МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МІЖНАРОДНИЙ ЕКОНОМІКО-ГУМАНІТАРНИЙ УНІВЕРСИТЕТ
ІМЕНІ АКАДЕМІКА СТЕПАНА ДЕМ'ЯНЧУКА

Р.М.ЛІТНАРОВИЧ

ТЕСТОВІ ЗАВДАННЯ
по дисципліні:
«Аплікативні системи»
для магістрантів факультету Кібернетики

Рівне, 2011

УДК 378.147
Літнарович Р.М. Тестові завдання по дисципліні «Аплікативні системи» для магістрантів факультету Кібернетики. МЕГУ, Рівне, 2011,- 30 с.
Litnarovich R.M. To the test of task on discipline « Systems of Aplikatiation » for Master's degree preparation of faculty of Cybernetics. IEGU, Rivne, 2010,- 30 p.

Приведені матеріали тестових завдань по дисципліні «Аплікативні системи» для магістрантів факультету Кібернетики.
Ключові слова: аплікативні системи, тести, критерії, програмування

Приведены материалы тестовых заданий по дисциплине «Аппликативные системы» для магистрантов факультета Кибернетики.
Ключевые слова: аппликативные системы, тесты, критерии, программирование

Materials of test tasks on discipline « Systems of Aplikatiation » for Master's degree preparation of faculty of Cybernetics
Keywords : systems of aplikatiation, tests, criteria, programming

Відповідальний за випуск:
Й.В.Джунь, доктор фізико-математичних наук, професор

Архів електронних ресурсів колекції Університети партнерів: http://essuir.sumdu.edu.ua/handle/123456789/

1. Чи За однією з класифікацій мови програмування діляться на ***процедурні***, які також називаються ***операторними*** або ***імперативними***, та ***декларативні*** мови: (- Ні), (= Так), (- Частково).

2. Чи Більшість мов, які сьогодні використовуються – Бейсік, Фортран, Паскаль, Сі, відносяться до **процедурних мов**.: (- Ні), (= Так), (- Частково).

3. Чи До класу декларативних мов відносяться ***функціональні*** або аплікативні – Лісп, Лого, та ***логічні*** мови, відомим представником якого є Пролог: (- Ні), (= Так), (- Частково).

4. Чи На практиці мови програмування не є чисто **процедурними, функціональними чи логічними**. На процедурній мові можна написати функціональну програму і навпаки.: (- Ні), (= Так), (- Частково).

5. Чи ***Процедурна програма*** складається з послідовності операторів та виразів, які керують її виконанням. Типовими операторами є оператори присвоєння, ввода-виводу, керування та циклу: (- Ні), (= Так), (- Частково).

6. Чи ***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. : (- Ні), (= Так), (- Частково).

7. Чи можна в якості інтерпретатора мови **Ліспи** використовувати **muLisp**.: (- Ні), (= Так), (- Частково).

8. Чи **muLisp** працює на комп'ютері з операційною системою MS-DOS або PC-DOS: (- Ні), (= Так), (- Частково).

9. Чи Програма **mulisp.com** є інтерпретатором мови програмування **muLisp**, яка має потужні функціональні засоби для обробки структур даних, створених користувачем.): (- Ні), (= Так), (- Частково).

10. Чи **muLisp** є символьною мовою програмування, яка призначена для обробки списків. Слово **Lisp** виникло з перших літер фрази ***Lis***t ***P***rocessing.

: (- Ні), (= Так), (- Частково).

11. Чи 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: (- Ні), (= Так), (- Частково).

12. Чи процес ***введення-читання-обчислення-видачі*** результату буде відбуватися в циклі доти, доки користувач не введе команду (SYSTEM), яка завершує роботу з muLisp і передає керування операційній системі.
: (- Ні), (= Так), (- Частково).

13. Чи Більш ніж 260 функцій мови **Ліспи** визначено на машинній мові для більшої ефективності виконання програм. Ці функції забезпечують побудову структур даних, включаючи повний набір перемикачів, конструкторів, розпізнавачів та порівнянь
: (-Ні), (=Так), (-Частково).

14. Чи Цілі числа нескінченно великої точності та **раціональна арифметика**, визначені у кожній проектованій основній системі числення від 2 до 36, та підтримка повної множини числових функцій : (-Ні), (=Так), (-Частково).

15. Чи **Колектор**, який збирає мусор, здійснює автоматичне динамічне керування пам'яттю в усіх областях даних. Час роботи колектора як правило, менший за секунду.

.: (-Ні), (=Так), (-Частково).

16. Чи **Динамічний перерозподіл границь областей** даних відбувається автоматично при найбільш ефективному використанні усіх доступних ресурсів пам'яті (максимально до 512 К).: (-Ні), (=Так), (-Частково).

17. Чи До **конструкцій контролю** відносяться функції COND, LOOP, IF, PROGN та RETURN. Ці конструкції дозволяють створювати програми у простому та елегантному Лісп - стилі.: (-Ні), (=Так), (-Частково).

18.Чи **Функції CATCH, THROW та UNWIND-PROTECT** забезпечують структурний загальний механізм виходів, який значно спрощує контроль помилок та виняткових ситуацій у програмах користувача: (-Ні), (=Так), (-Частково).

19. Чи **Повна множина первинно визначених функцій** відображення та предикатів відображення може використовуватися для тестування елементів списків: (-Ні), (=Так), (-Частково).

20.Чи **Функції Ліспу** можуть бути визначені або як обчислювані (EVAL) або необчислювані (NORMAL), або як розгорнуті (SPRED) та нерозгорнуті (NOSPRED). Макрос, який визначається користувачем, може характеризуватися або часом компіляції, або часом виконання.: (-Ні), (=Так), (-Частково).

21. Чи **Відладка програм** полегшується використанням резидентного дисплейно-орієнтованого редактора Ліспу та пакету відладки, який включає в себе засоби трасування, переривання та збору статистики : (-Ні), (=Так), (-Частково).

22. Чи **Послідовний та випадковий методи доступу**, послідовний файл вводу-виводу повністю підтримуються системою. Окрім того, можуть бути створені файли образу пам'яті для збереження середовища Ліспу, з метою

можливості його перезагрузки у довільний час: (-Ні), (=Так), (-Частково).

23. Чи Відносно маленький розмір **Ліспу** дає можливість залишити ділянку пам'яті ЕОМ вільною, достатньою для структур даних користувача. Мінімально система займе не менше ніж 128 К пам'яті, хоча Лісп може використовувати і до 512 К.
.: (-Ні), (=Так), (-Частково).

24. Чи **Швидкість виконання програм** досягається завдяки використанню такої технології, як малі зв'язки змінних, адресні структури даних та замкнені простори вказівників.
: (-Ні), (=Так), (-Частково).

25.Чи **Визначення функцій** автоматично переводиться на "чистий" код або D-код. Процес зворотнього переводу відбувається також автоматично після повернення визначення.: (-Ні), (=Так), (-Частково).

26. Чи **Можливості обробки тексту та мови** були збільшені шляхом додавання ефективних функцій рядків.
.:(-Ні), (=Так), (-Частково).

27. Чи **Написання звичайних шаблонів** полегшено за допомогою таблиці сканера Ліспу.
: (-Ні), (=Так), (-Частково).

28. Чи **Функція сортування SORT** використовує надійне, стабільне сортування списків, яке вимагає кількість часу пропорційно nlogn, де n - довжина списку, що сортується. (-Ні), (=Так), (-Частково).

29. Чи **muLISP**-символи можуть бути зв'язані, а область пам'яті може бути розподілена для підпрограм на машинній мові.: (- Ні), (= Так), (-Частково).

30. Чи **muLISP** може бути тимчасово призупинено для того, щоб стартувати який-небудь інший процес, наприклад, редактор текстів, або навіть іншу версію muLISP. Коли цей процес припиняє своє виконання,

muLISP відновлює роботу : (- Ні), (=Так), (-Частково).

31. Чи Будь-яка структура даних є ***об'єктом***: (-Ні), (=Так), (-Частково).

32. Чи Об'єкти можуть бути двох типів: ***прості*** та ***складені***: (-Ні), (=Так), (-Частково).

33. Чи Прості об'єкти називаються ***атомами***: (-Ні), (=Так), (-Частково).

34. Чи До атомів відносяться ***символи*** та ***числа***.: (-Ні), (=Так), (-Частково).

35. Чи **Символ** не може починатися з цифри.: (-Ні), (=Так), (-Частково).

36. Чи **muLisp** не розрізняє маленькі та великі літери, а перетворює всі введені літери у великі:(-Ні), (=Так), (-Частково).

37. Чи Атом є неподільним, тобто його не можна розбити на компоненти.: (-Ні), (=Так), (-Частково).

38. Чи Атом, як і людина, має ***ім'я***. Іменами атомів є рядки символів. DOG, CAT, qw1232df, -32 є типовими іменами атомів. Символи T та NIL мають в Ліспі спеціальне призначення: вони позначають відповідно логічні значення істини та хибності. 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***.: (-Ні), (=Так), (-Частково).

39. Чи 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).

: (-Ні), (=Так), (-Частково).

40. Чи Для того щоб введений вираз не обчислювався, перед ним ставиться ***апостроф*** (‘). Якщо вираз вводиться без апострофа, то повертається його значення

: (-Ні), (=Так), (-Частково).

41. Чи При запуску програми **muLisp** значенням кожного атома вважається він сам. Значенням числа завжди є саме число, тому перед числами апостроф не ставиться. 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.

: (-Ні), (=Так), (-Частково).

42. Чи **Списки** задаються переліком елементів, взятих в дужки, перед якими ставиться апостроф. Наприклад: ‘(ice, hen) або ‘((one 1) (two 2) (three 3)).

: (-Ні), (=Так), (-Частково).

43. Чи ***Виклик*** доівльної функції у Ліспі має наступний формат:

(*name arg1 arg2 ...*), де *name* — ім'я функції, *arg1*,*arg2*, ... — її аргументи.: (-Ні), (=Так), (-Частково).

44. Чи Мова програмування Lisp має п'ять ***примітивних функцій***.

1. (**CAR** <*list*>) — знаходження голови списку.
2. (**CDR** <*list*>) — знаходження хвосту 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 об'єкт <*object*> атомом.: (-Ні), (=Так), (-Частково).

45. Чи CAR та CDR називаються ***селекторними*** функціями, оскільки вони дають можливість вибирати або знищувати 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.

$ (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

: (-Ні), (=Так), (-Частково).

46. Чи 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 літер у реалізації інтерпретатора muLisp), яка вказує шлях обчислення.

$ (CAR (CDR (CDR '(q w e r t y))))
$ (CADDR '(q w e r t y))
e

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

$ (CADDR '((q 1) (w 2) (e 3)))
(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

: (-Ні), (=Так), (-Частково).

47. Чи Функція ***конструктора*** CONS використовується для додання об'єкту до заданого списку. Об'єкт який додається, стає головою списку. Якщо другий аргумент не задано, то він вважається рівним NIL.

$ (CONS '(q w) '(r (t y))) $ (CONS apple '(q w))
((q w) r (t y)) (apple q w)

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

: (-Ні), (=Так), (-Частково).

48. Чи якщо результатом обчислення виразу (CONS *object list*) буде *new*, то результатом (CAR *new*) буде *object*, а результатом (CDR *new*) буде *list*, тобто

(CAR (CONS *object list*)) = *object*,
(CDR (CONS *object list*>)) = *list*.

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

: (-Ні), (=Так), (-Частково).

49. Чи Функцією ***порівняння*** є EQL. Вона порівнює значення першого та другого аргумента, які

обов'язково повинні бути атомами, та повертає значення істини (T) або хибності (NIL).

```
$ (EQL 'qw 'qw)                    $ (EQL (CAR '(q w)) q)
T                                  T

$ (EQL (CAR '(q, w) NIL)   $ (EQL nil '(nil))
NIL                                NIL
```

: (-Ні), (=Так), (-Частково).

50. Чи 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 '())
T                        NIL
T

$ (ATOM '(q))        $ (ATOM 3)        $ (ATOM '(NIL))
F                    T                 NIL
```

: (-Ні), (=Так), (-Частково).

51. Чи Функції ***призначення*** застосовуються для надання значень програмним змінним. До них відносяться:

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)
   — кладе символ symbol в стек (список) form.

: (-Ні), (=Так), (-Частково).

52. Чи 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)
```

: (-Ні), (=Так), (-Частково).

53. Чи 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.

: (-Ні), (=Так), (-Частково).

54.Чи Функція **PSETQ і**дентична 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

: (-Hi), (=Так), (-Частково).

55. Чи При виконанні операції заміни необхідно розрізняти символ та значення. При старті системи **mulLsp** 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

: (-Hi), (=Так), (-Частково).

56. Чи **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)

: (-Hi), (=Так), (-Частково).

57. Чи Поряд з примітивними функціями можуть існувати функції, визначені користувачем. Функція викликається набором аргументів і повертає єдине значення. **Визначення функції в Ліспі має наступний 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

: (-Hi), (=Так), (-Частково).

58. Чи Визначимо функцію **NULL**, яка розпізнає порожній список. Вона повертає істину, якщо її аргументом є порожній список і хибність в іншому випадку.

$ (DEFUN NULL (obj) $ (NULL ‘(q w e r)) $ (NULL (CDR ‘(r)))
(EQL obj NIL)) F T

: (-Hi), (=Так), (-Частково).

59. Чи Функції, які застосовуються для перевірки певних умов та можуть повертати лише два значення – істини чи хиби, називаються ***предикатами***.: (-Hi), (=Так), (-Частково).

60. Чи 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.

: (-Ні), (=Так), (-Частково).

61. Чи Якщо аргументом є список, то предикат повертає істину, інакше — хибність. Функцію **LISTP** можна проінтерпретувати наступним чином: "Якщо obj є атомом, то повернути NIL, інакше повернути T".

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

В другій колонці написано предикат LISTP, 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 як результат функції повернути T. Для другого визначення функції LISTP маємо:

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

: (-Ні), (=Так), (-Частково).

62.Чи після виконання простого завдання керування завжди передається наступному завданню (якщо таке є). Якщо предикат умовного завдання істинний, то виконується його хвіст і повертається результат останнього виразу хвоста.

: (-Ні), (=Так), (-Частково).

63. Чи Вмонтована 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)   $ (LIST 'a '(b c) 'd)  $ (LIST)
(a b c d)              (a (b c) d)             NIL
```

: (-Ні), (=Так), (-Частково).

64. Чи функція **MEMBER**, 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 MEMBER (nam lst)                $ (DEFUUN MEMBER (nam lst)
  ((EQL nam (FIRST lst)))                ((NULL lst) NIL)
  ((EQL nam (SECOND lst)))               ((EQL nam (CAR lst)) T)
  ((EQL nam (THIRD lst)))                (MEMBER nam (CDR lst)) )
  ((EQL nam (THIRD (CAR lst))))
  . . . . . .
  : (-Hi), (=Так), (-Частково).
```

65. **Чи справедливі твердження**: Змінимо наш підхід до побудови функції. В другому стовпчику побудовано функцію MEMBER, 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)                (DEFUN REMBER-ALL (obj lst)
  ((NULL lst) NIL)                       ((NULL lst) NIL)
```



```
  ((EQL obj (CAR lst)) (CDR lst))        ((EQL obj (CAR lst))
  (CONS (CAR lst)                        (REMBER-ALL obj (CDR lst))
  (REMBER obj (CDR lst))) )              (CONS (CAR lst)
                                         (REMBER-ALL obj (CDR lst))))
  : (-Hi), (=Так), (-Частково).
```

66. Чи 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)                $ (DEFUN NOT (obj)
  ((NULL lst1) (NULL lst2))
     (EQL obj NIL) )
  ((NULL lst2) NIL)
  ((NOT (EQL (CAR lst1) (CAR lst2))) NIL)
  (EQLIST (CDR lst1) (CDR lst2)) )
  : (-Hi), (=Так), (-Частково).
```

67. Чи 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))
)
```

: (-Ні), (=Так), (-Частково).

68. Чи Середовище muLisp або поточний стан системи складається з усіх активних на даний момент структур даних, значень змінних та визначених функцій. Команда SAVE зберігає поточне середовище muLisp у вигляді SYS - файлу. Команда (SAVE 'C:HOME) зберігає середовище в файл HOME.SYS на диску C. Після успішного виконання команди запису повертається T, інакше — NIL.

: (-Ні), (=Так), (-Частково).

69. Чи Середовище **muLisp** може бути завантажене за допомогою команди LOAD: (LOAD <file>). Якщо файл не знайдено, повертається NIL, інакше жодне значення не повертається, а mulisp починає працювати з новим середовищем

: (-Ні), (=Так), (-Частково).

70. Чи Для завантаження SYS-файлів безпосередньо після запуску muLISP може використовуватися команда операційного середовища (ОС). Наприклад, команда ОС

> muLISP C:HOME

завантажує SYS-файл HOME.SYS із пристрою C після запуску muLISP із пристрою, взятого по замовченню. Відмітимо, що тип SYS-файла у команді не вказується. Якщо SYS-файл не знайдено при завантаженні з

використанням команди ОС, на екран дисплею видається повідомлення: File not found.

Після завантаження SYS-файла кількість пам'яті, призначеної для кожної області даних, корегується у відповідності до поточного об'єму пам'яті. Це означає, що поточний об'єм пам'яті не обов'язково повинен бути точно таким, як і при створенні SYS-файлів. Але якщо пам'яті для розташування середовища SYS-файлів недостатньо, то виникає помилка типу "Недостатньо пам'яті, переривання", і muLISP буде завершено.

(-Ні), (=Так), (-Частково).

71. Чи **Асоціативним списком** називається список пар (тобто cons-ів), які використовуються у muLISP для зв'язку ключа та об'єкта. Функції ASSOC та ASSOC-IF належать класу функцій відбору, які дають можливість отримати доступ до об'єкта, пов'язаному з ключем, який задовольняє тесту.

: (-Ні), (=Так), (-Частково).

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

1. (+ num1 num2 ... numM)
2. (- num1 num2 ... numM)
3. (* num1 num2 ... numM)
4. (/ num1 num2 ... numM)

: (-Ні), (=Так), (-Частково).

73. Чи **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

: (-Ні), (=Так), (-Частково).

74. Чи **muLISP-**програми можуть автоматично генерувати нові структури даних, використовуючи функції конструктора. Ці функції можуть утворювати бінарні дерева або зв'язні списки, які моделюють структури даних практично для довільної задачі.. : (-Ні), (=Так), (-Частково).

75. Чи Функції властивостей призначені для керування властивостями, пов'язаними із символами. CDR - елемент символа вказує на список властивостей, який містить властивості та прапорці. : (-Ні), (=Так), (-Частково).

76. Чи **Список властивостей** - це ASSOC-список (ASSOCiation) ключей властивостей, об'єднаних у пари зі значеннями властивостей (див. Опис ASSOC у розділі 4.1). Оскільки прапорці - це атоми у списку властивостей, вони можуть відрізнятися від властивостей.: (-Ні), (=Так), (-Частково).

77. Чи Для огляду **Caml** використовується інтерактивна система, яка запускається командою ocaml з оболонки Unix або додатком Ocamlwin.exe в середовищі Windows. Все введення є журналом однієї сесії. : (-Ні), (=Так), (-Частково).

78. Чи **Інтерактивна система** виводить запрошення #, чекає від користувача введення фраз Caml, які той закінчує символами ;;, потім компілює їх на льоту, виконує і виводить результат. Фрази можуть бути простими виразами або визначеннями ідентифікаторів (змінних або функцій) let.

```
# 1+2*3;;
- : int = 7
# let pi = 4.0 *. atan 1.0;;
val pi : float = 3.14159265359
# let square x = x *. x;;
#val square : float -> float = <fun>
# square(sin pi)+. square(cos pi);;
- : float = 1.
```

: (-Ні), (=Так), (-Частково).

79. Чи **Система Caml** обчислює як значення, так і тип фрази. Навіть у випадку з аргументами функції явна вказівка типу не потрібна. Воно виводиться з того, як змінні використовуються усередині функції. Але слід звернути увагу на те, що цілі числа і числа з плаваючою крапкою є різними типами з різними операторами: + і * призначені для цілих, а +. і *. для чисел з плаваючою крапкою.

```
# 1.0 * 2;;
This expression has type float but is here used with type int
```

Рекурсивні функції визначаються конструкцією let rec.

```
# #      let rec fib n = if n < 2 then 1 else fib(n - 1)+ fib(n -
2);;
val fib: int -> = <fun>
# #      fib 10;;
-: int = 89
```

(-Ні), (=Так), (-Частково).

80. Чи Крім цілих чисел і чисел з плаваючою крапкою Caml підтримує звичні типи даних - булеві значення, символи і символьні рядки.

```
# (1 < 2) = false;;
- : bool = false
# 'a';;
- : char = 'a'
# #      "Hello world";;
- : string = "Hello world"
```

: (-Ні), (=Так), (-Частково).

81. Чи **Списки**, як і інші структури даних, не треба явно розміщувати і знищувати в пам'яті: у Caml управління пам'яттю повністю автоматичне. Аналогічно, явного управління покажчиками немає - компілятор Caml сам при необхідності створює покажчики.

: (-Ні), (=Так), (-Частково).

82. Чи Дослідження і зміна структури списків, як і більшості інших структур даних здійснюється за допомогою пошуку за зразком. Зразки в цьому випадку записуються в тій же формі, що і визначення списків, а ідентифікатор представляє невідому частину списку. Ось приклад сортування списку вставкою:

```
# let rec sort lst =
      match lst with
          [] -> []
      | head :: tail -> insert head (sort tail)
and insert elt lst =
      match lst with
          [] -> [elt]
      | head :: tail -> if elt <= head then elt :: lst else head ::
insert elt tail;;
val sort: 'a list -> 'a list = <fun>
val insert 'a -> 'a list -> 'a list = <fun>
# sort l;;
-: string list = ["a"; "etc."; "is"; "tale"; "told"]
```

: (-Ні), (=Так), (-Частково).

83. Чи Тип даних, обчислений для функції sort('a list -> 'a list), означає, що вона може працювати із списками, що складаються з даних будь-якого типу і повертати такі ж списки. Тип 'a є змінним типом і може відповідати будь-якому типу даних. Функція sort може працювати із списками, що складаються з будь-яких типів, оскільки операції порівняння в Caml (=, <= і т.д.) поліморфічни:

вони працюють з будь-якими змінними одного типу. Таким чином, і функція sort стає поліморфічной.

```
# sort [6;2;5;3;];;
- : int list = [2; 3; 5; 6]
# sort [3.14; 2.718];;
- : float list = [2.718; 3.14]
```

: (-Ні), (=Так), (-Частково).

84. Чи Функція **sort** не змінює початковий список. Вона будує і повертає новий список, що складається з тих же елементів, розташованих у порядку зростання. У Caml немає можливості змінити вже створений список, тому він є незмінним (immutable). Те ж саме відноситься до більшості структур даних в Caml, проте існують і змінні (mutable) типи (в першу чергу, масиви). : (-Ні), (=Так), (-Частково).

85. Caml - функціональна мова. Він підтримує функції в математичному значенні, так що останні можуть оброблятися як звичні дані. Наприклад, функція deriv приймає як аргументу будь-яку функцію, що оперує числами з плаваючою крапкою, і повертає її похідну:

```
# let deriv f dx = function x -> (f (x +. dx) -. f(x)) /. dx;;
val deriv: (float -> float) -> float -> float -> float = <fun>
# let sin' = deriv sin 1e-6;;
val sin' : float -> float = <fun>
```



```
# sin' pi;;
-: float = -1.00000000014
```

: (-Ні), (=Так), (-Частково).

86. Чи Функції, що приймають як аргумент інші функції, називаються "функціоналами" або "функціями вищого порядку". Вони особливо зручні, коли виникає потреба в ітераторі або подібній йому операції для структури даних. Стандартна бібліотека Caml включає фукнкционал List.map, що застосовує функцію до кожного елементу списку і повертаючий список, складений з результатів функції.

```
# List.map(function n -> n*2 + 1) [0;1;2;3;4];;
- : int list = [1; 3; 5; 7; 9]
```

: (-Ні), (=Так), (-Частково).

87. Чи Призначені для користувача типи даних включають записи і варіанти. І ті, і інші визначаються декларацією type. Нижче приведене визначення типу раціонального числа у вигляді запису.

```
# type ratio = {num: int; denum: int};;
type ratio =  {num: int; denum: int};;
# let add_ratio r1 r2 =
    {num = r1.num * r2.denum + r2.num * r1.denum;
{denum = r1.denum * r2.denum};;
val add_ratio : ratio -> ratio -> ratio = <fun>
# add_ratio {num = 1; denum =3} {num = 2; denum =5};;
- : ratio = {num=11; denum=15}
```

: (-Ні), (=Так), (-Частково).

88. Чи У визначенні варіантного типу перераховуються всі можливі форми його значення. Кожна з них задається по імені, яке називається конструктором і служить як для створення значень варіантного типу, так і для дослідження шляхом зіставлення із зразком. Конструктори записуються із заголовної букви - так їх можна відрізнити від імен змінних (останні повинні починатися з рядковою). Ось, наприклад, варіантний тип для змішаної арифметики, що допускає операції з цілими і числами з плаваючою крапкою:

```
# type number = Int of int | Float of float | Error;;
        type number = Int of int | Float of float | Error
```

- : (-Ні), (=Так), (-Частково).

89. Чи Значення типу number може бути цілим, числом з плаваючою крапкою, або константою Error, відповідною результату неприпустимої операції (наприклад, розподіли на нуль). : (-Ні), (=Так), (-Частково).

90. Чи Особливим випадком варіантних типів є перераховувані типи. Всі альтернативи в них - константи.

```
# type sign = Positive | Negative;;
type sign = Positive | Negative
# let sign_int n = if n => 0 then Positive else Negative;;
val sign_int: int -> sign = <fun>
```

: (-Ні), (=Так), (-Частково).

91. Чи Операції з бінарними деревами записуються у вигляді рекурсивних функцій тієї ж структури, що і визначення типу. Ось, наприклад, операції пошуку і вставки у впорядковане бінарне дерево (елементи зростають зліва направо):

```
# let rec member x btree =
    match btree with
        Empty -> false
        | Node(y, left, right) ->
            if x = y then true else
            if x < y then member x left else member x right;;
val member : 'a -> 'a btree -> bool = <fun>
# let insert x btree =
    match btree with
        Empty -> Node (x, Empty, Empty)
        | Node(y, left, right) ->
            if x <= y then Node(y, insert x left, right)
 љ љ љ љ else Node(y, left, insert x right);;
val insert : 'a -> 'a btree -> 'a btree = <fun>
```

: (-Ні), (=Так), (-Частково).

92. Чи Виключення використовуються в Ocaml для сповіщення про виняткові ситуації і для їх обробки. Крім того, вони можуть застосовуватися як нелокальні контрольні структури загального призначення. Виключення оголошуються блоком exception і збуджуються оператором raise

: (-Ні), (=Так), (-Частково).

93. Чи У гілці **with** насправді виконується порівняння із зразком.: (-Ні), (=Так), (-Частково).

94. Чи для того, щоб врахувати пріоритет і асоціативність операторів, в рекурсивних низхідних аналізаторах використовуються проміжні функції розбору

: (-Ні), (=Так), (-Частково).

95. Чи Обробка потоку переважно є обробкою регулярної структури даних, оскільки дозволяє рекурсивно викликати функції розбору всередині зразків, виділяючи підкопоненти потоку : (-Ні), (=Так), (-Частково).

96. Чи код Ocaml може бути скомпільований компіляторами ocamlc і ocamlopt: (-Ні), (=Так), (-Частково).

97. Чи Початкові тексти звичайно зберігаються у файлах з розширенням .ml.

: (-Ні), (=Так), (-Частково).

98. Чи **Деревом** називається граф без циклів, в якому виділено окрему вершину, яку називають коренем дерева.

: (-Ні), (=Так), (-Частково).

99. Чи Структурою типу дерева будемо називати або NIL (порожнє дерево), або структуру **(Значення . (Ліва вітка . Права вітка))**, де ліва і права вітки є структурами типа дерево : (-Ні), (=Так), (-Частково). .

100. Чи Наступні дві функції виконують обхід дерева: PUD (Print Up-Down) — обхід згори вниз, PLR (Print Left-Right) — обхід зліва направо

: (-Ні), (=Так), (-Частково).

Руслан Миколайович Літнарович
кандидат технічних наук, доцент

ТЕСТОВІ ЗАВДАННЯ
по дисципліні:
«Аплікативні системи»

для магістрантів факультету Кібернетики

ФАКУЛЬТЕТ КІБЕРНЕТИКИ

КАФЕДРА МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ

Комп'ютерний набір, верстка, редагування і макетування та дизайн в редакторі Microsoft ® Offise ® Word 2003
Р.М.Літнарович
Відповідальний редактор Й.В. Джунь
Підп.до друку 11. 12. 2010 р.
Формат 60x84/16. Папір офсетн.№1.
Гарнітура Times New Roman.
Друк різограф. Тираж 300 пр.
Редакційно-видавничий центр «Тетіс»
Міжнародного економіко-гуманітарного університету
Імені академіка Степана Дем'янчука
33027 Рівне , Україна
Вул..С.Дем'янчука, 4, корпус 1
Телефон : (+00380) 362 23 – 73 – 09
Факс :(+00380) 362 23 – 01 – 86
E-mail:mail@regi.rovno.ua