Програмування інтелектуальних інформаційних систем

3 курс, осінь 2021

- Доц. Баклан І.В.
- Email: iaa@ukr.net
- Web: baklaniv.at.ua

Лекція 2

Мова штучного інтелекту

LISP

```
СТУДЕНТ -> УЧАЩИЙСЯ
 |
 |->  ПІБ -> ТЕКСТ -> ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ
V
 |->  ГРУППА  -> ТЕКСТ ->  ІП-91
V
 |->  ФАКУЛЬТЕТ -> ТЕКСТ -> ІОТ
V
 |->  ДНЮХА  -> ДАТА  -> 25 2 2000
```

Треба створити програмну структуру, за допомогою якої зберігаються подібні дані.

Для цього використовуємо LISP. Структуру подаємо у вигляді складного списку.

```
(СТУДЕНТ (УЧАЩИЙСЯ)

 ((ПІБ (ТЕКСТ (ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ)))

  (ГРУППА (ТЕКСТ (ІП-91)))

  (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ)))

  (ДНЮХА (ДАТА (25 2 2000))))))
```

Для зберігання списку в середовищі LISP використовуємо виклик наступної функції:

(**setq** cpisok `(СТУДЕНТ (УЧАЩИЙСЯ) ((ПІБ (ТЕКСТ (ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ))) (ГРУППА (ТЕКСТ (ІП-91))) (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ))) (ДНЮХА (ДАТА (25 2 2000))))))

```
ighor@ighor-notebook:~$ clisp
  i i i i i i i       ooooo    o        ooooooo   ooooo   ooooo
  I I I I I I I      8     8   8           8     8     o  8    8
  I  \ `+' /  I      8         8           8     8        8    8
   \  `-+-'  /       8         8           8      ooooo   8oooo
    `-__|__-'        8         8           8           8  8
        |            8     o   8           8     o     8  8
  ------+------       ooooo    8oooooo  ooo8ooo   ooooo   8

Добро пожаловать GNU CLISP 2.49.60+ (2017-06-25) <http://clisp.org/>

Copyright (c) Bruno Haible, Michael Stoll 1992, 1993
Copyright (c) Bruno Haible, Marcus Daniels 1994-1997
Copyright (c) Bruno Haible, Pierpaolo Bernardi, Sam Steingold 1998
Copyright (c) Bruno Haible, Sam Steingold 1999-2000
Copyright (c) Sam Steingold, Bruno Haible 2001-2010

Напечатайте :h и нажмите Ввод для получения справки.

[1]>
```

Спочатку запустимо середовище Common Lisp в терміналі ОС Ubuntu.

Після отримання запрошення > можемо ввести нашу команду.

```
[1]> (setq cpisok `(СТУДЕНТ (УЧАЩИЙСЯ)
 ((ПІБ (ТЕКСТ (ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ))) (ГРУППА (ТЕКСТ (ІП-91)))
  (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ))) (ДНЮХА (ДАТА (25 2 2000))))))
(СТУДЕНТ (УЧАЩИЙСЯ)
 ((ПІБ (ТЕКСТ (ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ))) (ГРУППА (ТЕКСТ (ІП-91)))
  (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ))) (ДНЮХА (ДАТА (25 2 2000)))))
[2]>
```

Після введення функції інтерпретатор виводить на екран значення функції. В нашому випадку список.

Функція встановлює зв’язок між іменем `cpisok` та нашим списком.

```
[2]> cpisok
(СТУДЕНТ (УЧАЩИЙСЯ)
 ((ПІБ (ТЕКСТ (ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ))) (ГРУППА (ТЕКСТ (ІП-91)))
  (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ))) (ДНЮХА (ДАТА (25 2 2000)))))
[3]>
```

```
[3]> (car cpisok)
СТУДЕНТ
[4]>
```

За допомогою функції car отримуємо “голову” списка.

```
[4]> (cdr cpisok)
((УЧАЩИЙСЯ)
 ((ПІБ (ТЕКСТ (ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ))) (ГРУППА (ТЕКСТ (ІП-91)))
  (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ))) (ДНЮХА (ДАТА (25 2 2000)))))
[5]>
```

Отримання хвоста списку.

Складна функція

```
[5]> (caadr cpisok)
УЧАЩИЙСЯ
[6]>
```

Визначення користувацької функції.

```
[6]> (defun сінонім (x) (caadr x))
СІНОНІМ
[7]> (сінонім cpisok)
УЧАЩИЙСЯ
[8]>
```

Продовжимо розбір нашого списку на окремі частини.

```
[8]> (caaddr cpisok)
(ПІБ (ТЕКСТ (ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ)))
[9]> (cdaddr cpisok)
((ГРУППА (ТЕКСТ (ІП-91))) (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ))) (ДНЮХА (ДАТА (25 2 2000))))
[10]> (car (cdaddr cpisok))
(ГРУППА (ТЕКСТ (ІП-91)))
[11]> (cadadr (car (cdaddr cpisok)))
(ІП-91)
[12]>
```

```
(caaddr cpisok)

(cdaddr cpisok)

(car (cdaddr cpisok))

(cadadr (car (cdaddr cpisok)))
```

Визначим функцію, значенням якої буде назва групи з нашого списку:

```
[12]> (defun какая-группа (x)(car (cadadr (car (cdaddr x)))))
КАКАЯ-ГРУППА
[13]> (какая-группа cpisok)
ІП-91
[14]>
```

```
(defun какая-группа (x)(car (cadadr (car
(cdaddr x)))))

(какая-группа cpisok)
```

Додамо в середовище ще один список аналогічної структури.

```
[14]> (setq cpisok2 `(СТУДЕНТ (УЧАЩИЙСЯ)
 ((ПІБ (ТЕКСТ (ДЖУРА ОКСАНА ПЕТРІВНА))) (ГРУППА (ТЕКСТ (ІП-91)))
  (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ))) (ДНЮХА (ДАТА (31 6 2000))))))
(СТУДЕНТ (УЧАЩИЙСЯ)
 ((ПІБ (ТЕКСТ (ДЖУРА ОКСАНА ПЕТРІВНА))) (ГРУППА (ТЕКСТ (ІП-91)))
  (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ))) (ДНЮХА (ДАТА (31 6 2000)))))
[15]>
```

```
(setq cpisok2 `(СТУДЕНТ (УЧАЩИЙСЯ)  ((ПІБ
(ТЕКСТ (ДЖУРА ОКСАНА ПЕТРІВНА))) (ГРУППА
(ТЕКСТ (ІП-91)))   (ФАКУЛЬТЕТ (ТЕКСТ (ІОТ)))
(ДНЮХА (ДАТА (31 6 2000))))))
```

Перевіримо як працють наші користувацькі функції на новому списку.

```
[15]> (сінонім cpisok2)
УЧАЩИЙСЯ
[16]> (какая-группа cpisok2)
ІП-91
[17]>
```

(сінонім cpisok2)

(какая-группа cpisok2)

```
[3]> (defun ПІБ (x) (cadar (cdr (caaddr x))))
ПІБ
[4]> (піб cpisok)
(ПЕТРЕНКО ІВАН МИКОЛАЙОВИЧ)
[5]> (піб cpisok2)
(ДЖУРА ОКСАНА ПЕТРІВНА)
```

Визначимо функцію, яка видає список прізвища, ім’я та по-батькові з наших списків.

```
(defun ПІБ (x) (cadar (cdr (caaddr x))))

(піб cpisok)

(піб cpisok2)
```

Визначимо функцію яка видає лише прізвище з нашого списку.

```
[8]> (defun прізвище (x) (car (піб x)))
ПРІЗВИЩЕ
[9]> (прізвище cpisok)
ПЕТРЕНКО
[10]> (прізвище cpisok2)
ДЖУРА
[11]>
```

```
(defun прізвище (x) (car (піб x)))

(прізвище cpisok)

(прізвище cpisok2)
```

Визначимо функцію яка видає лише ім’я з нашого списку.

```
[11]> (defun імя (x)(cadr (піб x)))
ІМЯ
[12]> (імя cpisok)
ІВАН
[13]> (імя cpisok2)
ОКСАНА
[14]>
```

```
(defun імя (x)(cadr (піб x)))

(імя cpisok)

(імя cpisok2)
```

Визначимо функція, яка видає список, що складається з прізвища та імені.

```
[14]> (defun хто (x) (list (прізвище x)(імя x)))
ХТО
[15]> (хто cpisok)
(ПЕТРЕНКО ІВАН)
[16]> (хто cpisok2)
(ДЖУРА ОКСАНА)
[17]>
```

Таким чином за короткий проміжок часу ми створили простеньку мову для роботи з базою знань.

Завдання: 1. Додати до спискової структури ще підструктури “Телефон” та “Адреса”.

2. Створити список зі своїми власними даними.

3. Написати функції “який-телефон” та “яка-адреса”, які видають відповідні значення з введених списків.

Більшість операторів в Common Lisp - це функції, але не всі. Виклики функцій завжди обробляються подібним чином. Аргументи обчислюються зліва направо і потім передаються функції, яка повертає значення всього виразу. Цей порядок називається правилом обчислення для Common Lisp.

Проте існують оператори, які не дотримуються прийнятого в Common Lisp порядку обчислень. Один з них - **`quote`**, або оператор цитування. **`quote`** - це спеціальний оператор; це означає, що у нього є власне правило обчислення, а саме: нічого не робити.

Фактично **quote** бере один аргумент і просто повертає його текстову запис:

```
[17]> (quote (+ 3 5))
(+ 3 5)
[18]>
```

Для зручності в Common Lisp можна замінювати оператор **quote** на лапки. Той же результат можна отримати, просто поставивши **'** перед цитованим виразом:

```
> '(+3 5)

(+3 5)
```

В явному вигляді оператор `quote` майже не використовується, більш поширена його скорочена запис з лапками.

Цитування в Ліспі є способом захисту вираження від обчислення.

У Ліспі є два типи, які рідко використовуються в інших мовах, - символи і списки. Символи - це слова. Зазвичай вони перетворюються до верхнього регістру незалежно від того, як ви їх ввели:

```
[20]> `Artichoke
ARTICHOKE
[21]>
```

Символи, як правило, не є самообчислюваним типом, тому, щоб послатися на символ, його необхідно цитувати, як показано вище.

Список - це послідовність з нуля або більше елементів, укладених в дужки. Ці елементи можуть належати до будь-якого типу, в тому числі можуть бути іншими списками. Щоб Лисп не вважав за список викликом функції, його потрібно процитувати:

```
[23]> `(my 3 "Sons")
(MY 3 "Sons")
[24]> `(the list (a b c) has 3 elements)
(THE LIST (A B C) HAS 3 ELEMENTS)
[25]>
```

Прийшов час оцінити одну з найбільш важливих особливостей Лиспа. Програми, написані на Ліспі, представляються у вигляді списків. Якщо наведені раніше доводи про гнучкість і елегантності не переконали вас в цінності прийнятої в Ліспі нотації, то, можливо, цей момент змусить вас змінити свою думку. Саме ця особливість дозволяє програмам, написаним на Ліспі, генерувати Лисп-код, що дає возожность розробнику створювати програми, які пишуть програми.

```
[27]> (list `(+ 2 1) (+ 2 1))
((+ 2 1) 3)
[28]>
```

Список може бути порожнім. В Common Lisp можливі два типи уявлення порожнього списку: пара порожніх дужок і спеціальний символ **nil**. Незалежно від того, як ви введете порожній список, він буде відображений як **nil**.

```
[28]> ()
NIL
[29]> nil
NIL
[30]>
```

Перед **()** необязательно ставить кавычку, так как символ **nil** самообчислюваний.

Побудова списків здійснюється за допомогою функції cons. Якщо другий її аргумент - список, вона повертає новий список з першим аргументом, доданим до його початок:

```
> (Cons 'a' (b c d))

(A B C D)
```

Список з одного елемента також може бути створений за допомогою cons і порожнього списку. Функція list, з якої ми вже познайомилися, - всього лише більш зручний спосіб послідовного використання cons.

```
> (Cons 'a (cons 'b nil))

(A B)

> (List 'a 'b)

(A B)
```

В Common Lisp істинність за замовчуванням видається символом **t**. Як і **nil**, символ **t** є самообчислювальним. Наприклад, функція **listp** повертає істину, якщо її аргумент - список:

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

T
```

Функції, які повертають логічні значення «істина» або «брехня», називаються предикатами. В Common Lisp імена предикатів часто закінчуються на «**p**».

Брехня в Common Lisp представляється за допомогою **nil**, порожнього списку. Застосовуючи **listp** до аргументу, який не є списком, отримаємо **nil**:

```
> (Listp 27)

NIL
```

Оскільки **nil** має два значення в Common Lisp, функція **null**, що має справжнє значення для пустого списку:

```
> (Null nil)

T
```

і функція **not**, яка повертає істинне значення, якщо її аргумент хибний:

```
> (Not nil)

T
```

роблять одне і те ж.

Найпростіший умовний оператор в Common Lisp - `if`. Зазвичай він примать три аргументи: `test-`, `then-` і `else`-вирази. Спочатку обчислює тестове `test`-вираз. Якщо воно істинне, обчислюється `then`-вираз («то») і повертається його значення. В іншому випадку обчислюється `else`-вираз («інакше»).

```
> (if (listp '(a b c))

(+ 1 2)

(+5 6))

3
```

```
> (if (listp 27)

(+ 1 2)

(+5 6))

11
```

Як і `quote`, `if` - це спеціальний оператор, а не функція, так як функції обчислюються всі аргументи, а у оператора `if` обчислює лише один з двох останніх виразів.

Вказувати останній аргумент `if` необов'язково. Якщо його пропущено, автоматично приймається за `nil`.

```
> (if (listp 27)

(+2 3))

NIL
```

Незважаючи на те, що за замовчуванням істина представляється у вигляді **t**, будь-який вираз, крім **nil**, також вважається істинним:

```
> (if 27 1 2)

1
```

Логічні оператори **and** (і) і **or** (або) діють схожим чином.

Обидва можуть приймати будь-яку кількість аргументів, але обчислюють їх до тих пір, поки не буде ясно, яке значення необхідно повернути. Якщо всі аргументи істинні (тобто не **nil**), то оператор **and** поверне значення останнього:

```
> (and t (+ 1 2))

3
```

Але якщо один з аргументів виявиться помилковим, то наступні за ним аргументи не будуть обчислені. Так само діє і **`or`**, обчислюючи значення аргументів до тих пір, поки серед них не знайдеться хоча б одне справжнє значення.

Ці два оператора - макроси. Як і спеціальні оператори, макроси можуть обходити звичайний порядок обчислення.

Нові функції можна визначити за допомогою оператора **defun**. Він зазвичай приймає три або більше аргументів: ім'я, список параметрів і одне або більше виразів, які складають тіло функції. Ось як ми можемо за допомогою **defun** визначити функцію **third**:

```
> (third '(a b c d))

C

> (defun our-third (x)

(car (cdr (cdr x))))

OUR-THIRD
```

Перший аргумент задає ім'я функції, в нашому прикладі це **our-third**. Другий аргумент, список **(x)**, повідомляє, що функція може приймати строго один аргумент: **x**. Використовуваний тут символ **x** називається змінною. Коли змінна є аргумент функції, як **x** в цьому прикладі, вона ще називається параметром.

Частина, що залишилася, **(car (cdr (cdr x)))**, називається тілом функції. Вона повідомляє Ліспі, що потрібно зробити, щоб повернути значення з функції. Виклик **our-third** повертає **(car (cdr (cdr x)))**, яке б значення аргументу **x** не було задано:

```
> (Our-third '(a b c d))
C
```

У Ліспі немає відмінностей між програмою, процедурою і функцією. Все це функції (та й сам Лисп здебільшого складається з функцій). Не має сенсу визначати одну головну функцію, адже будь-яка функція може бути викликана в `toplevel`. У числі іншого, це означає, що програму можна тестувати по маленьких шматочках в процесі її написання.

Функції, що викликають самі себе, називаються рекурсивними. В Common Lisp є функція member, яка перевіряє, чи є в списку якої-небудь об'єкт. Нижче приведена її спрощена реалізація:

```
(defun our-member (obj lst)
(if (null lst)
nil
(if (eql (car lst) obj)
lst
(our-member obj (cdr lst)))))
```

Предикат `eql` перевіряє два аргументи на ідентичність. Все інше в цьому виразі вам повинно бути вже знайоме.

```
> (our-member 'b' (a b c))
(B C)
> (our-member 'z' (a b c))
NIL
```

Опишемо словами, що робить ця функція. Щоб перевірити, чи є `obj` в списку `lst`, ми:

1. Перевіряємо, порожній чи список `lst`. Якщо він порожній, значить, `obj` не присутній в списку.
2. Якщо `obj` є першим елементом `lst`, значить, він є в цьому списку.
3. В іншому випадку перевіряємо, чи є `obj` серед решти елементів списку `lst`.

Один з найбільш часто використовуваних операторів у Common Lisp - це **let**, який дозволяє вам ввести нові локальні змінні:

```
> (let ((x 1) (y 2))
(+ x y))
3
```

Вираз з використанням **let** складається з двох частин. Перша містить інструкції, що визначають нові змінні. Кожна така інструкція містить ім'я змінної і відповідне їй вираз.

Вище нами вже вводилися **cons, car** і **cdr** - найпростіші функції для маніпуляцій зі списками. Насправді, cons об'єднує два об'єкти в один, званий клітинкою (**cons**). Якщо бути точніше, то **cons** - це пара покажчиків, перший з яких вказує на **car**, другий - на **cdr**.

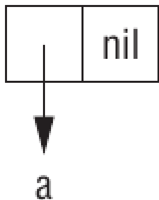
За допомогою **cons**-осередків зручно об'єднувати в пару об'єкти будь-яких типів, в тому числі і інші осередки. Саме завдяки такій можливості за допомогою **cons** можна будувати довільні списки.

Нема чого представляти кожен список у вигляді **cons**-клітинок, але потрібно пам'ятати, що вони можуть бути задані таким чином. Любий непустой список може вважатися парою, що містить перший елемент списку і решту його частину. У Ліспі списки є втіленням цієї ідеї. Саме тому функція **car** дозволяє отримати перший елемент списку, а **cdr** - його залишок (який є або **cons**-клітинкою, або **nil**). І домовленість завжди була такою: використовувати car для позначення першого елементу списку, а **cdr** - для його залишку. Так ці назви стали синонімами операцій **first** і **rest**. Таким чином, списки - це не окремий вид об'єктів, а всього лише набір пов'язаних між собою **cons**-клітинок.

Якщо ми спробуємо використовувати `cons` разом з `nil`,

```
> (setf x (cons 'a nil))
(A)
```

то отримаємо список, що складається з одного осередку, як показано на рис. 1.



Мал. 1. Список, що складається з однієї комірки

Такий спосіб зображення осередків називається блоковим, тому що кожна клітинка представляється у вигляді блоку, що містить покажчики на **car** і **cdr**. Викликаючи **car** або **cdr**, ми отримуємо об'єкт, на який вказує відповідний покажчик:

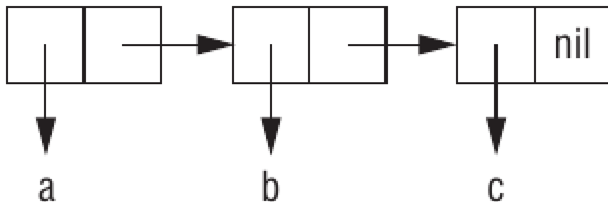
```
> (car x)
A
> (cdr x)
NIL
```

Складаючи список з декількох елементів, ми отримуємо ланцюжок осередків:

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

Ця структура показана на рис. 2. Тепер **cdr** списку буде вказувати на список з двох елементів:

```
> (cdr y)
(B C)
```



Мал. 2. Список з трьох комірок

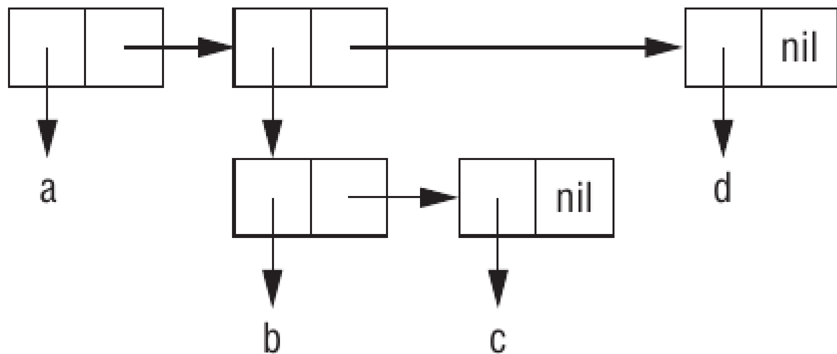
Для списку з кількох елементів покажчик на car дає перший елемент списку, а покажчик на **cdr** - його залишок.

Елементами списку можуть бути будь-які об'єкти, в тому числі і інші списки:

```
> (setf z (list 'a (list' b 'c)' d))
(A (B C) D)
```

Відповідна структура показана на рис. 3; **car** другого осередку вказує на інший список:

```
> (car (cdr z))
(B C)
```



Мал. 3. Вкладений список

Наступна лекція буде присвячена продовженню розгляду мови Common Lisp.