Примітивні об’єкти даних Lisp

Примітивними об’єктами даних є ***символи***, ***числа*** та ***конси***. Lisp має безліч функцій розпізнання, порівняння, комбінування та обробки цих об’єктів. Це дозволяє конструювати будь-які складні об’єкти даних. Як було сказано раніше, Lisp має два типи даних: атоми та списки. Атоми поділяються на символи та числа. Списки є підмножиною об’єктів, які мають більш загальну структуру – бінарне дерево. Вони створені за допомогою консів.

Символ є об’єктом даних, з яким пов’язано 4 атрибути, кожен з яких є вказівником на:

- PRINT-ім’я. Це унікальний рядок ASCII символів, за допомогою якого система ідентифікує символ при операціях введення-виведення. PRINT-ім’я не можна змінити. Імена обмежені за розміром: вони повинні мати не більше від 65536 символів.
- поточне значення. Значенням символа може бути будь-який об’єкт даних, який зберігається в комірці пам’яті. Якщо в середовищі Ліспу ввести PRINT-ім’я символу, то на виході буде його значення. Поточне значення доступно як CAR-елемент символа.
- список властивостей. Він містить значення властивостей символа, проіндексованих за ключем, його форма має вигляд: (ім’я1 значення1 ім’я2 значення2 ... ім’яN значенняN). При ініціалізації системи список властивостей є порожнім (дорівнює NIL). Його можна змінити за допомогою функцій властивостей та прапорців. Доступний як CDR-елемент символа.
- визначення функції. При створенні символу в Lisp цей атрибут дорівнює “функція невизначена”. Визначення функції складається або за шаблонами машинної мови, або на D-коді. Значення цього атрибута можна отримати в результаті виконання функції прапорців (GETD символ).

Не все з написаного у виділеному фрагменті реалізовано в LispWorks. Про список властивостей символа мова йтиме трохи згодом. Визначення функції у LispWorks повертає не GETD, а SYMPOL-FUNCTION.

SYMBOLP є функція, яка розпізнає символ. Вона повертає T, якщо аргумент є символом і NIL в протилежному випадку.

```
$ (SYMBOLP 'XYZ)            $ (SYMBOLP 41)
T                           NIL

$ (SYMBOLP '(q w))          $ (SYMBOLP '())
NIL                         T
```

У Common Lisp визначені функції **BOUNDP**, **SYMBOL-VALUE**, **SYMBOL-PLIST** для роботи з символами. BOUNDP розпізнає, чи пов’язано символ з якимось значенням, SYMBOL-VALUE повертає значення символа, а функція SYMBOL-PLIST повертає весь список властивостей символа (Property LIST).

Узагальненою функцією присвоєння є **SETF**. Вона заносить данні в комірку пам’яті символа: (SETF <комірка пам’яті> <значення>). Через функцію SETF можна представити описані раніше функції SET та SETQ.

```
(SETQ x y)      це    (SETF x y)
(SET x y)       це    (SETF (SYMBOL-VALUE x) y)
```

Проміжки, дужки, коми, одинарні та подвійні лапки, крапка, крапка з комою відіграють спеціальну роль в Ліспі. Одинарним Escape-символом є \. Багатократним Escape-символом є |. Спеціальні літери можуть використовуватися у PRINT-іменах символів, але для цього перед ними треба ставити \, або весь рядок брати в |. Вирази |q w e| та |sym(bol| є символами. Для використання літер \ та | в символах необхідно ставити перед ними \. Якщо виводиться на екран символ, який містить спеціальні літери, то він виводиться з багатократним escape-символом. Програмна змінна

PRINT-ESCAPE булевського типу відповідає за виведення escape-символів. Якщо вона дорівнює NIL, то escape-символи на екран не виводяться. Подвійні лапки “ грають роль літери |. Розглянемо приклади (спочатку *PRINT-ESCAPE*=T):

```
$ (SETQ |sym(bol| 3)          $ (SETQ a |q w e|)        $ s\a        $ s\\a
$ |sym(bol|                   $ a                       sa           |s\\a|
3                             |q w e|

$ (SETQ *PRINT-ESCAPE* NIL)          $ (SETQ a |q w e|)        $ (SETQ “s\\a” 2)
$ s\\a                               $ a                       $ |s\\a|
s\a                                  q w e                     2
```

Число є іншим примітивним об’єктом. Воно може бути ***цілим*** або ***раціональним***. Ціле число вводиться як послідовність цифр, перед якою може стояти знак мінус. За внутрішнім поданням цілі числа діляться на малі цілі (до 65536) та великі цілі. Оскільки значенням числа завжди є саме число, то немає необхідності перед ним ставити апостроф.

Чотири атрибути характеризують число як об’єкт даних:

- елемент тотожності. Це є вказівник на саме число. Він доступний як CAR-елемент числа.
- знак. Він містить один з наступних символів, які характеризують тип числа:

	додатне	від’ємне
мале	NIL	T
велике	LAMBDA	NLAMBDA
раціональне	MACRO	SPECIAL

```
$ (CDR 5.6)          $ (CAR 5.6)
MACRO                5.6

$ (CDR 1212)         $ (CDR -121212)
NIL                  NLAMBDA
```

Значення атрибута знака доступне як CDR-елемент числа.

- довжина. Якщо число є малим цілим, то цей атрибут містить значення цілого. Якщо число – велике ціле, то елемент "довжина" містить довжину слова вектора числа. Якщо число дробове – елемент містить вказівник на його чисельник, який обов’язково повинен бути цілим (додатним або від’ємним).
- вектор. Якщо число мале ціле, то значення атрибута є вказівником на інше мале ціле (хеш-з’єднувач). Якщо число велике ціле, то це поле містить вказівник на найменший значущий байт. Якщо число дробове – елемент містить вказівник на його знаменник, який повинен бути додатним цілим числом.

Функція порівняння EQL може використовуватися для порівняння чисел. Але більш загальною функцією для порівняння множини чисел є рівність:

```
$ (EQL -3 4)          $ (EQL 4 4)        $ (= 2 2 2)        $ (= 2 2 3 2)
NIL                   T                  T                  NIL
```

Раціональні числа можуть подаватися у десятковому вигляді та з дробовою рискою. Внутрішня змінна ***PRINT-POINT*** відповідає за тип виведення раціональних чисел. Якщо вона дорівнює NIL, то всі дробові числа подаються на виведення з дробовою рискою. Якщо *PRINT-POINT* = n, то дробові числа виводяться з n знаками після десяткової коми. При введенні дробового числа воно автоматично скорочується.

```
$ 3/4     $ 3/9       $ 5/1       $ 12/9
3/4       1/3         5           4/3
```

Внутрішня змінна ***PRINT-BASE*** відповідає за основу системи числення, в якій обробляються числа. Якщо значення цієї змінної є цілим та перебуває в інтервалі від 2 до 32, то такою і буде основа системи числення, інакше Lisp працює в десятковій системі числення.

```
$ (SETQ ten 10)                    $ (SETQ *PRINT-BASE* 2)        $ 234
$ (SETQ *PRINT-BASE* 16)           $ ten                          11101010
$ ten                              1010
0A
```

Функцією, яка розпізнає цілі числа, є **INTEGERP**. Вона повертає T, якщо її аргумент є цілим числом та NIL інакше. Функція **NUMBERP** розпізнає число.

```
$ (INTEGERP 100)            $ (INTEGERP 3.5)
T                           NIL
$ (NUMBERP 3.5)             $ (NUMBERP 4/5)
T                           T
```

Число в подвійних лапках завжди є символом:

```
$ (SYMBOLP “23”)            $ (NUMBERP “23”)
T                           NIL
```

Символи та числа є атомами. Наступні вирази повертають істину: (ATOM 3.5), (ATOM “23”), (ATOM 'APPLE).

Конс є примітивним об’єктом, який вказує на будь-які два інші об’єкти даних.. Він не є атомом. Назва конс пішла від функції конструктора CONS. Кожен конс складається з CAR- та CDR-елементів. Конс часто називають точковою парою. Якщо X і Y об’єкти даних, то вираз (X . Y) є консом, CAR-елемент якого є X, а CDR-елемент – Y.

```
$ (SETQ A (cons X Y))       $ (CAR A)       $ (CDR A)       $ (CDR '(R . S))
$ A                         X               Y               S
(X . Y)
```

За допомогою точкового подання можна показати структуру будь-якого об’єкту. Список (x1 x2 x3) є ланцюгом консів, які зв’язані за допомогою CDR-елементів. Його CAR-елементи вказують на елементи списку. CDR-елемент останнього конса вказує на NIL. Вказаний список можна подати у вигляді (x1 . (x2 . (x3 . NIL))). Функція READ читання виразу розпізнає як точкове подання виразу, так і спискове. Функція виведення PRINT виводить об’єкти в списковому поданні.

```
$ (SETQ a '(q . (w . nil))      $ a       $ (CONSP '(q . w))      $ (CONSP (q w))
(q w)                           (q w)     T                       T
```

Функція (CONSP obj) розпізнає конси. Список не є примітивним об’єктом, а є ланцюгом консів. Отже, результатом застосування функції CONSP до списку буде T.

Функції властивостей

Розглянемо, як можна працювати зі списком властивостей символа. Його можна по необхідності створювати, обробляти та видаляти. Властивості символа є глобальними, тобто доступними з довільної точки програми, поки вони не будуть явно змінені чи видалені. Використання символа в якості змінної чи імені функції не впливає на список властивостей.

Функції властивостей керують властивостями символів. CDR-елемент символа вказує на список властивостей. Разом з функціями прапорців вони полегшують процес побудови динамічних баз даних.

1. (**PUT** <символ> <ключ> <об’єкт>). У список властивостей <символа> кладеться значення <об’єкта> відповідно до вказівника <ключ>.

У LispWorks нема вбудованої функції PUT, проте її легко визначити за допомогою узагальненої форми присвоєння:

```
(DEFUN PUT (SYMBOL KEY VALUE) (SETF (GET SYMBOL KEY) VALUE))

$ (PUT 'capital 'usa 'washington)          $ (SETQ capital 'world)
$ (PUT 'capital 'germany 'bonn)            $ (PUT 'world 'ocean 'atlantic)
$ (PUT 'capital 'england 'london)
$ (SYMBOL-PLIST 'capital)
((ENGLAND . LONDON) (GERMANY . BONN) (USA . WASHINGTON))

$ (SYMBOL-PLIST capital)
((ocean . atlantic))

$ (SYMBOL-VALUE 'capital)           $ capital
world                               world
```

2. (**GET** <символ> <ключ>). Повертає значення властивості, яке відповідає <символу> відповідно до вказівника <ключ>. Якщо такого вказівника не існує, то повертається NIL. Якщо змінна capital має властивості, які їй були надані у попередньому прикладі, то:

```
$ (GET 'capital 'england)     $ (GET 'capital 'germany)
london                        bonn
```

3. (**REMPROP** <символ> <ключ>). Видалення зі списка властивостей <символа> властивості, яка відповідає <ключу>. Повертається старе значення властивості, якщо воно знайдено, та NIL – інакше. Нехай символ capital має три попередні властивості.

```
$ (REMPROP 'capital 'germany)       $ (REMPROP 'capital 'usa)
bonn                                washington

$ (SYMBOL-PLIST 'capital)
((england . london))
```

Функції розпізнання

Функції розпізнання — це твердження, які використовуються для розпізнання або ідентифікації об’єктів даних muLisp. Ці функції мають тільки один аргумент, а повертають булеве значення. Вони розпізнають об’єкт, який може мати довільну структуру. Ми вже розглянули деякі функції розпізнання: SYMBOLP, INTEGERP, NUMBERP, ATOM, LISTP, NULL. Розглянемо інші.

(**ZEROP** <obj>). Повертає T, якщо obj — число 0.
(**PLUSP** <obj>). Повертає T, якщо obj — додатне ціле число.
(**MINUSP** <obj>). Повертає T, якщо obj — від’ємне ціле число.
(**ODDP** <obj>). Повертає T, якщо obj — непарне ціле число.
(**EVENP** <obj>). Повертає T, якщо obj — парне ціле число.

Функція (**ASCII** <sym>) повертає ASCII-код символа <sym>. Функція (ASCII <num>) повертає символ, ASCII код якого дорівнює числу <num>. Для того, щоб визначити, чи є символ sym літерою, можна використати функцію: (< (ASCII 'a) (ASCII sym) (ASCII 'z)). Оскільки muLisp не розрізняє малі та великі літери, то (ASCII 's) = (ASCII 'S) для будь-якого символа s. Функція ISCHAR

розпізнає літери. Для знаходження ASCII кодів символів, які позначають цифри, необхідно використовувати одинарний Escape-символ.

```
$ (DEFUN ISCHAR (char)
(<= (ASCII 'a) (ASCII char) (ASCII 'z)) )
```

```
$ (ASCII 'f)
70
$ (ASCII '\9)
57
```

```
$ (ASCII 70)
F
$ (ASCII 57)
\9
```

Наступні функції дають можливість розпізнавати символи та числа.

(**ALPHA-CHAR-P** <obj>) – повертає T, якщо <obj> – літера.
(**NUMERIC-CHAR-P** <obj>) – повертає T, якщо <obj> – цифра.
(**ALPHANUMERICP** <obj>) – повертає T, якщо <obj> – літера або цифра.

```
$ (ALPHA-CHAR-P W)
T

$ (NUMERIC-CHAR-P W)
NIL

$ (ALPHANUMERICP W)
T
```

```
$ (ALPHA-CHAR-P \3)
NIL

$ (NUMERIC-CHAR-P \3)
T

$ (ALPHANUMERICP \3)
T
```

```
$ (ALPHA-CHAR-P ~)
NIL

$ (NUMERIC-CHAR-P ~)
NIL

$ (ALPHANUMERICP ~)
NIL
```

Зазначимо, що символ пропуску (' ') є літерою.

Завдання

1. Перевірити, чи складається список лише з:
   а) рівних чисел
   б) символів
   в) від'ємних дробових чисел
   г) додатних та від'ємних малих цілих, які чергуються через одне, причому їхня кількість парна
   д) від'ємних парних цілих чисел

2. За допомогою примітивних об'єктів даних створити:
   а) збалансоване бінарне дерево висоти 3, листки якого є символами.
   б) список з підсписками глибини вкладеності 3, елементами якого є числа.

3. Написати функцію, яка за вхідним списком будує необхідну структуру:
   а) '(q w e r t y) —> (q (w (e (r (t (y))))))
   б) '(q w e r t y) —> ((((((y) t) r) e) w) q)
   в) '(q w e r t y) —> ((q) (w) (e) (r) (t) (y))
   г) '(q w e r t y) —> ((q w) (e r) (t y))

4. Написати функцію, яка за списком який дано в завданні 3 в другому стовпчику, будує лінійний список.

Числові функції.

Числові функції виконують основні математичні операції над цілими та дробовими числами. Користувач може обрати для роботи точну або наближену раціональну арифметику. Для точної раціональної арифметики розмір цілих чисел, чисельників та знаменників обмежений приблизно до 25000 десяткових знаків.

Примітивними числовими функціями є ***додавання***, ***віднімання***, ***множення*** та ***ділення***. В мові програмування Ліспі вони є n-арними, тобто кількість їхніх аргументів необмежена. Синтаксис числових функцій наступний:

1. (+ <num1> <num2> ... <numN>).
2. (- <num1> <num2> ... <numN>)
3. (* <num1> <num2> ... <numN>)
4. (/ <num1> <num2> ... <numN>)

Функція додавання повертає суму своїх аргументів. Функція віднімання повертає різницю першого аргумента та суми всіх інших аргументів. Функція множення повертає добуток своїх аргументів. Функція ділення повертає частку від ділення першого аргумента та добутку інших аргументів.

```
$ (+ 2 4 6 7)          $ (- 20 3 5 6)          $ (* 2 4 6)          $ (/ 24 2 2 3)
19                     6                       48                   2
```

Функції ***збільшення*** та ***зменшення*** мають наступний синтаксичний вигляд:

1. (**ADD1** <n>). Повертає значення, яке на одиницю більше за аргумент.
2. (**SUB1** <n>). Повертає значення, яке на одиницю менше за аргумент.
3. (**INCQ** <sym><n>) Збільшує значення символа <sym> на число <n>.
4. (**DECQ** <sym><n>) Зменшує значення символа <sym> на число <n>.

Якщо функцію додавання (віднімання) одиниці запустити без аргументів, то виникне переривання по помилці: недостатня кількість аргументів. Якщо у функцію INCQ або DECQ передати один аргумент - символ, то збільшення (зменшення) значення символа відбудеться на одиницю. Окрім того, що функції INCQ та DECQ повертають результат арифметичної дії, значення символів, які передаються до них як аргументи, змінюється.

```
$ (ADD1 6)          $ (SUB1 10)
7                   9
$ (SETQ S 10)       $ (INCQ S 14)       $ (DECQ S 4)
10                  24                  30
```

Якщо у вашій реалізації Lisp нема цихфункцій, то визначте їх за допомогою DEFUN.

Функції **MIN** та **MAX** повертають символ з відповідно ***мінімальним*** (***максимальним***) значенням.

1. (MIN <n1> <n2> ... <nM>).
2. (MAX <n1> <n3> ... <nM>).

```
$ (MIN 12 3 45 67)      $ (MAX 1 2 5 3)
3                       5
```

Числові вирази в Ліспі записуються в префіксній формі. Вираз 3*5+5*7 для обчислення треба подати у вигляді (+ (* 3 5) (* 5 7)), вираз (3+6)*7 — у вигляді (* (+ 3 6) 7).

Функції **порівняння** менше та більше мають n аргументів.

1. (< <n1> <n2> ... <nM>) Повертає істину, якщо <n1> < <n2> < ... < <nM>.
2. < ><n1> <n2> ... <nM>) Повертає істину, якщо <n1> > <n2> > ... > <nM>.
3. (/= <n1> <n2> ... <nM>) Повертає істину, якщо існують хоча б два числа, які не дорівнюють одне одному.

До функцій ***порівняння*** також відносяться <= , = та >=.

```
$ (< 2 4 6)             $ (>= 5 3 3 2)          $ ( /= 4 4 5)
T                       T                       T

$ (< 6 6 8 15)   $ (<= 6 6 8 15)    $ ( /= 4 4 4)
NIL              T                  NIL
```

Функції округлення

(**TRUNCATE** m n), (**ROUND** m n), (**CEILING** m n) (**FLOOR** m n)

Ці функції використовуються для округлення дробових чисел до цілих. TRUNCATE виконує округлення до ближчого цілого у напрямку нуля. ROUND виконує округлення до ближчого цілого по значенню до m/n. CEILING виконує округлення до ближнього цілого по верхній межі, FLOOR – по нижній межі. Виклик будь-якої функції з двома аргументами (<f> m n) еквівалентний виклику функції з одним аргументом: (<f> (/ n m)), де f – будь-яка з наведених чотирьох функцій.

```
$ (TRUNCATE 6/4)       $ (TRUNCATE -6/4)        $ (CEILING 9 4)          $ (CEILING -9 4)
1                      -1                       3                        -2

$ (FLOOR 6 4)          $ (FLOOR -6 4)           $ (FLOOR 6/4)            $ (FLOOR -6/4)
1                      -2                       1                        -2
```

Функції остачі

(**REM** m n), (**MOD** m n), (**DIVIDE** m n)

Примітивна функція REM повертає остачу від ділення числа m на n. Функція MOD працює як REM, але повертає модуль остачі. Якщо (TRUNCATE m n) повертає q, а (REM m n) повертає r, то m=q*n+r. Функція (DIVIDE m n) повертає конс, CAR якого дорівнює частці, а CDR — остачі від ділення m на n.

```
$ (REM 6 4)            $ (DIVIDE 7 2)           $ (REM -6 4)       $ (MOD 6 4)
2                      (3 . 1)            -     2                  2
```

Знак числа

(**SIGNUM** n)
Повертає значення -1, 0 або 1 якщо n відповідно від’ємне, 0, або додатнє.

Модуль числа

(**ABS** n) – Модуль числа n.

Чисельник та знаменник

(**NUMERATOR** n), (**DENOMINATOR** n) – чисельник та знаменник раціонального числа n.

```
$ (signum -5/3)        $ (abs -5/3)       $ (numerator 10/8)       $ (denominator 10/8 )
-1                     5/3                5                        4
```

Побітові логічні функції

(**LOGAND** <n1><n2>...<nM>)
(**LOGIOR** <n1><n2>...<nM>),
(**LOGXOR** <n1><n2>...<nM>)
(**LOGNOT** n).

```
$ (LOGAND 5 7 3)       $ (LOGIOR 4 2 1)         $ (LOGXOR 5 2 3)         $ (LOGNOT 6)
1                      7                        4                        -7
```

Булеві функції

(**NOT** <об'єкт>)
(**AND** <форма1> <форма2> ... <формаN>)
(**OR** <форма1> <форма2> ... <формаN).

```
$ (AND (EQL 'as 'as) (< 2 4))       $ (OR NIL (< 4 56))       $ (NOT (EQL 'd 'g))
T                                   T                         T
```

Зсув

(**SHIFT** m n) — зсув числа m на n бітів.

```
$ (SHIFT 3 1)          $ (SHIFT 3 -1)          $ (GCD 24 66 600)         $ (LCM 24 66 600)
6                      1                       6                         6600
```

НСД, НСК Ці функції знаходять відповідно найбільший спільний дільник M чисел та найменше спільне кратне.

(**GCD** n1 n2 ... nM)
(**LCM** n1 n2 ... nM).

Є великий набір ірраціональних та трансцендентних функцій. Аргументи тригонометричних функцій задаються в радіанах.

1. (**EXP** x) експонента e^x
2. (**EXPT** x y) степінь x^y
3. (**LOG** x y) логарифм $\log_y x$. Якщо y не задано, основа вважається рівною e.
4. (**LN** x) натуральний логарифм
5. (**SQRT** x) квадратний корінь
6. (**ISQRT** x) ціла частина з квадратного кореня
7. (**SIN** x) та (**ASIN** x) сінус та арксінус
8. (**COS** x) та (**ACOS** x) косинус та арккосинус
9. (**TAN** x) та (**ATAN** x) тангенс та арктангенс
10. (**RANDOM** n) генерується натуральне число, менше за n.

Контрольні конструкції

Lisp використовує неявну форму PROGN для обчислення форм, які складають тіло функції. Спеціальні форми забезпечують контроль за обчисленням форм в процесі виконання програм. Розглянемо деякі контрольні інструкції.

1. ***QUOTE <об'єкт>*** повертає об'єкт <obj> без його обчислення. QUOTE може використовуватися для запобігання обчислення значень констант, які передаються як аргумент функції, що обчислюється.

```
$ (SETQ a 125)
$ a              $ (QUOTE a)          $ (CAR (CONS 4 7))          $ (CAR '(CONS 4 7))
125              a                    4                           CONS
```

2. ***LOOP WHILE <предикат> DO <форма1> <форма2> ... <формаN>*** Повторно обчислює форми у послідовному порядку доти, поки предикат не рівнй NIL.

Розглянемо функцію LENGTH обчислення довжини списку. В першому стовпчику запропоновано рекурсивний, в лівому — нерекурсивний варіант програми.

```
(DEFUN LENGTH-R (lst)
 (COND
  ((NULL lst) 0)
  ((+ 1 (LENGTH-R (CDR lst))))
 )
)
```

```
(DEFUN LENGTH-IT (lst)
 (SETQ K 0)
 (LOOP WHILE (IDENTITY lst)
  DO (SETQ lst (CDR lst) K (+ 1 K))
 ) K
)
```

3. ***IF <предикат> [THEN] <форма1> [ELSE] <форма2>*** Якщо значення предиката не дорівнює NIL, то видається [THEN] форма, інакше видається [ELSE] форма.

```
$ (IF (EQL 'r 'r) (CAR '(q w e r t y)) (CDR '(q w e r t y)))  –  q
$ (IF (EQL 'r 'w) (CAR '(q w e r t y)) (CDR '(q w e r t y)))  –  (w e r t y)
```

4. ***IDENTITY <об'єкт>*** Повертає об'єкт без жодних змін. Ця функція застосовується для використання змінних як предикатів в умовних виразах.

5. ***PROGN <форма1> <форма2> ... <формаN>*** Послідовно обчислює форми та повертає результат обчислення формиN.

6. ***PROG1 <форма1> <форма2> ... <формаN>*** Послідовно обчислює форми та повертає результат обчислення форми1. Функцію використовують для того, щоб вводити допоміжні змінні для збереження результатів в процесі обчислення інших виразів.

```
$ (SETQ a '(q w e r t y))                    $ a
$ (PROG1 (CAR a) (SETQ a (CDR a)))           (w e r t y)
q
```

7. ***COND <cond1> <cond2> ... <condN>*** Обчислює CAR кожної COND форми доти, доки не зустрінеться деяке значення, відмінне від NIL, або доки всі предикати не будуть обчислені. В першому випадку COND обчислює CDR елемент cons - форми з предикатом, який не дорівнює NIL, як тіло функції, використовуючи неявну функцію PROGN. Якщо CDR - елемент COND форми, яка не дорівнює NIL, є порожнім, то повертається значення предиката. Якщо обчислені всі предикати та всі вони повернули NIL, то COND повертає NIL.

8. ***COMMENT <коментар>*** Ігнорує свої аргументи та повертає NIL. Визначає засіб включення коментарів безпосередньо у визначені функції.

9. ***RETURN <об'єкт>*** Зупиняє виконання функції, яка містить RETURN, звільняє стек та повертає об'єкт в ролі свого значення.

10. ***RESTART*** Закриває всі відкриті файли, відмовляється від поточного середовища та ініціює нову систему muLisp. Всі зв'язки між змінними, функції користувача та значення властивостей поточного середовища знищуються.

11. ***SYSTEM*** Закриває всі відкриті файли, завершує виконання muLisp та повертає керування операційній системі.

12. ***EXECUTE <програма> <командний рядок>*** Зупиняється робота системи muLisp, передається керування програмі з командним рядком. EXECUTE повертає код виходу з програми або NIL, якщо <програма> не знайдена.

Обчислення рекурсивних функцій

1. ***Факторіалом*** числа n називається число (позначається n!), яке рекурсивно визначається так: 0! = 1; N! = N*(N-1)! якщо N>0.

```
$ (DEFUN FACTORIAL (n)                    $ (FACTORIAL 10)
  (COND ((ZEROP n) 1)                     3628800
   ((* n (FACTORIAL (- n 1))))
 ))
```

Якщо в рекурсивній програмі аргументом буде велике число, то може виникнути переповнення стеку. Використовуючи команду циклу LOOP можна уникнути рекурсивного виклику. Наступна функція буде більш ефективною:

```
$ (DEFUN FACTORIAL1 (n rslt)              $ (FACTORIAL1 10 F)
   (SETQ rslt 1)                          3628800
   (LOOP WHILE (> n 1)
    DO (SETQ rslt (* n rslt))             $ (FACTORIAL 0 a)
       (SETQ n (- n 1))  )  )             1
```

2. Послідовність чисел, кожен елемент якої дорівнює сумі двох попередніх, а перші два елементи дорівнюють 1, називається ***послідовністю Фібоначчі***. N-те число послідовності Фібоначчі F(N) може бути знайдене за рекурсивною формулою: F(0)=1, F(1)=1, F(N) = F(N-1) + F(N-2).

```
$ (DEFUN FIBON (n)                        $ (FIBON 20)
  (COND                                   10946
   ((<= n 1) 1)
   ((+ (FIBON (- n 1)) (FIBON (- n 2))))
  ))
```

Визначена таким чином функція не є ефективною, оскільки для обчислення N-ого числа Фібоначчі необхідно обчислити (N-2) число Фібоначчі двічі, (N-3) — тричі і так далі. Визначимо функцію FIB з трьома аргументами, останні два з яких при виклику функції повинні дорівнювати відповідно F(0) та 0).

```
$ (DEFUN FIB (n f1 f2)                    $ (FIB 20 1 0)
   (COND ((ZEROP n) f1)                   10946
    ((FIB (- n 1) (+ f1 f2) f1))  ))
```

Так само ефективним буде відповідне ітеративне визначення.

Завдання

1. Визначити функції MIN, MAX, INCR, DECR для списків. Функція INCR (DECR) повертає істину, якщо значення аргументів знаходяться в зростаючому (спадному) порядку.

2. Написати функцію, яка за списком з підсписками знаходить:
   а) суму елементів
   б) кількість елементів
   в) кількість підсписків
   г) лінеризує список

3. Написати функцію:
   а) (DIVIS x y) — повертає частку та остачу від ділення x на y. Повернути результат у вигляді конса. Не використовувати функцій ділення та остачі.
   б) (POW x y) — x в степені y. Запропонувати алгоритми з часовою оцінкою O(y) та O(log y).
   в) (SLIST n) — розклад числа n на прості множники. Як результат виконання функції повернути список простих чисел, добуток яких дорівнює n.
   г) (PERLEN n) — за натуральним числом n повернути довжину періоду дробу 1/n.
   д) (SUMFACT n) – сума 1/0! + 1/1! + ... + 1/n!.

4. Дано дві цілі змінні a та b. Не вводячи допоміжної змінної, поміняти місцями значення a та b (нове значення a дорівнює старому значенню b та навпаки).

5. Написати функцію:

а) (BINARY n) – кількість знаків у двійковому представленні числа n.

б) НСД та НСК двох чисел за алгоритмом Евкліда.

НСД(a, b) = НСД(a - b, b), якщо a>b,
НСД(a, b - a), якщо a<b,
a, якщо a = b.

в) НСД двох чисел за модифікованим алгоритмом Евкліда.

НСД(a, b) = НСД(a mod b, b), якщо a>b,
НСД(a, b mod a), якщо a<b,
a, якщо b = 0.
b, якщо a = 0.

г) (INVERTBIT a n) – обернути n-ий біт числа a.

д) (EQ2 a b c) – розв’язати квадратне рівняння.

е) (SQTR a b c) – знайти площу трикутника за трьома сторонами (використати формулу Герона).