Мова програмування Ліспу

За однією з класифікацій мови програмування (МП) ділять на ***імперативні*** (процедурні) та ***декларативні*** мови. Більшість мов, що сьогодні використовуються, відносять до імперативних мов. До класу декларативних мов відносяться ***функціональні*** або апплікативні – Ліспу, Лого та ***логічні*** мови, відомим представником яких є Пролог. На практиці МП не є чисто процедурними, функціональними чи логічними. Процедурною мовою можна написати функціональну програму і навпаки.

Функціональна програма складається з сукупності визначених функцій. Функції, в свою чергу, можуть викликати інші функції. Обчислення починається з виклику деякої функції. Чисте функціональне програмування не має присвоєнь та засобів передачі керування. Повторні обчислення здійснюються за допомогою рекурсії, яка є основним засобом функціонального програмування.

Робота з Ліспом нагадує роботу з кишеньковим калькулятором: користувач вводить вираз (він обов'язково повинен закінчуватися символом <RETURN> та мати збалансовану кількість дужок), який читає машина, потім обчислює (інтерпретує), та видає результат. Цей процес ***введення-читання-обчислення-видачі*** результату буде відбуватися в циклі доти, доки користувач не введе команду, яка завершує роботу з Lisp і передає керування операційній системі.

Об'єкти Ліспу

Об'єкти можуть бути двох типів: ***прості*** та ***складені***. Прості об'єкти називаються ***атомами***. До атомів відносяться ***символи*** та ***числа***. Символ не може починатися з цифри. Lisp не розрізняє маленькі та великі літери, а перетворює всі введені літери на великі. Атом є неподільним, тобто його не можна розбити на компоненти. Атом має ***ім'я***. Іменами атомів є рядки символів. DOG, CAT, qw1232df є типовими іменами атомів. Символи T та NIL мають в Ліспі спеціальне призначення: вони позначають відповідно логічні значення істини та хиби. Ці символи завжди повинні мати одне фіксоване значення. Їх не можна використовувати в якості імен інших об'єктів Ліспу. Числа та логічні значення T та NIL є ***константами***, всі інші символи – ***змінними***.

Складними об'єктами даних є *списки*. Список містить нуль (тоді говорять про порожній список) або більше об'єктів, кожний з яких може бути як простим, так і складеним. (FACE, LOOK, NOSE) є списком, який складається з трьох атомів. Порожній список позначається NIL = (), який є атомом. Список називається ***лінійним***, якщо його елементи є атомами. Інакше говорять про списки з підсписками, наприклад: (7 (8 9) TR).

Для того щоб введений вираз не обчислювався, перед ним ставлять апостроф (’). Якщо вираз ввудено без апострофа, то повертається його значення. При запуску програми Lisp значенням кожного атома вважається він сам. Значенням числа завжди є саме число, тому перед числами апостроф не ставлять. Тобто, після старту системи при вводі Q результатом буде його значення – Q, а при вводі ’Q — буде завжди Q. Апостроф перед виразом – це скорочення форми QUOTE, яка записується в наступній формі: ’вираз == (QUOTE вираз). QUOTE можна використовувати як спеціальну функцію з одним аргументом, яка нічого з ним не робить, а повертає як результат сам аргумент.

Списки задають переліком елементів, взятих в дужки, перед якими ставлять апостроф. Наприклад: ’(ice, hen) або ’((one 1) (two 2) (three 3)).

Примітивні функції Ліспу

Lisp має п'ять ***примітивних функцій***. Виклик функції має наступний формат:
(name arg1 arg2 ...), де *name* — ім'я функції, *arg1,arg2*,... — її аргументи.

1. (**CAR** <list>) голова списку.
2. (**CDR** <list>) хвіст списку.

3. (**CONS** <object> <list>) об’єднання (конкатенація) об’єкта зі списком.
4. (**EQL** <atom1> <atom2>) порівняння двох атомів.
5. (**ATOM** <object>) перевірка чи є <object> атомом.

CAR та CDR називаються ***селекторними*** функціями, оскільки вони дають можливість вибирати частину об’єкта. Результатом функції (CAR list) завжди є перший елемент списку list, якщо він непорожній і NIL в іншому випадку. Результатом функції (CDR list) є список list без першого елемента, якщо list містить більш одного елемента і NIL в іншому випадку. (У прикладах нижче знаком $ позначено запрошення системи.)

```
$ (CAR '(q w e r t y)) $ (CDR '(q w e r t y))    $ (CAR '((one 1) (two 2)))
q                      (w e r t y)              (one 1)

$ (CAR '())      $ (CDR '(tree))    $ (CDR '((q w))    $ (CDR '())
NIL              NIL                NIL                NIL
```

За допомогою функцій CAR, CDR можна знаходити за даним списком будь-який його підсписок або атом. Дозволяється використовувати функції, які є комбінаціями CAR та CDR. Імена таких функцій починаються на C і закінчуються на R, а між ними знаходиться послідовність літер A та D (але не більше 4–5 літер, залежно від реалізації), яка вказує шлях обчислення.

```
$ (CAR (CDR (CDR '(q w e r t y))))        $ (CAR (CDR (CDR '((q 1) (w 2) (e 3)))))
$ (CADDR '(q w e r t y))                  $ (CADDR '((q 1) (w 2) (e 3)))
e                                         (e 3)

$ (CDR (CDR '((q 1) (w 2) (e 3))))       $ (CAR (CAR '((q w))))
$ (CDDR '((q 1) (w 2) (e 3)))            $ (CAAR '((q w)))
((e 3))                                   q
```

Функцію ***конструктора*** CONS використовують для додання об’єкту до заданого списку. Об’єкт, який додають, стає головою списку.

```
$ (CONS '(q w) '(r (t y)))          $ (CONS apple '(cherry strawberry))
((q w) r (t y))                     (apple cherry strawberry)

$ (CONS '(q w) '(r t y))            $ (CONS 5 nil)
((q w) r t y)                       (5)
```

Якщо результатом виразу (CONS <object> <list>) буде <new>, то результатом (CAR <new>) буде <object>, а результатом (CDR <new>) буде <list>.

```
$ (CAR (CONS '(q w) '(r (t y))))    $ (CAR (CONS apple NIL))
(q w)                               apple
```

Функцією ***порівняння*** є EQL. Вона порівнює значення першого та другого аргумента, які обов’язково повинні бути атомами, та повертає значення істина (T) або хиба (NIL).

```
$ (EQL 'qw 'qw)        $ (EQL (CAR '(q w)) q)          $ (EQL (CAR '(q,w) NIL)
T                      T                               F
```

При написанні програм на Ліспі часто виникає запитання: чи є даний об’єкт атомом? Це питання вирішує предикат ATOM. Він повертає T, якщо об’єкт є атомом і NIL в іншому випадку. Порожній список NIL є атомом.

```
$ (ATOM qwerty)        $ (ATOM '(q w e))        $ (ATOM '())       $ (ATOM 3)
T                      NIL                      T                  T
```

Функції призначення

Функції ***призначення*** застосовують для надання значень програмним змінним.

1. (**SET** <symbol><object>) заміна символа об'єктом
2. (**SETQ** <sym1><form1> <sym2><form2> ...) спеціальна форма функції SET
3. (**PSETQ** <sym1><form1> <sym2><form2> ...) спеціальна форма функції SET
4. (**POP** <symbol>) повертає вершину стека (списку)
5. (**PUSH** <symbol><form>) проштовхує символ <symbol> в стек (список) <form>.

Операція ***заміни*** значення символа здійснюється за допомогою функції SET. Вона присвоює символу <symbol> значення <object>, або зв'язує <symbol> з <object> . Для скорочення замість SET ' пишуть SETQ (SET Quote). Як результат функція присвоєння повертає другий аргумент.

```
$ (SET  'fox '(a s d))        $ (SETQ vowels '(a e i o u)))
$ (SETQ fox '(a s d))         $ (SETQ vowels (CONS 'y vowels))
(a s d)                       (y a e i o u)
```

Функція SETQ дозволяє здійснювати заміну значень декільком символам в одній команді: (SETQ a 1 b 2 c 3). При цьому зміни виконуються послідовно зліва направо. Після цього значенням символу a стане 1, b – 2, c – 3.

Функція PSETQ ідентична до функції SETQ за винятком того, що всі форми оцінюються до того, як будуть здійснені будь-які заміни. Проілюструємо це на прикладі. Значення символа Sym позначатимемо через Val(Sym).

```
$ (SETQ w 1 e 2) Val(w)=1, Val(e)=2        $ (SETQ w 1 e 2)   Val(w)=1, Val(e)=2
$ (SETQ w e e w) Val(w)=2, Val(e)=2        $ (PSETQ w e e w)  Val(w)=2, Val(e)=1
```

При виконанні операції заміни необхідно розрізняти символ та значення. При старті системи Lisp значенням кожного символа є він сам. Якщо ми введемо DOG, то і результатом буде DOG. Присвоїмо символові DOG значення CAT: (SET 'DOG 'CAT). Результатом виразу (SET DOG 'HEN) буде HEN, але значення HEN ми присвоювали не символу DOG, а значенню символа DOG, тобто символу CAT. Значення символа DOG залишилося без зміни. Розглянемо результат наступних дій:

```
(SET 'car 'road)       Val(car) = road    Val(road) = road
(SET car flower)       Val(car) = road    Val(road) = flower Val(flower) = flower
(SET 'car car)         Val(car) = road    Val(road) = flower Val(flower) = flower
(SET road car)         Val(car) = road    Val(road) = flower Val(flower) = road
(SET 'road 4)          Val(car) = road    Val(road) = 4            Val(flower) = road
(SET road 'hen)        помилка, 4 не є символом і не може приймати інші значення
```

POP повертає голову списка (вершину стека) і замінює значення <symbol> на його хвіст. PUSH кладе <symbol> в стек та змінює його значення на збільшений стек.

```
$ (SETQ a '(q w e r t))Val(a) = (q w e r t)
$ (POP a)              Val(a) = (w e r t)
$ (PUSH 'n a)          Val(a) = (n w e r t)
```

Завдання 1

1. Побудувати список, який задовільняє наступним умовам:
   а) містить два підсписки, перший з яких має три атоми, а другий – чотири атоми;
   б) містить три атоми, але його хвіст дорівнює NIL;
   в) містить три складені об’єкти, і лише його другий елемент є атомом;
   г) голова списку містить три атоми, а кількість атомів в усьому списку дорівнює 3.
   д) містить тільки порожній список, а голова списку не є атомом.
   е) голова та хвіст є списками з підсписками.

2. Що буде в результаті обчислення наступних виразів:

```
а) (CONS NIL NIL)                                  г) (ATOM (CDR '(q NIL)))
б) (CONS (CAR '((q w))) (CDR '((q (w e)))))        д) (EQL NIL 'NIL)
в) (EQL (CDR '(q)) NIL)                            е) (PUSH nil nil) (EQL (ATOM '(q w)) nil)
```

3. Скласти вираз, який би за вхідними даними побудував би заданий результат.

```
а) дано: (A, B, C), (X, Y, Z).                     побудувати: (A, Y, Z).
б) дано: ((one 1) (two 2 3) (three 4 5 6))         побудувати: 5.
в) дано: ((q w (r) t) y)                           побудувати: NIL
г) дано: ((q (w (e) r) t) y)                       побудувати: ((q) w (e) r)
д) дано: (q (w e))                                 побудувати: w, e
е) дано: (q w)                                     побудувати: (((q w)))
```

4. Скласти вираз, який надає значення вхідним даним та вираз, який будує заданий результат, використовуючи лише вихідні символи.

```
а) дано: one=1, two=2, three=3              зробити: one=2, two=3, three=1.
б) дано: Val(house)=sky, Val(sky)=house     зробити: Val(sky)=sky, Val(house)=house
в) дано: Val(lst)=(q)                       зробити: Val(lst)=(((q) q) q)
г) дано: Val(q)=w, Val(w)=s                 зробити: Val(q)=(s s)
```

5. Не використовуючи селекторні функції:

```
а) дано: Val(a) = (q w e r t y)             зробити: Val(a) = q
б) дано: Val(a) = (q w e r t y)             зробити: Val(a) = (w)
```

6. Вказати значення всіх змінних після виконання наступних дій:

```
(SET one 'two)
(SETQ two 'one)
(SET three two four 'one two three)
(PSETQ four one three 'four two three one four)
```

Визначення функцій в Ліспі

Поряд з примітивними функціями можуть існувати функції, визначені користувачем. Функція викликається набором аргументів і повертає єдине значення. Визначення функції в Ліспі має такий вигляд:

```
(DEFUN name (arg1 arg2 ...)
  task1
  task 2
. . . . .  )
```

де name – ім'я функції, arg1, arg2, ... – аргументи (параметри). Тіло функції містить послідовність задач. Ключове слово DEFUN виникло з DEfine FUNction.

```
$ (DEFUN FIRST (lst)          $ (FIRST '(q w e r t y))
   (CAR lst) )                   q

$ (DEFUN THIRD (lst)          $ (THIRD '(q w e r t y))
   (CADDR lst) )                 e
```

Визначимо функцію **NULL**, яка розпізнає порожній список. Вона повертає істину, якщо її аргументом є порожній список і хибу в іншому випадку. (У середовищі *LispWorks* така функція вже визначена – одна з стандартних.)

```
$ (DEFUN NULL (obj)           $ (NULL '(q w e r))             $ (NULL (CDR '(r)))
   (EQL obj NIL) )            NIL                             T
```

Нам вже відомі три функції розпізнання: EQL, ATOM, NULL. Функції, які застосовуються для перевірки певних умов та можуть повертати лише два значення – істини чи хиби, називаються ***предикатами***.

Тіло функції складається з послідовності завдань. Завдання можуть бути двох типів: прості та умовними. Будь-яке завдання береться в круглі дужки і може розглядатися як список виразів, які треба проінтерпретувати.

Якщо завдання є атомом або його перший елемент є атомом, то таке завдання називається ***простим***. Наприклад, (CONS 'NR 'LST).

Якщо перший елемент списка, який описує завдання не є атомом, то таке завдання називається ***умовним***. Наприклад, ((ATOM lst) (CONS expr lst)).

В умовному завданні перший елемент списку обов'язково є предикатом. Якщо значення предикату NIL, то значення завдання стає рівним NIL і Лісп переходить до виконання наступного завдання. Якщо предикат повертає не NIL, відбувається виконання хвосту списку завдання, а інші завдання ігноруються. Якщо предикат повертає T, а хвіст завдання порожній, то результатом всієї функції буде T.

Напишемо предикат, який розпізнає списки. Якщо аргументом є список, то предикат повертає істину, інакше – хибу. Функцію **ISLIST** можна проінтерпретувати наступним чином: "Якщо obj є атомом, то повернути NIL, інакше повернути T".

```
$ (DEFUN ISLIST (obj)              $ (DEFUN ISLIST (obj)
   ((ATOM obj) NIL)                    ((NULL obj))
   T )                                 ((ATOM obj) NIL)
                                       T )
```

У стовпці праворуч написано предикат ISLIST, який розпізнає додатково порожній список (повертає істину). Перше завдання є умовним, хвіст якого порожній. Його можна проінтерпретувати так: перевірити об'єкт obj на порожній список, і якщо він є таким, передати як результат функції істину. Немає потреби писати: ((NULL obj) T), оскільки це те ж саме, що і ((NULL obj)). Останнім

завданням цих предикатів є атом T. Це означає, що якщо жодне з умовних завдань не виконане (лише за цієї умови керування програмою дійде до останнього завдання), то повернути T.

У середовищі *LispWorks* умовні завдання потрібно оголошувати аргументами вбудованої функції COND. Визначені вище функції матимуть вигляд:

```
$ (DEFUN ISLIST (obj)
   (COND
    ((ATOM obj) NIL)
    (T))
  )
```

```
$ (DEFUN ISLIST (obj)
   (COND
    ((NULL obj))
    ((ATOM obj) NIL)
    (T) ))
```

Для другого визначення функції ISLIST маємо:

```
$ (LISTP 'tree)
NIL
```

```
$ (LISTP '())
T
```

```
$ (LISTP '(q w e r t y))
T
```

Для розпізнавання списків у середовищі *LispWorks* можна використовувати вбудовану функцію LISTP.

Вбулована функція (**LIST** x1 ... xn) утворює та видає список, елементами якого є x1, ..., xn. Якщо аргументи не задані, результатом буде NIL.

```
$ (LIST 'a 'b 'c 'd)
(a b c d)
```

```
$ (LIST 'a '(b c) 'd)
(a (b c) d)
```

```
$ (LIST)
NIL
```

Напишемо функцію **BELONGS**, яка має два аргументи: *nam* – символ та *lst* – список і яка повинна перевірити, чи належить символ списку. Інтуїтивно необхідно порівняти символ з першим елементом списку, потім з другим елементом і так далі. Проблема в такому розв'язку виникає в тому, що ми не знаємо наперед довжини списку. А якщо ми і знаємо цю довжину, і якщо вона велика, то тіло функції буде дуже великим. Така функція буде мати приблизно такий вигляд (перший стовпчик):

```
$ (DEFUN BELONGS (nam lst)
    ((EQL nam (FIRST lst)))
    ((EQL nam (SECOND lst)))
    ((EQL nam (THIRD lst)))
    ((EQL nam (THIRD (CAR lst))))
  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .  .
```

```
$ (DEFUUN BELONGS (nam lst)
   (COND
    ((NULL lst) NIL)
    ((EQL nam (CAR lst)) T)
    ((BELONGS nam (CDR lst))) ))
```

Змінимо наш підхід до побудови функції. В другому стовпчику побудовано функцію BELONGS, в основі якої лежить *рекурсивний* підхід, який базується на наступних фактах:

1. Якщо список порожній (не має елементів), то *nam* не належить списку.
2. Якщо *nam* дорівнює голові списку, то *nam* належить списку.
3. Якщо *nam* не дорівнює голові списку, то *nam* може належити списку тоді і тільки тоді, коли *nam* належить хвосту списку.

Розглянемо дві рекурсивні функції: **REMBER** (REMove memBER), яка має два аргументи — атом obj та список lst і яка видаляє перше зустрічання атома obj в списку lst. REMBER-ALL яка видаляє всі атоми obj в списку lst.

```
$ (DEFUN REMBER  (obj lst)
   ((NULL lst) NIL)
   ((EQL obj (CAR lst)) (CDR lst))
   (CONS (CAR lst)
    (REMBER obj (CDR lst))) )
```

```
(DEFUN REMBER-ALL (obj lst)
 ((NULL lst) NIL)
 ((EQL obj (CAR lst))
 (REMBER-ALL obj (CDR lst))
 (CONS (CAR lst)
       (REMBER-ALL obj (CDR lst))) )
```

Результат роботи цих функцій проілюструємо на прикладах:

```
$ (REMBER 'a '(q a w e r t  a y))   $ (REMBER-ALL 'a '(q a w e r t a y))
(q w e r t a y)                     (q w e r t y)
```

Примітивна функція EQL використовується для порівняння атомів. Часто виникає потреба порівнювати списки. Напишемо функцію **EQLIST**, яка порівнює списки. Її побудуємо на основі наступних фактів:

1. Якщо перший список порожній, то, якщо і другий список порожній, повернути T, інакше повернути NIL (або просто повернути (NULL другого списку)).
2. Якщо другий список порожній, повернути NIL.
3. Якщо голова першого списку не дорівнює голові другого списку, повернути NIL.
4. Перевірити рівність хвостів першого та другого списків.

```
$ (DEFUN EQLIST (lst1 lst2)
   ((NULL lst1) (NULL lst2))
   ((NULL lst2) NIL)
   ((NOT (EQL (CAR lst1) (CAR lst2))) NIL)
   (EQLIST (CDR lst1) (CDR lst2)) )
```

```
$ (DEFUN NOT (obj)
   (EQL obj NIL) )
```

Функція NOT повертає NIL, якщо список не порожній і T інакше.

Розглянемо задачу об'єднання списків. Роботу функції **APPEND**, аргументами якої є два списки lst1 та lst2, можна описати наступним чином:

1. Якщо lst1 порожній, повернути lst2.
2. З'єднати голову першого списку зі списком, який отримано в результаті об'єднання хвоста першого списку з другим списком.

```
$ (DEFUN APPEND (lst1 lst2)
   ((NULL lst1) lst2)
   (CONS (CAR lst1) (APPEND (CDR lst1) lst2)) )
```

Функція (**REVERSE** lst1) обертає список lst1. Якщо вихідний список порожній, то і результатом буде порожній список. Інакше необхідно об'єднати обернений хвіст вихідного списку з його першим елементом. Оскільки на вхід другого аргумента функції APPEND повинен подаватися список, необхідно з першого елемента списку зробити список, який складається лише з нього. Це виконує команда (CONS (CAR lst) NIL).

```
$ (DEFUN REVERSE (lst)
   ((NULL lst) NIL)
   (APPEND (REVERSE (CDR lst)) (CONS (CAR lst) NIL)) )
```

Напишемо функцію REVERSE без використання функції APPEND. Для цього побудуємо функцію REVERSE з двома аргументами на принципі обробки стеку. Вихідний список — стек символів. Якщо він порожній, то і результуючий стек буде порожнім. Інакше взяти символ з вершини стеку і покласти його на другий стек. Другий стек при виклику повинен бути NIL: (REVER <list> NIL).

```
$ (DEFUN REVER (lst1 lst2)
((NULL lst1) lst2)
(REVER (CDR lst1) (CONS (CAR lst1) lst2)) )
```

```
$ (REVER '(q w e) NIL)
(e w q)
```

Завдання 2

1. Написати функцію, яка знаходить:
   а) третій елемент четвертого підсписку
   б) перший елемент другого підсписку
   в) перший атом списку з підсписками
   г) останній атом лінійного списку
   д) останній атом списку з підсписками
   е)перевіряє належність елемента до списку

2. Написати функцію REVERSE, не використовуючи функцій селектора та конструктора. Вказівка: використайте функції PUSH та POP.

3. Написати функцію, яка:
   а) з вихідного списку робить множину
   б) об’єднує дві множини
   в) знаходить різницю двох множин
   г) знаходить перетин двох множин