 a-merge.lisp

Объединить два упорядоченных числовых списка без нарушения порядка.

Решение 10.34.1

```
(defun a-merge (w v) (merge 'list w v #'<))

> (a-merge '(1 4 5 6 12) '(2 3 4 5))
(1 2 3 4 4 5 5 6 12)
```

Задача 10.35 zip.lisp

Разработать функцию, объединяющую два списка в результирующий список, в котором чередуются элементы исходных списков. Например:
Вход: (1 2 3 4 5), (a b c)
Выход: (1 a 2 b 3 c 4 5)

Решение 10.35.1

```
(defun zip (w v)
  (cond ((null w) v) ((null v) w)
        ((cons (car w) (zip v (cdr w))))))

> (zip '(1 2 3 4) '(a b c d))
(1 A 2 B 3 C 4 D)
> (zip '(1 2 3 4 5) '(a b c))
(1 A 2 B 3 C 4 5)
```

Задача 10.36 inter-mass.lisp

Напишите функцию (f x y), которая возвращает список z - "пересечение" списков x и y, т.е. список, содержащий их общие элементы, причем кратность (число повторений) каждого элемента в списке z равняется максимуму из его кратностей (числа повторений) в списках x и y.

Решение 10.36.1

```
(defun inter-mass (w v)
  (reduce #'nconc
          (loop for a in (remove-duplicates (intersection w v))
                collect (loop for e from 1 to
                              (max (count a w) (count a v))
                              collect a))))

> (inter-mass '(1 2 3 3 4 5) '(3 3 3 4 5 6 7))
(3 3 3 4 5)
```

Задача 10.37 mix-zip.lisp

L1, L2 – списки, элементами которых являются атомы. Построить список, четные элементы которого содержат соответствующие четные элементы списка L1, а нечетные - из L2. Пример: (f '(1 2 3 4) '(4 3 2 1)) Результат: (1 3 3 1)

Решение 10.37.1

```
(defun mix-zip (w v)
  (when w (cons (car w) (mix-zip (cdr v) (cdr w)))))

> (mix-zip '(1 2 3 4) '(5 6 7 8))
(1 6 3 8)
```

Задача 10.38 max-atoms.lisp

Даны списки списков чисел w и v. Определить, какой из списков меньше по длинам списков.

Решение 10.38.1

```
(defun len (w)
  (cond ((null w) 0)
        ((atom (car w)) (1+ (len (cdr w))))
        ((+ (len (car w)) (len (cdr w))))))

(defun max-atoms (w v)
  (if (>= (len w) (len v)) w v))

> (max-atoms '((2 (3) (4 ((5))))) '(1 (2 (3) 4) 5))
(1 (2 (3) 4) 5)
```

Задача 10.39 max-sum.lisp

Даны списки списков чисел w и v. Определить, какой из списков меньше по значениям элементов.

Решение 10.39.1

```
(defun sum (w)
  (cond ((null w) 0)
        ((atom (car w)) (+ (car w) (lenn (cdr w))))
        ((+ (lenn (car w)) (lenn (cdr w))))))

(defun max-sum (w v)
  (if (>= (sum w) (sum v)) w v))

> (max-sum '((2 (3) (4 ((5))))) '(1 (2 (3) 4) 5))
(1 (2 (3) 4) 5)
```

Задача 10.40 extract-nths.lisp

Определить функцию, возвращающую элементы первого списка, заданные номерами во втором списке.

Решение 10.40.1

```
(defun extract-nths (w v &optional (n 1))
  (cond ((null w) nil)
        ((member n v) (cons (car w) (extract-nths (cdr w) v (1+ n))))
        ((extract-nths (cdr w) v (1+ n)))))

> (extract-nths '(a b c d e) '(2 4))
(B D)
```

Задача 10.41 united.lisp

Определить функцию, результатом которой является список, представляющий объединение многоуровневых списков в смысле объединения множеств.

Решение 10.41.1

```
defun flat (w &optional acc)
  (cond ((null w) acc)
        ((atom w) (cons w acc))
        ((flat (car w) (flat (cdr w) acc)))))

(defun uni (w v &aux (a (car w)))
  (cond ((null w) (remove-duplicates v))
        ((or (member a (cdr w)) (member a v)) (uni (cdr w) v))
        ((cons a (uni (cdr w) v)))))

(defun united (w v)
  (uni (flat w) (flat v)))

> (united '((1) 3 4 (3)) '((3 4) 5 3))
(1 4 5 3)
```

Задача 10.42 sum-pairs.lisp

Определить функцию сложения элементов двух списков. Возвращает список, составленный из сумм элементов списков-параметров. Учесть, что переданные списки могут быть разной длины.

Решение 10.42.1

```
(defun sum-pairs (w v)
  (cond ((null w) v)
        ((null v) w)
        ((cons (+(car w) (car v)) (sum-pairs (cdr w) (cdr v))))))

> (sum-pairs '(1 1) '(1 1))
(2 2)
> (sum-pairs '(1 1) '(1 1 1))
(2 2 1)
```

Задача 10.43 construct.lisp

Определить функцию, формирующую список из второго списка по порядку номеров, представленных в первом списке.

Решение 10.43.1

```
(defun construct (w v)
  (cond ((null w) nil)
        ((cons (nth (car w) v) (construct (cdr w) v)))))

> (construct '(3 2 1 0) '(2 4 3 1))
(1 3 4 2)
```

Решение 10.43.2

```
(defun construct (w v)
  (when w (cons (nth (car w) v) (construc (cdr w) v))))

> (construct '(3 2 1 0) '(2 4 3 1))
(1 3 4 2)
```

Задача 10.44 max-inter-seq.lisp

Для двух списков найти общую последовательность элементов наибольшей длины (одну или несколько).

Решение 10.44.1

```
(defun max-inter-seq (w v &aux (z (arrange (inter-seq w v))))
  (most-wanted z (length (car z))))

(defun most-wanted (w n &aux (a (car w)))
```

```
  (when (and w (= (length  a) n)) (cons a (most-wanted (cdr w) n))))

(defun inter-seq (w v)
  (intersection (all-seq w) (all-seq v) :test #'equalp))

(defun arrange (w)
  (sort w #'> :key #'length))

(defun all-seq (w)
  (reduce #'nconc (mapcar #'cage (chariot w))))

(defun chariot (w)
 (when w (cons w (chariot (cdr w)))))

(defun cage (w)
  (when w (cons w (cage (butlast w)))))

> (max-inter-seq '(1 b c d e 2 3 4) '(0 c d e f g h))
((C D E))
> (max-inter-seq '(1 b c d e 2 3 4 e f g) '(0 c d e f g h))
((C D E) (E F G))
```

Решение 10.44.2

```
(defun max-inter-seq
    (w v &aux (z (sort (inter-seq w v) #'> :key #'length)))
  (most-wanted z (length (car z))))

(defun most-wanted (w n &aux (a (car w)))
  (when (and w (= (length  a) n)) (cons a (most-wanted (cdr w) n))))

(defun inter-seq (w v)
  (intersection (all-seq w) (all-seq v) :test #'equalp))

(defun all-seq (w)
  (reduce #'nconc (mapcar #'cage (loop for a on w collect a))))

(defun cage (w)
  (when w (cons w (cage (butlast w)))))

> (max-inter-seq '(1 b c d e 2 3 4) '(0 c d e f g h))
((C D E))
> (max-inter-seq '(1 b c d e 2 3 4 e f g) '(0 c d e f g h))
((C D E) (E F G))
```

Решение 10.44.3

```
(defun max-inter-seq (w v)
  (wanted (sort (inter-seq w v) #'> :key #'length)))

(defun wanted (w &optional (n (length (car w))) &aux (a (car w)))
  (when (and w (= (length a) n)) (cons a (wanted (cdr w) n))))
```

```
(defun inter-seq (w v)
  (intersection (all-seq w) (all-seq v) :test #'equalp))

(defun all-seq (w)
  (reduce #'nconc (mapcar #'cage (loop for a on w collect a))))

(defun cage (w)
  (when w (cons w (cage (butlast w)))))

> (max-inter-seq '(1 b c d e 2 3 4) '(0 c d e f g h))
((C D E))
> (max-inter-seq '(1 b c d e 2 3 4 e f g) '(0 c d e f g h))
((C D E) (E F G))
```

Задача 10.45 maxx-inter-seq.lisp

Для двух и более список найти общую последовательность элементов наибольшей длины (одну или несколько).

Решение 10.45.1

```
(defun maxx-inter-seq (&rest w)
  (wanted (sort (inter-seq w) #'> :key #'length)))

(defun wanted (w &optional (n (length (car w))) &aux (a (car w)))
  (when (and w (= (length a) n)) (cons a (wanted (cdr w) n))))

(defun inter-seq (w)
  (reduce #'(lambda (a b) (intersection a b :test #'equalp))
          (mapcar #'all-seq w)))

(defun all-seq (w)
  (reduce #'nconc (mapcar #'cage (loop for a on w collect a))))

(defun cage (w)
  (when w (cons w (cage (butlast w)))))

> (maxx-inter-seq '(b c d e f 2 e) '(0 c d e f g) '(b c d e))
((C D E))
```

Задача 10.46 deep-member.lisp

Определить функцию, проверяющую является ли список элементом другого списка.

Решение 10.46.1

```
(defun deep-member (v w)
  (cond ((null w) nil)
        ((equalp (car w) v) t)
        ((atom (car w)) (deep-member v (cdr w)))
```

```
        (t (or (deep-member v (car w)) (deep-member v (cdr w))))))

> (deep-member '(1 2 (7)) '(1 2 2 (1 2(7)) 1))
T
> (deep-member '(7) '(1 2 2 (1 2(7)) 1))
T
```

Решение 10.46.2

```
(defun deep-member (v w)
  (cond ((null w) nil)
        ((equalp (car w) v))
        ((atom (car w)) (deep-member v (cdr w)))
        ((or (deep-member v (car w)) (deep-member v (cdr w))))))

> (deep-member '(1 2 (7)) '(1 2 2 (1 2(7)) 1))
T
> (deep-member '(7) '(1 2 2 (1 2(7)) 1))
T
```

Задача 10.47 cartesian-product-sorted.lisp

Определить функцию, образующую упорядоченное декартово произведение двух заданных множеств'(b c a) и '(3 1 2).

Решение 10.47.1

```
(defun cartesian-product-sorted (w v)
  (cartesian-product (sort-lets w) (sort v #'<)))

(defun sort-lets (w)
  (loop for a across
        (sort (apply #'concatenate
                     'string
                     (mapcar #'string w))
              #'char<)
        collect (intern (string a))))

(defun cartesian-product (w v)
  (when w
    (append
     (maplist
      #'(lambda (e) (list (car w) (car e))) v)
     (cartesian-product (cdr w) v))))

> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.47.2

```
(defun cartesian-product-sorted (w v)
  (cartesian-product (sort-lets w) (sort v #'<)))
```

```
(defun sort-lets (w)
  (loop for a across
        (sort (apply #'concatenate
                     'string
                     (mapcar #'string w))
              #'char<)
        collect (intern (string a))))

(defun cartesian-product (w v)
  (apply
   #'nconc
   (mapcar
    #'(lambda (e) (mapcar #'(lambda (a) (list e a)) v)) w)))

> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.47.3

```
(defun cartesian-product-sorted (w v)
  (cartesian-product (sort-lets w) (sort v #'<)))

(defun sort-lets (w)
  (loop for a across
        (sort (apply #'concatenate
                     'string
                     (mapcar #'string w))
              #'char<)
        collect (intern (string a))))

(defun cartesian-product (w v)
  (mapcan (lambda (e) (mapcar (lambda (a) `(,a ,e)) w)) v))

> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.47.4

```
(defun cartesian-product-sorted (w v)
  (cartesian-product (sort-lets w) (sort v #'<)))

(defun sort-lets (w)
  (loop for a in
        (sort (mapcar #'string w) #'string<)
        collect (intern a)))

(defun cartesian-product (w v)
  (mapcan (lambda (e) (mapcar (lambda (a) `(,a ,e)) w)) v))

> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.47.5

```
(defun cartesian-product-sorted (w v)
  (mapcan #'(lambda (e)
              (mapcar #'(lambda (a) `(,a ,e))
                      (loop for a in
                            (sort (mapcar #'string w)
                                  #'string<)
                            collect (intern a))))
          (sort v #'<)))

> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.47.6

```
(defun cartesian-product-sorted (w v)
  (cartesian-product (sort-lets w) (sort v #'<)))

(defun sort-lets (w)
  (loop for a in
        (sort (mapcar #'string w) #'string<)
        collect (intern a)))

(defun cartesian-product (w v)
   (loop for e in w nconc (loop for a in v collect (list e a))))

> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.47.7

```
(defun cartesian-product-sorted (w v)
  (cartesian-product
   (loop for a in
         (sort (mapcar #'string w) #'string<)
         collect (intern a))
   (sort v #'<)))

(defun cartesian-product (w v)
   (loop for e in w nconc (loop for a in v collect (list e a))))

> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.47.8

```
(defun cartesian-product-sorted (w v)
  (loop for e in (loop for a in (sort (mapcar #'string w) #'string<)
                       collect (intern a))
        nconc (loop for a in (sort v #'<)
                    collect (list e a))))
```

```
> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Решение 10.47.9

```
(defun cartesian-product-sorted (w v)
  (loop for e in
        (loop for a in (sort (loop for c in w
                                   collect (string c))
                             #'string<)
              collect (intern a))
        nconc (loop for a in (sort v #'<)
                    collect (list e a))))
```

```
> (cartesian-product-sorted '(b c a) '(3 1 2))
((A 1) (A 2) (A 3) (B 1) (B 2) (B 3) (C 1) (C 2) (C 3))
```

Задача 10.48 bowl-sorted.lisp

Определить функцию, соединяющую два упорядоченных по неубыванию списка в упорядоченный по неубыванию список.

Решение 10.48.1

```
(defun bowl-sorted (w v &optional acc)
  (cond ((and (null w) (null v)) (reverse acc))
        ((null w) (nconc (reverse acc) v))
        ((null v) (nconc (reverse acc) w))
        ((<= (car w) (car v))
         (bowl-sorted (cdr w) v (cons (car w) acc)))
        ((bowl-sorted w (cdr v) (cons (car v) acc)))))
```

```
> (bowl-sorted '(1 1 2 5) '(0 3 4))
(0 1 1 2 3 4 5)
```

Решение 10.48.2

```
(defun bowl-sorted (w v)
  (cond ((null w) v)
        ((null v) w)
        ((< (car w) (car v))
         (cons (car w) (bowl-sorted (cdr w) v)))
        ((cons (car v) (bowl-sorted w (cdr v))))))
```

```
> (bowl-sorted '(1 1 2 5) '(0 3 4))
(0 1 1 2 3 4 5)
```

Решение 10.48.3

```
(defun bowl-sorted (w v)
  (cond ((null w) v)
        ((null v) w)
```

```
        ((< (car w) (car v)) (glue w v))
        (t (glue v w))))

(defun glue (w v)
  (cons (car w) (bowl-sorted (cdr w) v)))

> (bowl-sorted '(1 1 2 5) '(0 3 4))
(0 1 1 2 3 4 5)
```

Решение 10.48.4

```
(defun bowl-sorted (w v)
  (cond ((null w) v) ((null v) w)
        ((< (car w) (car v)) (glue w v)) ((glue v w))))

(defun glue (w v)
  (cons (car w) (bowl-sorted (cdr w) v)))

> (bowl-sorted '(1 1 2 5) '(0 3 4))
(0 1 1 2 3 4 5)
```

Задача 10.49 sum-/.lisp

Даны два списка: (1 2 3 4) и (4 5 6 7). Получить новый список (1/4 +2/5 + 3/6 +4/7).

Решение 10.49.1

```
(defun sum-/ (w v)
  (reduce #'+ (mapcar #'/ w v)))

> (sum-/ '(1 2 3 4) '(4 5 6 7))
241/140
> (reduce #'+ '(1/4 2/5 3/6 4/7))
241/140
```

Решение 10.49.2

```
(defun sum-/ (w v)
  (butlast (mapcan #'(lambda (a b) (list (/ a b) '+)) w v)))

> (sum-/ '(1 2 3 4) '(4 5 6 7))
(1/4 + 2/5 + 1/2 + 4/7)
```

Решение 10.49.3

```
(defun sum-/ (w v)
  (butlast (mapcan #'(lambda (a b) `(,(/ a b) +)) w v)))

> (sum-/ '(1 2 3 4) '(4 5 6 7))
(1/4 + 2/5 + 1/2 + 4/7)
```

```
Решение 10.49.4

(defun sum-/ (w v)
  (butlast (loop for a in w
                 for b in v
                 nconc (list (/ a b) '+))))

> (sum-/ '(1 2 3 4) '(4 5 6 7))
(1/4 + 2/5 + 1/2 + 4/7)
```

Задача 10.50 matrices-tensor-product.lisp

```
Определить функцию, возвращающую тензорное произведение двух матриц.

Решение 10.50.1

(defun matrices-tensor-product (w v)
  (tensor (reduce #'nconc w) (reduce #'nconc v)))

(defun tensor (w v)
  (when w (cons (product v (car w)) (tensor (cdr w) v))))

(defun product (w n)
  (when w (cons (* (car w) n) (product (cdr w) n))))

> (matrices-tensor-product '((1 2) (3 4)) '((11 22) (33 44)))
((11 22 33 44) (22 44 66 88) (33 66 99 132) (44 88 132 176))

Решение 10.50.2

(defun matrices-tensor-product (ww vv
                                &aux
                                (w (reduce #'nconc ww))
                                (v (reduce #'nconc vv)))
  (mapcar #'(lambda (a) (product v a)) w))

(defun product (w n)
  (when w (cons (* (car w) n) (product (cdr w) n))))

> (matrices-tensor-product '((1 2) (3 4)) '((11 22) (33 44)))
((11 22 33 44) (22 44 66 88) (33 66 99 132) (44 88 132 176))

Решение 10.50.3

(defun matrices-tensor-product (ww vv
                                &aux
                                (w (reduce #'nconc ww))
                                (v (reduce #'nconc vv)))
  (loop for a in w collect (product v a)))

(defun product (w n)
  (when w (cons (* (car w) n) (product (cdr w) n))))
```

```
> (matrices-tensor-product '((1 2) (3 4)) '((11 22) (33 44)))
((11 22 33 44) (22 44 66 88) (33 66 99 132) (44 88 132 176))
```

Решение 10.50.4

```
(defun matrices-tensor-product (ww vv
                                &aux
                                (w (reduce #'nconc ww))
                                (v (reduce #'nconc vv)))
  (mapcar
   #'(lambda (a)
       (mapcar
        #'(lambda (b) (* a b))
        v))
   w))
```

```
> (matrices-tensor-product '((1 2) (3 4)) '((11 22) (33 44)))
((11 22 33 44) (22 44 66 88) (33 66 99 132) (44 88 132 176))
```

Решение 10.50.5

```
(defun matrices-tensor-product (ww vv
                                &aux
                                (w (reduce #'nconc ww))
                                (v (reduce #'nconc vv)))
  (loop for a in w collect
        (loop for b in v collect (* a b))))
```

```
> (matrices-tensor-product '((1 2) (3 4)) '((11 22) (33 44)))
((11 22 33 44) (22 44 66 88) (33 66 99 132) (44 88 132 176))
```

Структура 11 (функция ATOM ATOM ATOM) > ATOM

Задача 11.1 sum-fibonacci.lisp

Each new term in the Fibonacci sequence is generated by adding the previous two terms. By starting with 1 and 2, the first 10 terms will be: 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ... Find the sum of all the even-valued terms in the sequence which do not exceed four million.

Решение 11.1.1

```
(defun sum-fibonacci (a b z &aux (c (+ a b)))
  (if (< b z)
      (if (evenp b)
          (+ b (sum-fibonacci b c z))
          (sum-fibonacci b c z))
      0))
```

```
> (sum-fibonacci 1 2 4000000)
4613732
```

Задача 11.2 common-elmsp.lisp

Написать функцию, которая проверяет - состоят ли два заданных списка из одних и тех же элементов (независимо от порядка их расположения).

Решение 11.2.1

```
(defun common-elmsp (w v)
  (null (set-difference w v)))

> (common-elmsp '(a b c) '(c b a))
T
> (common-elmsp '(a b c) '(c b b))
NIL
> (odd3+ 2 3 5)
10
```

Задача 11.3 in-range.lisp

Написать функцию, которая проверяет принадлежит ли число какому-либо диапазону чисел.

Решение 11.3.1

```
(defun in-range (a b c)
  (<= b a c))

> (in-range 1 1 3)
T
```

Задача 11.4 n-fibonacci.lisp

Последовательность чисел Фибоначчи 1,1,2,3,5,8,13,21,34,55,89,144, 233,377,610,987,1597,2584,4181,6765,10946… строится по следующему правилу: первые два числа – единицы; любое следующее число есть сумма двух предыдущих f(n) = f(n - 1) + f(n - 2). Напишите функцию (f n f1 f2) с накапливающимися параметрами f1 и f2, которая вычисляет n-ое число Фибоначчи.

Решение 11.4.1

```
(defun n-fibonacci (n) *без накапливающих параметров, простая рекурсия
  (if (< n 2)
      n
      (+ (n-fibonacci (- n 1)) (n-fibonacci (- n 2)))))

> (n-fibonacci 7)
 13
```

```
> (time (n-fibonacci 35))
Real time: 11.477045 sec.
Run time: 11.434874 sec.
Space: 0 Bytes
9227465
```

Решение 11.4.2

```
(defun n-fibonacci (n) *без накапливающих, хвостовая рекурсия
  (labels ((fibonacci (n a b)
             (if (= n 0)
                 a
                 (fibonacci (- n 1) b (+ a b)))))
    (fibonacci n 0 1)))
```

```
> (n-fibonacci 7)
 13
> (time (n-fibonacci 35))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 0 Bytes
9227465
```

Решение 11.4.3 *с накапливающими, хвостовая рекурсия

```
(defun n-fibonacci (a b n)
  (cond ((= n 2) b)
        ((n-fibonacci b (+ a b) (1- n)))))
```

```
> (n-fibonacci 1 1 7)
 13
> (time (n-fibonacci 1 1 35))
Real time: 0.0 sec.
Run time: 0.0 sec.
Space: 0 Bytes
9227465
```

Задача 11.5 replace-lets.lisp

Дан текст. В каждом слове текста заменить заданную литеру заданным сочетанием литер. Пример: заменяемая литера b, заменяющее сочетание литер: “ku”, слово: "abrakadabra", результат: "akurakadakura".

Решение 11.5.1

```
(defun replace-lets (s p r &aux (e (search p s)))
  (cond ((null e) s)
        ((concatenate
          'string
          (subseq s 0 e)
          r
          (replace-lets (subseq s (1+ e)) p r)))))
```

```
> (replace-lets "abrakadabra" "b" "ku")
"akurakadakura"
```

Решение 11.5.2

```
(defun replace-lets (s p r &aux (e (search p s)))
  (if e
      (concatenate
       'string
       (subseq s 0 e)
       r
       (replace-lets (subseq s (1+ e)) p r))
      s))

> (replace-lets "abrakadabra" "b" "ku")
"akurakadakura"
```

Задача 11.6 a-delete-b.lisp

Дан текст. В каждом слове удалить литеру, стоящую между двумя заданными.

Решение 11.6.1

```
(defun sw (s)
  (map 'list #'string s))

(defun az (w a b)
  (cond ((null w) nil)
        ((and (string= (car w) a) (string= (caddr w) b))
         (cons (car w) (az (cddr w) a b)))
        ((cons (car w) (az (cdr w) a b)))))

(defun cn (w)
  (cond ((null w) nil)
        ((concatenate 'string (car w) (cn (cdr w))))))

(defun a-delete-b (s a b)
  (cn (az (sw s) a b)))

> (a-delete-b "abc abc" "a" "c")
"ac ac"
```

Решение 11.6.2

```
defun sw (s)
  (map 'list #'string s))

(defun az (w a b)
  (cond ((null w) nil)
        ((and (equal (car w) a) (equal (caddr w) b))
```

```
         (cons (car w) (az (cddr w) a b)))
        ((cons (car w) (az (cdr w) a b)))))

(defun cn (w)
  (format nil "~{~A~^~}" w))

(defun a-delete-b (s a b)
  (cn (az (sw s) a b)))

> (a-delete-b "abc abc" "a" "c")
"ac ac"
```

Решение 11.6.3

```
(defun sw (s)
  (map 'list #'string s))

(defun az (w a b)
  (cond ((null w) nil)
        ((and (string-equal (car w) a) (string-equal (caddr w) b))
         (cons (car w) (az (cddr w) a b)))
        ((cons (car w) (az (cdr w) a b)))))

(defun cn (w)
  (format nil "~{~A~^~}" w))

(defun a-delete-b (s a b)
  (cn (az (sw s) a b)))

> (a-delete-b "abc abC" "a" "c")
"ac aC"
```

Задача 11.7 sum-n-fibonacci.lisp

Вычислить сумму n чисел Фибоначчи.

Решение 11.7.1

```
(defun sum-n-fibonacci (a b z &aux (c (+ a b)))
  (if (= z 1) 1 (+ b (sum-n-fibonacci b c (1- z)))))

> (sum-n-fibonacci 1 1 100)
927372692193078999175
```

Задача 11.8 sum-fibonacci<=n.lisp

Последовательность чисел Фибоначчи 1,1,2,3,5,8,13… строится по следующему правилу: первые два числа – единицы; любое следующее число есть сумма двух предыдущих f(n) = f(n - 1) + f(n - 2). Напишите функцию (f f1 f2 n), которая вычисляет сумму чисел Фибоначчи которые не превышают n.

Решение 11.8.1

```
(defun sum-fibonacci<=n (&optional (a 0) (b 1) &aux (c (+ a b)))
  (if (< b 100) (+ b (sum-fibonacci<=n b c)) 0))

> (sum-fibonacci<=n)
232
```

Решение 11.8.2

```
(defun sum-fibonacci<=n (&optional (a 0) (b 1))
  (if (< b 100) (+ b (sum-fibonacci<=n b (+ a b))) 0))

> (sum-fibonacci<=n)
232
```

Решение 11.8.3

```
(defun sum-fibonacci<=n (a b z &aux (c (+ a b)))
  (if (< b z)
      (+ b (sum-fibonacci<=n b c z))
      0))

> (sum-fibonacci<=n 0 1 100)
232
```

Решение 11.8.4

```
(defun sum-fibonacci<=n (a b z)
  (if (< b z)
      (+ b (sum-fibonacci<=n b (+ a b) z))
      0))
> (sum-fibonacci<=n 0 1 100)
232
```

Задача 11.9 balloon-eq.lisp

Воздушный шар летит в одном направлении со скоростью 20 км/час в течение 1 часа и 45 минут. Затем направление движения меняется на заданный угол (60°), и воздушный шар летит еще 1 час и 45 минут с той же скоростью. Найти расстояние от точки старта до точки приземления.

Решение 11.9.1

```
(defun balloon-eq (v t a)
  (* 2 (sin (* pi (/ a 360))) v t))

> (balloon-eq 20 1.75 60)
35.000000000000000003L0
```

Задача 11.10 arithmetic-n.lisp

Найти n-й член арифметической прогрессии (нумерация начинается с нуля).

Решение 11.10.1

```
(defun arithmetic-n (zero step n)
  (+ zero (* step n)))

> (arithmetic-n 0 2 100)
200
```

Задача 11.11 check-expt.lisp

Определить функцию, проверяющую является ли третий элемент списка результатом возведения в степень первого элемента с показателем,равным второму элементу. Если является,то вернуть t,иначе - nil.

Решение 11.11.1

```
(defun check (a b c)
  (eql (expt a b) c))

> (check 2 3 8)
T
> (check 2 3 3)
NIL
> (eq 1.1234 1.1234)
NIL
> (eql 1.1234 1.1234)
T
> (check 2 3.123 8.711975)
T
```

Задача 11.12 check-random.lisp

Определить функцию, которая будет проверять, попало ли случайно выбранное из отрезка (5 155) целое число в интервал (25 100).

Решение 11.12.1

```
(defun check-random (n m a b &aux (k (+ n (random (1+ (- m n))))))
  (when (< a k b) k))

> (check-random 5 155 25 100)
37
> (check-random 5 155 25 100)
34
> (check-random 5 155 25 100)
NIL
```

Задача 11.13 triangle.lisp

Определить функцию (f a b c), которая равна истине тогда и только тогда, когда из отрезков с длинами a,b и c можно построить треугольник.

Решение 11.13.1

```
(defun triangle (a b c)
  (< (abs (- a b)) c (+ a b)))

> (triangle 2 2 3)
T
> (triangle 1 2 3)
NIL
```

Структура 12 (функция ATOM ATOM ATOM) > (СПИСОК)

Задача 12.1 our-list.lisp

Определите функцию (list x1 x2 x3) с помощью базовых функций. Используйте имя our-list, чтобы не переопределять одноименную встроенную функцию Лиспа.

Решение 12.1.1

```
(defun our-list (a b c)
  (cons a (cons b (cons c nil))))

> (our-list 'z 'zz 'zzz)
(Z ZZ ZZZ)
```

Решение 12.1.2

```
(defun our-list (&rest w) w)

> (our-list 'z 'zz 'zzz)
(Z ZZ ZZZ)
```

Задача 12.2 odd3+.lisp

Даны три числа. Построить список из кубов этих чисел, если все три числа - нечетные, вернуть сумму чисел - иначе.

Решение 12.2.1

```
(defun odd3+ (a b c)
  (if (and (oddp a) (oddp b) (oddp c))
      (list (expt a 3) (expt b 3) (expt c 3))
      (+ a b c)))
```

```
> (odd3+ 1 3 5)
(1 27 125)
```

Задача 12.3 wagon.lisp

Надо построить 180 км железной дороги. Вес одного метра рельса 36 кг. Сколько потребуется вагонов грузоподьемности 60 тонн, что бы перевезти все рельсы к месту стройки одноколейной дороги.

Решение 12.3.1

```
(defun wagon (r s w)
  (/ (* r 2 s 1000) w 1000))

> (wagon 36 180 60)
216
```

Решение 12.3.2

```
(defun wagon (r s w)
  (/ (* r 2 s) w))

> (wagon 36 180 60)
216
```

Задача 12.4 square2nd-or-remove3rd.lisp

Написать функцию с тремя аргументами, которая проверяет, является ли второй элемент списка вещественным или рациональным числом. Если является, то поменять его на квадрат этого числа. Если не является, то вернуть исходный список без последнего элемента.

Решение 12.4.1

```
(defun square2nd-or-remove3rd (a b c)
  (if (realp b) (list a (expt b 2) c) (list a b)))

> (square2nd-or-remove3rd 1 2.2 3)
(1 4.84 3)
```

Решение 12.4.2

```
(defun square2nd-or-remove3rd (a b c)
  (if (realp b) `(,a ,(expt b 2) ,c) `(,a ,b)))

> (square2nd-or-remove3rd 1 2.2 3)
(1 4.84 3)
```

Задача 12.5 trapezoid.lisp

Определить функцию, которая печатает числа в виде трапеции.

Решение 12.5.1

```
(defun trap (w m &aux (v (butlast w)))
  (cond ((zerop m) nil)
        (t (print w) (trap v (1- m)))))

(defun trapezoid (k n m)
  (trap (loop for a from k to n collect a) m))


> (trapezoid 1 6 3)
(1 2 3 4 5 6)
(1 2 3 4 5)
(1 2 3 4)
NIL
```

Задача 12.6 div-rem.lisp

Напишите программу поиска пятизначного числа, которое при делении на 133 дает в остатке 125, а при делении на 134 дает в остатке 125.

Решение 12.6.1

```
(defun div-rem (n m d r)
  (loop for a from n to m when (eq (rem a d) r) collect a))

> (intersection (div-rem 10000 99999 133 125) (div-rem 10000 999999
134 125))
(17947 35769 53591 71413 89235)
```

Решение 12.6.2

```
(defun div-rem (b c d e)
  (loop for a from 10000 to 99999 when (and (eq (rem a b) c)
                                            (eq (rem a d) e))
                                       collect a))

> (div-rem 133 125 134 125)
(17947 35769 53591 71413 89235)
```

Структура 13 (функция (СПИСОК) ATOM ATOM) > ATOM

Задача 13.1 sum-sequence.lisp

Определить функцию, которая суммирует элементы одноуровневого числового списка w, расположенные на позициях от n до m включительно (n < m).

Решение 13.1.1

```
(defun sum-sequence (w n m)
  (cond ((= m 0) 0)
        ((= n 1) (+ (car w) (sum-sequence (cdr w) n (1- m))))
        ((sum-sequence (cdr w) (1- n) (1- m)))))

> (sum-sequence '(1 2 3 4 5 6 7) 3 5)
12
```

Решение 13.1.2

```
(defun sum-sequence (w n m)
  (reduce #'+ (subseq w (1- n) m)))

> (sum-sequence '(1 2 3 4 5 6 7) 3 5)
12
```

Задача 13.2 zeros-only.lisp

Определить рекурсивную функцию, (f w a b), которая проверяет, действительно ли между элементами a и b в одноуровневом списке w нет иных элементов, кроме нулей (элемент a встречается раньше элемента b).

Решение 13.2.1

```
defun zeros (w b)
  (cond ((null w) t)
        ((equalp (car w) b) t)
        ((equalp (car w) 0) (zeros (cdr w) b))
        (t nil)))

(defun zeros-only (w a b)
  (cond ((null w) t)
        ((equalp (car w) a) (zeros (cdr w) b))
        ((zeros-only (cdr w) a b))))

> (zeros-only '(1 2 3 0 0 7) 3 7)
T
> (zeros-only '(1 2 3 100 7) 3 7)
NIL
```

Структура 14 (функция (СПИСОК) АТОМ АТОМ) > (СПИСОК)

Задача 14.1 sbst.lisp

Определить рекурсивную функцию, которая заменяет все вхождения данного элемента в списке на новый элемент.

Решение 14.1.1

```
(defun sbst (n m w)
   (cond ((null w) nil)
         ((eql (car w) m) (cons n (sbst n m (cdr w))))
         ((cons (car w) (sbst n m (cdr w))))))

> (sbst 2 3 '(1 2 3))
(1 2 2)
```

Комментарии (построчно):
1. Определяем функцию **sbst** с тремя параметрами: новый атом **n**, старый атом **m** и список **w**.
2. Если (**cond**) пустой (**null**) список w функция **sbst** возвращает **nil**.
3. Если первый элемент (**car w**) равен **m**, то вставляет (**cons**) элемент **n** на первое место в результат вызова **sbst** с параметрами: **n**, **m** и списком без первого элемента (**cdr w**).
4. При невыполнении 1 и 2 условий вставляет (**cons**) первый элемент (**car w**) в результат вызова **sbst** с параметрами: **n**, **m** и списком без первого элемента (**cdr w**).

Решение 14.1.2

```
(defun sbst (w n m &aux (a (car w)))
  (when w (cons (if (equal a n) m a) (sbst (cdr w) n m))))

> (sbst '(1 a 3 a 5 a 7) 'a 'b)
(1 B 3 B 5 B 7)
```

Задача 14.2 drop-range.lisp

Все списки содержат подсписки. Программировать с заходом в подсписки - параллельная рекурсия. Элементами списка могут быть атомы-символы, атомы-числа, подсписки из атомов-символов, атомов-чисел. Нужно: Удалить из списка все атомы-числа из интервала [a,b].

Решение 14.2.1

```
(defun drop-range (n m w)
  (when w ((lambda (a d)
             (cond ((listp a)
                    (cons (drop-range n m a) (drop-range n m d)))
                   ((and (numberp a) (<= n a m)) (drop-range n m d))
                   ((cons a (drop-range n m d)))))
           (car w) (cdr w))))

> (drop-range 3 5 '((1 2 a 4 8) ((3) b (u) 2)))
((1 2 A 8) (NIL B (U) 2))
```

Задача 14.3 list-goods.lisp

Прайс компьютерного магазина описывается списком (наименование фирма параметры цена): ((Терминал Acer 15 4500) (Терминал Sumsung 17

7000)(Матплата Asus 238 2500) (Процессор Intel 2.5 3000)). Определить функцию возвращающую список товаров с заданным наименованием и ценой.

Решение 14.3.1

```
(defun list-googs (m p w)
  (delete-if-not
   #'(lambda (a) (and (eq (car a) m) (eql (cadddr a) p))) w))

> (mp 'terminal 7000 '((terminal acer 15 4500) (terminal samsung 17
7000) (motherboard intel 238 2500) (processor intel 2.5 3000)))
((TERMINAL SAMSUNG 17 7000))
```

Задача 14.4 select-range.lisp

Написать функцию, которая принимает произвольный одноуровневый список и возвращает только числа заданного диапазона.

Решение 14.4.1

```
(defun select-range (w x z &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((and (numberp a) (<= x a z)) (cons a (select-range d x z)))
        ((select-range d x z))))

> (select-range '(a 1 2 3 4) 1 3)
(1 2 3)
```

Решение 14.4.2

```
(defun select-range (w x z)
  (delete-if-not #'(lambda (a) (and (numberp a) (<= x a z))) w))

> (select-range '(a 1 2 3 4) 1 3)
(1 2 3)
```

Задача 14.5 frame-elms.lisp

Определить функцию, которая добавляет заданный элемент слева и справа от другого заданного элемента.

Решение 14.5.1

```
(defun frame-elms (z u w &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((atom a) (if (eq u a)
                      (cons z (cons a (cons z (frame-elms z u d))))
                      (cons a (frame-elms z u d))))
        ((cons (frame-elms z u a) (frame-elms z u d)))))

> (frame-elms 5 8 '((1 8 2) (3 (8 4))))
```

```
((1 5 8 5 2) (3 (5 8 5 4)))
```

Задача 14.6 reduce-combined.lisp

В числовом списке сложить или умножить каждые N соседних элемента, пример работы функции: (f #'+ 2 '(2 3 4 5 6 7)) > (5 9 13)

Решение 14.6.1

```
(defun combine-elms (n w &optional (m n) ac acc)
  (cond ((null w) (reverse (cons (reverse ac) acc)))
        ((zerop m) (combine-elms n w n nil (cons (reverse ac) acc)))
        ((combine-elms n (cdr w) (1- m) (cons (car w) ac) acc))))

(defun reduce-combined (function n w)
  (mapcar #'(lambda (v) (reduce function v)) (combine-elms n w)))

> (reduce-combined #'+ 2 '(2 3 4 5 6 7))
(5 9 13)
> (reduce-combined #'* 3 '(2 3 4 5 6 7))
(24 210)
```

Задача 14.7 insert-lets.lisp

Дан текст. В каждом слове вставить после заданного 3-буквенного сочетания заданное 2-буквенное.

Решение 14.7.1

```
(defun insert-lets (w a b)
  (cond ((null w) nil)
        ((eql (car w) a)
         (cons (intern (concatenate 'string
                                    (string a)
                                    (write-to-string b)))
               (insert-lets (cdr w) a b)))
        ((cons (car w) (insert-lets (cdr w) a b)))))

> (insert-lets '(ab abc ac abd) 'abc 12)
(AB ABC12 AC ABD)
```

Решение 14.7.2

```
(defun insert-lets (w a b)
  (mapcar
   #'(lambda (x)
       (if (eql x a)
           (intern (concatenate 'string
                                (string a)
                                (write-to-string b)))
           x))
```

```
   w))

> (insert-lets '(ab abc ac abd) 'abc 12)
(AB ABC12 AC ABD)
```

Решение 14.7.3 (MuLisp)

```
(defun insert-lets (w a b)
  (mapcar #'(lambda (x) (if (eql x a) (pack (list a b)) x)) w))

> (insert-lets '(ab abc ac abd) 'abc 12)
(AB ABC12 AC ABD)
```

Задача 14.8 replace-elts.lisp

Заменить в исходном списке первое вхождение заданного значения другим.

Решение 14.8.1

```
(defun replace-elts (n m w)
  (nsubstitute n m w :count 1))

> (replace-elts 'z 'a '(a a))
(Z A)
```

Задача 14.9 sort-cadr.lisp

Отсортировать список вида ((A 3) (B 2) (C 5)) так, чтобы получилось ((C 5) (A 3) (B 2))

Решение 14.9.1

```
> (sort-cadr '((A 3) (B 2) (C 5)) #'> :key #'second)
((C 5) (A 3) (B 2))
```

Решение 14.9.2

```
> (sort-cadr '((A 3) (B 2) (C 5)) #'> :key #'cadr)
((C 5) (A 3) (B 2))
```

Задача 14.10 insert-on .lisp

Написать программу, которая добавляет элемент в список и ставит его за заданным элементом.

Решение 14.10.1

```
(defun insert-on (n m w)
  (cond ((null w) nil)
        ((eq (car w) n) (cons n (cons m (insert-on n m (cdr w)))))
        ((cons (car w) (insert-on n m (cdr w))))))
```

```
> (insert-on 3 100 '(1 2 3 4 5))
(1 2 3 100 4 5)
```

*Задача 14.11 a-*2-change .lisp*

Дан список произвольной длины, который состоит из числовых и символьных атомов. Определить функцию, которая числовые атомы увеличивает в 2 раза, а символьные меняет на атомы A

Решение 14.11.1

```
(defun a-*2-change (w a n)
  (cond ((null w) nil)
        ((numberp (car w)) (cons (* (car w) n)
                                 (a-*2-change (cdr w) a n)))
        (t (cons a (a-*2-change (cdr w) a n)))))

> (a-*2-change '(1 b 3 c 5 f) 'a 2)
(2 A 6 A 10 A)
```

Задача 14.12 dive-n+m .lisp

Дан многоуровневый список. Подсчитать количество атомов-чисел на первом и третьем уровнях вложенности.

Решение 14.12.1

```
(defun dive-count (n w &aux (a (car w)) (d (cdr w)))
   (cond ((null w) 0)
         ((= n 0) (count-if #'numberp w))
         ((atom a) (dive-count n d))
         ((+ (dive-count (1- n) a) (dive-count n d)))))

(defun dive-n+m (n m w)
  (+ (dive-count n w) (dive-count m w)))

> (dive-n+m 1 3 '(a(b(c (1))d e)f((2(g))3)))
2
```

Задача 14.13 del-lev-n.lisp

Заданы глубина подсписка и позиция. Удалить из всех имеющихся подсписков заданной глубины элементы, находящиеся на указанной позиции.

Решение 14.13.1

```
(defun del-lev-n (w n m)
  (when w ((lambda (a d)
             (cond ((atom a) (cons a (del-lev-n d n m)))
                   ((= n 0) (cons (del-n a m) (del-lev-n d n m)))
```

```
                  ((cons (del-lev-n a (1- n) m) (del-lev-n d n m)))))
          (car w) (cdr w))))

(defun del-n (w n)
  (cond ((null w) nil)
        ((= n 1) (cdr w))
        (t (cons (car w) (del-n (cdr w) (1- n))))))

> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 0 2)
(A B (1 3 (1 2 3 (1 2 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 1 2)
(A B (1 2 3 (1 3 (1 2 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 2 2)
(A B (1 2 3 (1 2 3 (1 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 3 2)
(A B (1 2 3 (1 2 3 (1 2 3))) C D)
```

Решение 14.13.2

```
(defun del-lev-n (w n m &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        ((atom a) (cons a (del-lev-n d n m)))
        ((= n 0) (cons (del-n a m) (del-lev-n d n m)))
        ((cons (del-lev-n a (1- n) m) (del-lev-n d n m)))))

(defun del-n (w n)
  (when w (if (= n 1)
              (cdr w)
              (cons (car w) (del-n (cdr w) (1- n))))))

> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 0 2)
(A B (1 3 (1 2 3 (1 2 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 1 2)
(A B (1 2 3 (1 3 (1 2 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 2 2)
(A B (1 2 3 (1 2 3 (1 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 3 2)
(A B (1 2 3 (1 2 3 (1 2 3))) C D)
```

Решение 14.13.3

```
(defun del-lev-n (w n m &aux (a (car w)) (d (cdr w)))
  (cond ((null w) nil)
        (t (cons (cond ((atom a) a)
                       ((= n 0) (del-n a m))
                       ((del-lev-n a (1- n) m)))
                 (del-lev-n d n m)))))

(defun del-n (w n)
  (cond ((null w) nil)
        ((= n 1) (cdr w))
        (t (cons (car w) (del-n (cdr w) (1- n))))))
```

```
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 0 2)
(A B (1 3 (1 2 3 (1 2 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 1 2)
(A B (1 2 3 (1 3 (1 2 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 2 2)
(A B (1 2 3 (1 2 3 (1 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 3 2)
(A B (1 2 3 (1 2 3 (1 2 3))) C D)
```

Решение 14.13.4

```
(defun del-lev-n (w n m &aux (a (car w)) (d (cdr w)))
  (when w (cons (cond ((atom a) a)
                      ((= n 0) (del-n a m))
                      ((del-lev-n a (1- n) m)))
                (del-lev-n d n m))))

(defun del-n (w n)
  (when w (if (= n 1)
              (cdr w)
              (cons (car w) (del-n (cdr w) (1- n))))))
```

```
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 0 2)
(A B (1 3 (1 2 3 (1 2 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 1 2)
(A B (1 2 3 (1 3 (1 2 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 2 2)
(A B (1 2 3 (1 2 3 (1 3))) C D)
> (del-lev-n '(a b (1 2 3 (1 2 3 (1 2 3))) c d) 3 2)
(A B (1 2 3 (1 2 3 (1 2 3))) C D)
```

Задача 14.14 atoms-n-m.lisp

Определить функцию, возвращающую список атомов в заданном интервале уровней. Функция принимает три параметра: список, начальный уровень и конечный уровень.

Решение 14.14.1

```
(defun atom-level (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (cond ((null w)  nil)
        ((atom a) (cons (list a n) (atom-level d n)))
        ((nconc (atom-level a (1+ n)) (atom-level d n)))))

(defun atoms-n-m (w n m)
  (loop for a in (atom-level w)
        when (<= n (cadr a) m)
        collect (car a)))
```

```
> (atoms-n-m '(a b (((c d)) e f)) 0 3)
(A B C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 1 3)
```

```
(C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 2 3)
(C D)
```

Решение 14.14.2

```
(defun atom-level (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (cond ((null w)  nil)
        ((atom a) (cons (list a n) (atom-level d n)))
        ((nconc (atom-level a (1+ n)) (atom-level d n)))))

(defun atoms (w n m)
  (cond ((null w) nil)
        ((<= n (cadar w) m) (cons (caar w) (atoms (cdr w) n m)))
        ((atoms (cdr w) n m))))

(defun atoms-n-m (w n m)
  (atoms (atom-level w) n m))

> (atoms-n-m '(a b (((c d)) e f)) 0 3)
(A B C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 1 3)
(C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 2 3)
(C D)
```

Решение 14.14.3

```
(defun atom-level (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (cond ((null w)  nil)
        ((atom a) (cons (list a n) (atom-level d n)))
        ((nconc (atom-level a (1+ n)) (atom-level d n)))))

(defun atoms (w n m &aux (a (car w)))
  (cond ((null w) nil)
        ((<= n (cadr a) m) (cons (car a) (atoms (cdr w) n m)))
        ((atoms (cdr w) n m))))

(defun atoms-n-m (w n m)
  (atoms (atom-level w) n m))

> (atoms-n-m '(a b (((c d)) e f)) 0 3)
(A B C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 1 3)
(C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 2 3)
(C D)
```

Решение 14.14.4

```
(defun atom-level (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (when w (nconc (if (atom a) (list (list a n)) (atom-level a (1+ n)))
                 (atom-level d n))))
```

```
(defun atoms (w n m &aux (a (car w)))
  (cond ((null w) nil)
        ((<= n (cadr a) m) (cons (car a) (atoms (cdr w) n m)))
        ((atoms (cdr w) n m))))

(defun atoms-n-m (w n m)
  (atoms (atom-level w) n m))

> (atoms-n-m '(a b (((c d)) e f)) 0 3)
(A B C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 1 3)
(C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 2 3)
(C D)
```

Решение 14.14.5

```
(defun atom-level (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (when w (nconc (if (atom a) `((,a ,n)) (atom-level a (1+ n)))
                 (atom-level d n))))

(defun atoms (w n m &aux (a (car w)))
  (cond ((null w) nil)
        ((<= n (cadr a) m) (cons (car a) (atoms (cdr w) n m)))
        ((atoms (cdr w) n m))))

(defun atoms-n-m (w n m)
  (atoms (atom-level w) n m))

> (atoms-n-m '(a b (((c d)) e f)) 0 3)
(A B C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 1 3)
(C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 2 3)
(C D)
```

Решение 14.14.6

```
(defun atom-level (w &optional (n 0) &aux (a (car w)) (d (cdr w)))
  (when w (nconc (if (atom a) `((,a ,n)) (atom-level a (1+ n)))
                 (atom-level d n))))

(defun atoms-n-m (w n m)
  (loop for a in (atom-level w)
        when (<= n (cadr a) m)
        collect (car a)))

> (atoms-n-m '(a b (((c d)) e f)) 0 3)
(A B C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 1 3)
(C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 2 3)
```

```
(C D)
```

Решение 14.14.7

```
(defun atom-level (w &optional (n 0) &aux (a (car w)))
  (when w (nconc (if (atom a) `((,a ,n)) (atom-level a (1+ n)))
                 (atom-level (cdr w) n))))

(defun atoms-n-m (w n m)
  (loop for a in (atom-level w)
        when (<= n (cadr a) m)
        collect (car a)))

> (atoms-n-m '(a b (((c d)) e f)) 0 3)
(A B C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 1 3)
(C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 2 3)
(C D)
```

Решение 14.14.8

```
(defun atoms-n-m (w n m)
  (loop for a in
        (labels ((atom-level (v n)
                  (when v (nconc (if (atom (car v))
                                     `((,(car v) ,n))
                                     (atom-level (car v) (1+ n)))
                                 (atom-level (cdr v) n)))))
          (atom-level w 0))
        when (<= n (cadr a) m)
        collect (car a)))

> (atoms-n-m '(a b (((c d)) e f)) 0 3)
(A B C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 1 3)
(C D E F)
> (atoms-n-m '(a b (((c d)) e f)) 2 3)
(C D)
```

Задача 14.15 remove-n-m.lisp

Определить функцию, удаляющую из списка все элементы с n-ного по m-ный.

Решение 14.15.1

```
(defun remove-n-m (w n m &optional (c 0))
  (cond ((null w) nil)
        ((<= n c m) (remove-n-m (cdr w) n m (1+ c)))
        ((cons (car w) (remove-n-m (cdr w) n m (1+ c))))))
```

```
> (remove-n-m '(a b c d e f g h) 2 5)
(A B G H)
```

Решение 14.15.2

```
(defun remove-n-m (w n m &optional (c 0))
  (when w (if (<= n c m)
              (remove-n-m (cdr w) n m (1+ c))
              (cons (car w)
                    (remove-n-m (cdr w) n m (1+ c))))))
```

```
> (remove-n-m '(a b c d e f g h) 2 5)
(A B G H)
```

Решение 14.15.3

```
(defun remove-n-m (w n m)
  (loop for a in w
        for b upto (1- (length w))
        unless (<= n b m) collect a))
```

```
> (remove-n-m '(a b c d e f g h) 2 5)
(A B G H)
```

Решение 14.15.4

```
(defun remove-n-m (w n m &optional (c 0))
  (when w (nconc (unless (<= n c m) `(,(car w)))
                 (remove-n-m (cdr w) n m (1+ c)))))
```

```
> (remove-n-m '(a b c d e f g h) 2 5)
(A B G H)
```

Структура 15 (функция (СПИСОК) (СПИСОК) АТОМ) > АТОМ

Задача 15.1 capital.lisp

Дан список стран x и список столиц этих стран y. По заданной стране a выдать название столицы.

Решение 15.1.1

```
(defun capital (c w v &aux (m (position c w)))
  (when m (nth m v)))
```

```
> (capital 'Germany '(RF USA Germany UK) '(Moscow Washington  Berlin
London))
BERLIN
```

Решение 15.1.2

```
(defun capital (c w v)
  (car (loop for a in w
             for b in v when (equal a c) collect b)))

> (capital 'Germany '(RF USA Germany UK) '(Moscow Washington  Berlin
London))
BERLIN
> (capital 'Hungary '(RF USA Germany UK) '(Moscow Washington  Berlin
London))
NIL
```

Решение 15.1.4

```
(defun capital (c w v)
  ((lambda (a b)
    (cond ((null w) nil)
          ((eq a c) b)
          ((capital c (cdr w) (cdr v)))))
  (car w) (car v)))

> (capital 'Germany '(RF USA Germany UK) '(Moscow Washington  Berlin
London))
BERLIN
```

Решение 15.1.3

```
(defun capital (c w v)
  (car (loop for (a b) in (mapcar #'list w v)
             when (eq a c) collect b)))

> (capital 'Germany '(RF USA Germany UK) '(Moscow Washington  Berlin
London))
BERLIN
```

Структура 16 (функция (СПИСОК) (СПИСОК) АТОМ) > (СПИСОК)

Задача 16.1 gen-arrangement.lisp

Генерация всех возможных расстановок знаков арифмтических операций в заданном шестизначном числе так, чтобы результатом полученного выражения было число 100 (цифры исходного числа допускается через пробел)

Решение 16.1.1

```
(defun pm (v &optional r)
  (dolist (a v r)
    (dolist (b v)
      (dolist (c v)
        (dolist (d v)
```

```
          (dolist (e v)
            (push (list a b c d e) r)))))))

(defun mx (w v)
  (when w (cons (car w) (mx v (cdr w)))))

(defun th (w p)
  (subseq w (1- p) (+ p 2)))

(defun ta (w p)
  (subseq w (+ p 2)))

(defun tb (w p)
  (subseq w 0 (1- p)))

(defun pu (w)
  (list (cadr w) (car w) (caddr w)))

(defun sb (z w &aux (p (position z w)))
  (append (tb w p) (list (eval (pu (th w p)))) (ta w p)))

(defun ev (w)
  (cond ((null (cdr w)) (car w))
        ((member '/ w) (ev (sb '/ w)))
        ((member '* w) (ev (sb '* w)))
        ((member '+ w) (ev (sb '+ w)))
        ((ev (sb '- w)))))

(defun gen-arrangement (w v z)
  (mapcar
   #'(lambda (e)
       (mx w e)) (mapcan
                  #'(lambda (a)
                      (when (eq (ev (mx w a)) z)
                        (list a)))
                  (pm v))))

> (gen-arrangement '(5 2 3 4 5 6) '(+ - * /) Knapsack)
((5 / 2 / 3 * 4 * 5 * 6) (5 * 2 * 3 * 4 * 5 / 6))
> (gen-arrangement '(5 2 3 4 5 6) '(+ - * /) 50)
((5 + 2 * 3 / 4 * 5 * 6))
> (gen-arrangement '(5 2 3 4 5 6) '(+ - * /) 40)
((5 + 2 * 3 * 4 + 5 + 6))
```

Примеры по функциям:

```
> (pm '(+ - * /))
((/ / / /) (/ / / / *) ... (+ + + + *) (+ + + + -) (+ + + + +))
```

mx - смешивает поочередно знаки и числа:

```
> (mx '(2 5 7 4 2 4) '(+ * - / +))
```

```
(2 + 5 * 7 - 4 / 2 + 4);

> (mx '(5 2 3 4 5 6) '(+ + + + *))
(5 + 2 + 3 + 4 + 5 * 6)

> (th '(5 + 2 + 3 + 4 + 5 * 6) 9)
(5 * 6)

> (ta '(5 + 2 + 3 + 4 + 5 * 6) 9)
NIL

> (tb '(5 + 2 + 3 + 4 + 5 * 6) 9)
(5 + 2 + 3 + 4 +)

> (pu '(5 * 6))
(* 5 6)

> (sb '* '(5 + 2 + 3 + 4 + 5 * 6))
(5 + 2 + 3 + 4 + 30)

> (ev '(5 + 2 + 3 + 4 + 5 * 6))
44
```

Задача 16.2 insert-els.lisp

Напишите функцию, которая вставляла бы на заданное место элементы второго списка-аргумента.

Решение 16.2.1

```
(defun insert-els (w v n)
  (cond ((zerop n) (append v w))
        ((cons (car w) (insert-els (cdr w) v (1- n))))))

> (insert-els '(1 2 3) '(a b) 2)
(1 2 A B 3)
```

Решение 16.2.2

```
(defun insert-els (w v n)
  (cond ((null w) nil)
        ((> n 0) (cons (car w) (insert-els (cdr w) v (1- n))))
        ((nconc v w))))

> (insert-els '(1 2 3) '(a b) 2)
(1 2 A B 3)
```

Решение 16.2.3

```
(defun insert-els (w v n)
  (if (zerop n)
      (nconc v w)
```

```
      (cons (car w) (insert-els (cdr w) v (1- n)))))

> (insert-els '(1 2 3) '(a b) 2)
(1 2 A B 3)
```

Решение 16.2.4

```
(defun insert-els (w v n)
  (nconc (subseq w 0 n) v (subseq w n)))

> (insert-els '(1 2 3) '(a b) 2)
(1 2 A B 3)
```

Задача 16.3 insert-list.lisp

Опишите функцию, которая вставляла заданное место списка другой список.

Решение 16.3.1

```
(defun insert-list (w v n)
  (cond ((null w) (cons v nil))
        ((> n 0) (cons (car w) (insert-list (cdr w) v (- n 1))))
        ((cons v  w))))

> (insert-list '(1 2 3) '(a b) 2)
(1 2 (A B) 3)
```

Задача 16.4 bus-ticket.lisp

Дан автобусный билет с номером, состоящим из 6 цифр. Расставить между цифрами знаки арифметических операций '+', '-', '*', '/' (целочисленное деление) таким образом, чтобы значение полученного выражения было равно 100.

Решение 16.4.1

```
(defun pm (v &optional r)
  (dolist (a v r)
    (dolist (b v)
      (dolist (c v)
        (dolist (d v)
          (dolist (e v)
            (push (list a b c d e) r)))))))

(defun mx (w v)
  (when w (cons (car w) (mx v (cdr w)))))

(defun th (w p)
  (subseq w (1- p) (+ p 2)))
```

```
(defun ta (w p)
  (subseq w (+ p 2)))

(defun tb (w p)
  (subseq w 0 (1- p)))

(defun pu (w)
  (list (cadr w) (car w) (caddr w)))

(defun sb (z w &aux (p (position z w)))
  (append (tb w p) (list (eval (pu (th w p)))) (ta w p)))

(defun ev (w)
  (cond ((null (cdr w)) (car w))
        ((member '/ w) (ev (sb '/ w)))
        ((member '* w) (ev (sb '* w)))
        ((member '+ w) (ev (sb '+ w)))
        ((ev (sb '- w)))))

(defun bus-ticket (w v z)
  (mapcar
   #'(lambda (e)
       (mx w e)) (mapcan
                  #'(lambda (a)
                      (when (eq (ev (mx w a)) z)
                        (list a)))
                  (pm v))))

> (bus-ticket '(5 2 3 4 5 6) '(+ - * /) 100)
((5 / 2 / 3 * 4 * 5 * 6) (5 * 2 * 3 * 4 * 5 / 6))
```

Задача 16.5 run-function.lisp

Даны списки v и w. В списке v все элементы различные. Определить функцию высшего порядка, которая выдает список результатов применения функции к элементу из v и списку w.

Решение 16.5.1

```
(defun run-function (f v w)
  (mapcar #'(lambda (a) (funcall f a w)) v))

> (run-function #'count '(a b) '(c a b a d))
(2 1)
```

Задача 16.6 insert-elms.lisp

Список w, элементами которого являются списки. Заменить n-й элемент списка w элементами списка v.

Решение 16.6.1

```
(defun insert-elms (w n v)
  (cond ((zerop n) (nconc v (cdr w)))
        ((cons (car w) (insert-elms (cdr w) (1- n) v)))))

> (insert-elms '((a b) (c d) (e f)) 1 '(x y z))
((A B) X Y Z (E F))
```

Решение 16.6.2

```
(defun insert-elms (w n v)
  (if (zerop n)
      (nconc v (cdr w))
      (cons (car w) (insert-elms (cdr w) (1- n) v))))

> (insert-elms '((a b) (c d) (e f)) 1 '(x y z))
((A B) X Y Z (E F))
```

Задача 16.7 insert.lisp

Дан список, который состоит из подсписков. Напишите функцию, которая вставляла бы на заданное место элементы второго списка-аргумента во все подсписки данного списка.

Решение 16.7.1

```
(defun insert (v n w)
  (when w (cons (insert-elms v n (car w)) (insert v n (cdr w)))))

(defun insert-elms (v n w)
  (cond ((zerop n) (ncon v w))
        ((cons (car w) (insert-elms v (1- n) (cdr w))))))

> (insert '(9 10) 2 '((1 2 3) (4 5 6) (7 8 9)))
((1 2 9 10 3) (4 5 9 10 6) (7 8 9 10 9))
```

Задача 16.8 insert-after.lisp

Дан список, который состоит из подсписков. Напишите функцию, которая вставляла бы после n-х элементов каждого подсписка элементы второго списка-аргумента.

Решение 16.8.1

```
(defun insert-after (v n w)
  (mapcar #'(lambda (a)
              (if (and (listp a) (nth n a ))
                  (insert-elms v n a)
                  a))
          w))
```

```
(defun insert-elms (v n w)
  (if (minusp n)
      (append v w)
      (cons (car w) (insert-elms v (1- n) (cdr w)))))

> (insert-after '(9 10) 1 '((1) (4 5) (7 8 9)))
((1) (4 5 9 10) (7 8 9 10 9))
> (insert-after '(9 10) 1 '((1) (4 5) (7 8 9) 11))
((1) (4 5 9 10) (7 8 9 10 9) 11)
```

Решение 16.8.2

```
(defun insert (v n w)
  (mapcar #'(lambda (a)
              (if (listp a) (insert-elms v n a) a)) w))

(defun insert-elms (v n w)
  (if (minusp n)
      (append v w)
      (cons (car w) (insert-elms v (1- n) (cdr w)))))

> (insert '(9 10) 1 '((1) (4 5) (7 8 9) 11))
((1 NIL 9 10) (4 5 9 10) (7 8 9 10 9) 11)
```

Задача 16.9 dive-change.lisp

Написать программу замены атомов на заданном уровне новыми атомами. Новые атомы вводить в виде списка. Атомы, содержащиеся в исходном списке имеют уровень 0, атомы, содержащиеся в следующем вложенном списке, имеют уровень 1 и т.д.

Решение 16.9.1

```
(defparameter c -1)

(defun dive-change (n w v)
  (progn (setf c -1) (dive n w v)))

(defun dive (n w v)
  (cond ((null w) nil)
        ((and (atom (car w)) (zerop n))
         (setf c (1+ c)) (cons (elt v c) (dive n (cdr w) v)) )
        ((listp (car w))
         (cons (dive (1- n) (car w) v) (dive n (cdr w) v)))
        ((cons (car w) (dive n (cdr w) v)))))


> (dive-change 0 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(A (2 (3)) ((4)) ((5)))
> (dive-change 0 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(A (2 (3)) ((4)) ((5)))
> (dive-change 1 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
```

```
(1 (A (3)) ((4)) ((5)))
> (dive-change 2 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(1 (2 (A)) ((B)) ((C)))
> (dive-change 3 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(1 (2 (3)) ((4)) ((5)))
```

Решение 16.9.2

```
(defparameter c -1)

(defun dive-change (n w v)
  (progn (setf c -1) (dive n w v)))

(defun dive (n w v)
  (cond ((null w) nil)
        ((and (atom (car w)) (zerop n))
         (setf c (1+ c)) (cons (nth c v) (dive n (cdr w) v)) )
        ((cons (if (listp (car w))
                   (dive (1- n) (car w) v)
                   (car w))
               (dive n (cdr w) v)))))
```

```
> (dive-change 0 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(A (2 (3)) ((4)) ((5)))
> (dive-change 0 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(A (2 (3)) ((4)) ((5)))
> (dive-change 1 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(1 (A (3)) ((4)) ((5)))
> (dive-change 2 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(1 (2 (A)) ((B)) ((C)))
> (dive-change 3 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(1 (2 (3)) ((4)) ((5)))
```

Решение 16.9.3

```
(defparameter c -1)

(defun dive-change (n w v)
  (progn (setf c -1) (dive n w v)))

(defun dive (n w v)
  (cond ((null w) nil)
        ((listp (car w))
         (cons (dive (1- n) (car w) v) (dive n (cdr w) v)))
        ((zerop n)
         (setf c (1+ c))
         (cons (elt v c) (dive n (cdr w) v)))
        ((cons (car w) (dive n (cdr w) v)))))
```

```
> (dive-change 0 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(A (2 (3)) ((4)) ((5)))
> (dive-change 0 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(A (2 (3)) ((4)) ((5)))
```

```
> (dive-change 1 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(1 (A (3)) ((4)) ((5)))
> (dive-change 2 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(1 (2 (A)) ((B)) ((C)))
> (dive-change 3 '(1 (2 (3)) ((4)) ((5))) '(a b c d e))
(1 (2 (3)) ((4)) ((5)))
```

Структура 17 (функция (СПИСОК) (СПИСОК) (СПИСОК)) > (СПИСОК)

Задача 17.1 replace-elms.lisp

Даны три списка, элементами которых являются атомы. Заменить в первом списке все вхождения элементов из второго списка на элементы из третьего соответствующими порядковыми номерами.

Решение 17.1.1

```
(defun replace-elms (v1 v2 v3)
  (mapcar #'(lambda (a) (if
                          (member a v2)
                          (nth (position a v2) v3)
                          a))
          v1))

> (replace-elms '(1 2 3 a b c) '(3 a b) '(33 aa bb))
(1 2 33 AA BB C)
```

Задача 17.2 remove-numbers.lisp

Даны два списка и третий - контрольный. Составить программу, которая удаляет из обоих списков элементы, которые входят в контрольный список. Например, контрольный (1 5 7 8), два других (4 5 3 8) и (1 2 3 6 8), из них получить списки (4 3) и (2 3 6).

Решение 17.2.1

```
(defun clear (w z)
  (remove-if #'(lambda (a) (member a z)) w))

(defun remove-numbers (w v z)
  (values (clear w z) (clear v z)))

> (remove-numbers '(4 5 3 8) '(1 2 3 6 8) '(1 5 7 8))
(4 3)
(2 3 6)
```

Задача 17.3 unique-elms.lisp

Заданы три списка. Сформировать из них новый список, элементы которого это - элементы каждого исходного, не повторяющиеся в двух других.

Решение 17.3.1

```
(defun unique (v vv vvv)
  (cond ((null v) nil)
        ((or (member (car v) vv) (member (car v) vvv))
         (unique (cdr v) vv vvv))
        ((cons (car v) (unique (cdr v) vv vvv)))))

(defun unique-elms (a b c)
  (nconc (unique a b c) (unique b c a) (unique c a b)))

> (unique-elms '(1 2 3) '(3 4 5) '(5 6 7))
(1 2 4 6 7)
```

Решение 17.3.2 (muLISP)

```
(setq lst1 '(1 2 3 4))
(setq lst2 '(6 3 8))
(setq lst3 '(9 2 1))

(defun unique (v vv vvv)
  (cond ((null v) nil)
        ((or (member (car v) vv) (member (car v) vvv))
         (unique (cdr v) vv vvv))
        ((cons (car v) (unique (cdr v) vv vvv)))))

(defun unique-elms (a b c)
  (append (unique a b c) (unique b c a) (unique c a b)))

> (unique-elms lst1 lst2 lst3)
(4 6 8 9)
```

Задача 17.4 place-cons.lisp

Расставить cons в выражении,чтобы объединить три элемента (car (cdr '(y u i))) (car (cdr '(g1 g2 g3))) (car(cdr(cdr '(kk ll mm jjj)))). Нужно объединить первое выражение со вторым, второе с третьим. Написать для этого именованную функцию.

Решение 17.4.1

```
(defun place-cons (a b c)
  (cons a (cons b (cons c nil))))

> (place-cons (car(cdr '(y u i))) (car(cdr '(g1 g2 g3))) (car(cdr(cdr
'(kk ll mm jjj)))))
(U G2 MM)
```

Решение 17.4.2

```
(defun place-cons (a b c)
 (cons (car (cdr a))
       (cons (car (cdr b))
             (cons (car (cdr (cdr c)))
                   nil))))

> (place-cons '(y u i) '(g1 g2 g3) '(kk ll mm jjj))
(U G2 MM)
```

Решение 17.4.3

```
(defun place-cons (a b c)
 (cons (cadr a) (cons (cadr b) (cons (caddr c) nil))))

> (place-cons '(y u i) '(g1 g2 g3) '(kk ll mm jjj))
(U G2 MM)
```

Задача 17.5 replace-a-b.lisp

Определить функцию, заменяющую все вхождения одного списка или атома на другой список или атом.

Решение 17.5.1

```
(defun replace-a-b (w a b)
  (cond ((null w) nil)
        ((equalp (car w) a) (cons b (replace-a-b (cdr w) a b)))
        ((listp (car w)) (cons (replace-a-b (car w) a b)
                               (replace-a-b (cdr w) a b)))
        ((cons (car w) (replace-a-b (cdr w) a b)))))

> (replace-a-b '(a c ((b ((c)) d ))) 'c '(1 2 3))
(A (1 2 3) ((B (((1 2 3))) D)))
> (replace-a-b '(A (1 2 3) ((B (((1 2 3))) D))) '(1 2 3) 'c)
(A C ((B ((C)) D)))
```

Решение 17.5.2

```
(defun replace-a-b (w a b)
  (when w
    (cons (cond ((equalp (car w) a) b)
                ((listp (car w)) (replace-a-b (car w) a b))
                (t (car w)))
          (replace-a-b (cdr w) a b))))

> (replace-a-b '(a c ((b ((c)) d ))) 'c '(1 2 3))
(A (1 2 3) ((B (((1 2 3))) D)))
> (replace-a-b '(A (1 2 3) ((B (((1 2 3))) D))) '(1 2 3) 'c)
(A C ((B ((C)) D)))
```

Структура 18 (функция (СПИСОК) (СПИСОК) АТОМ АТОМ) > (СПИСОК)

Задача 18.1 balance.lisp

Даны два числовых списка w и v, k и n - целые числа (k, n = 1, 2, 3). Получить разность между k-м элементом списка w и n-м элементом списка v.

Решение 18.1.1

```
(defun n-element (n w)
  (cond ((null w) nil)
        ((zerop n) (car w))
        ((n-element (- n 1) (cdr w)))))

(defun balance (w v k n)
  (- (n-element k w) (n-element n v)))

> (balance '(2 4 6) '(1 3 5) 0 0)
1
```

Задача 18.2 run-over.lisp

Даны два числовых одноуровневых списка x=(x1, x2....xn) и y=(y1,y2....,yn) одинаковой длины, g - некоторая функция от двух переменных, определенная в области чисел (умножение, сложение, возведение в степень и.т.д.). Например, g(x,y)=x*y, где x и y - числа. Определить функционал (f g a x y), где g - функциональный аргумент, a - заданное число, a x и y - числовые списки, выдающий в качестве значения новый список, образованный из чисел g(xi,yi), не превосходящих заданное a (xi,yi - элементы исходных списков).

Решение 18.2.1

```
(defun drop>n (w n)
  (cond ((null w) nil)
        ((< (car w) n) (cons (car w) (drop>n (cdr w) n)))
        ((drop>n (cdr w) n))))

(defun run-over (g a x y)
  (drop>n (mapcar g x y) a))

> (run-over #'* 50 '(1 2 3) '(11 22 33))
(11 44)
```

Решение 18.2.2

```
(defun drop>n (w n)
  (cond ((null w) nil)
        ((< (car w) n) (cons (car w) (drop>n (cdr w) n)))
        ((drop>n (cdr w) n))))
```

```
(defun run-over (g a x y)
  (drop>n (mapcar (intern g) x y) a))

> (run-over "*" 50 '(1 2 3) '(11 22 33))
(11 44)
```

Решение 18.2.3

```
(defun drop>n (w n)
  (when w (if (< (car w) n)
              (cons (car w) (drop>n (cdr w) n))
              (drop>n (cdr w) n))))

(defun run-over (g a x y)
  (drop>n (mapcar (intern g) x y) a))

> (run-over "*" 50 '(1 2 3) '(11 22 33))
(11 44)
```

Решение 18.2.4

```
(defun run-over (g a x y)
  (remove-if #'(lambda (e) (> e a)) (mapcar g x y)))

> (run-over #'* 50 '(1 2 3) '(11 22 33))
(11 44)
```

Задача 18.3 divided.lisp

Заданы списки целых чисел, вернуть список чисел, которые делятся на два заданных числа.

Решение 18.3.1

```
(defun divided (n m &rest w)
  (remove-if-not
   #'(lambda (a)
       (and (zerop (mod a n))
            (zerop (mod a m))))
   (reduce #'nconc w)))

> (divided 2 3 '(6 12) '(6 7) '(11 12))
(6 12 6 12)
```

Задача 18.4 divided-infinity.lisp

Дан список делителей и произвольное количество числовых списков. Вернуть список чисел, которые делятся без остатка на каждый из делителей.

Решение 18.4.1

```
(defun divided-infinity (v &rest w)
  (remove-if-not
   #'(lambda (a)
       (every #'(lambda (b) (zerop (mod a b))) v))
       (reduce #'nconc w)))

> (divided-infinity '(2 3 5) '(30 40) '(300 400))
(30 300)
```

Задача 18.5 xinsert-elms.lisp

Получить "простой" список включением элементов списка v между n-ым и n+1-ым элементами списка w, если некоторый объект a совпадает с k-ым элементом списка v.

Решение 18.5.1

```
(defun xinsert-elms (w v n a k)
  (if (equalp (nth k v) a)
      (append (subseq w 0 n)
              (cons (nth n w) v)
              (nthcdr (1+ n) w))
      w))

> (xinsert-elms '(1 2 3 4 5) '(a b) 2 'b 1)
(1 2 3 A B 4 5)
```

Литература

Абельсон Х., Сассман Д. Структура и интерпретация компьютерных программ. – М.: Добросвет, 2004. – 576 с. – англ. – Mode of access: http://mitpress.mit.edu/sicp/ Видеозаписи лекций – Mode of access: http://groups.csail.mit.edu/mac/classes/6.001/abelson-sussman-lectures/

Хювёнен Э., Сеппянен И. Мир Лиспа. В 2-х т. Т.1: Введение в язык Лисп и функциональное программирование. – М.: Мир, 1990. – 439 с. – Mode of access: http://lisp.ru/page.php?id=17

Allen Colin, Dhagat Maneesh. LISP Primer. – 1992. – 134 p. – Mode of access: http://www.svbug.com/documentation/lisp/

Burgemeister Bert. Common Lisp Quick Reference. – 2011. – 52 p. – Mode of access: http://clqr.boundp.org/clqr-a4-booklet-all.pdf

Cooper David J., Jr. Basic Lisp Techniques. 2003. – 92 p. – Mode of access: http://www.franz.com/resources/educational_resources/cooper.book.pdf

Graham Paul. ANSI Common Lisp. – Upper Saddle River, NJ: Prentice Hall, 1995, 432 p. – chapters 1-3. – Mode of access: http://www.paulgraham.com/acl.html. Перевод на русский язык: Грэм П. ANSI Common Lisp. - СПб.: Символ-Плюс, .2012. - 448 с.

Graham Paul. On Lisp. – Upper Saddle River, NJ: Prentice Hall, 1993, 413 p. – Mode of access: http://lib.store.yahoo.net/lib/paulgraham/onlisp.pdf

Common Lisp HyperSpec. – Mode of access: ftp://ftp.lispworks.com/pub/software_tools/reference/HyperSpec-7-0.tar.gz

Knott Gary D. Interpreting LISP. – 1997. – 103 p. – Mode of access: http://www.civilized.com/files/lispbook.pdf

Lamkins David B.Successful Lisp: How to Understand and Use Common Lisp. – 2004. – 360 p. – Mode of access: http://psg.com/~dlamkins/sl/contents.html

McCarthy et al. LISP 1.5 Programmer's Manual. The M.I.T. Press, 1962, – 106 p. – Mode of access: http://www.softwarepreservation.org/projects/LISP/book/LISP%201.5%20Programmers%20Manual.pdf

Norvig Peter, Pitman Kent. Tutorial on Good Lisp Programming Style. 1993. – 116. – Mode of access: http://obfusco.com/tmp/luv-slides.pdf

Norvig Peter. Paradigms of Artificial Intelligence Programming: Case Studies in Common Lisp. Morgan Kaufmann, 1992. - 946 p.

Pierce David R., Shapiro Stuart C. Common Lisp: A Brief Tutorial. – 2004 – 32 p. – Mode of access: http://curry.ateneo.net/~jpv/cs171/LispTutorial.pdf

Russell Stuart J., Norvig Peter. Artificial Intelligence A Modern Approach. – Englewood Cliffs, New Jersey: Prentice Hall, 1995. – 932 p.

Seibel P. Practical Common Lisp. New York, NY: Apress, 2005. – 499 p. – Mode of access: http://www.gigamonkeys.com/book/. Перевод на русский язык, PDF-версия – Mode of access: http://lisper.ru/pcl/pcl.pdf

Shapiro Stuart C. Common Lisp: An Interactive Approach. - N.Y.: Computer Science Press, 1992. – 322 p. – Mode of access: http://www.cse.buffalo.edu/~shapiro/Commonlisp/commonLisp.pdf

Steele Guy L. Common Lisp the Language, 2nd edition. Thinking Machines, Inc. Digital Press, 1990 – 1029 p. – Mode of access: http://www.cs.cmu.edu/Groups/AI/html/cltl/cltl2.html

Tanimoto Steven L. The Elements of Artificial Intelligence. An introduction Using Lisp. – Seattle, Washington: Computer Science Press, 1987. – 529 p.

Touretzky D.S. Common Lisp: A Gentle Introduction to Symbolic Computation. – Redwood City, California: Carnegie Mellon University: The Benjamin/Cummings Publishing Company. – 1990 – 587 p. – Mode of access: http://www.cs.cmu.edu/~dst/LispBook/book.pdf

Указатель функций

C

D

E

F

G

I

L

M

N

S

Для заметок

Горлянский Сергей Петрович

Решение задач на языке программирования Лисп

Учебно-методическое пособие по дисциплине
"Компьютерные технологии и историческая наука"

для студентов 5 курса дневной формы обучения и 6 курса заочной формы обучения специальностей – 8.030301 – «история» образовательно-квалификационного уровня «магистр» и 7.030301 – «история» образовательно-квалификационного уровня «специалист» профессионального направления подготовки 0203 «Гуманитарные науки»

Редактор: Н. А. Василенко

Подписано к печати 20.01.2012. Формат 60х84/16. Бумага тип. ОП.
Объем 16,1 п. л. Тираж – 100. Заказ –

95007, Симферополь, пр-т Академика Вернадского, 4.
Таврический национальный университет имени В. И. Вернадского