# This page is intentionally left blank.

Kazimir Majorinc

# LISP 2

## Povijest Lispa 27.

Razmjena vještina
Hacklab u mami
13. travnja 2013.

Ubrzo nakon izdavanja knjige Lisp 1.5 počeo je razvoj jezika Lisp 2 u kojem je isprva sudjelovao i **John McCarthy**[1]. Tijekom sprnja **1963**.[2] na sveučilištu Stanford je održana LISP 2 SPECIFICATION CONFERENCE[3] iz koje je vidljivo da je motivacija za novi jezik nije bila potraga za „apstraktnijim" ili „čišćim" koncepcijama nego potreba za praktičnijim i bržim alatom.

---

[1] **McCarthy**, Storage conventions in LISP 2, SAIP Memo 8, **1963**

[2] Ibid. p 1.

[3] Program konferencije je objavljen na softwarepreservation.org

McCarthy[4] je kasnije zapisao:

> As a programming language, LISP had many limitations... ultraslow numerical computation, inability to represent objects by blocks of registers and garbage collect the blocks, and lack of a good system for input/output of symbolic expressions in conventional notations. All these problems and others were to be fixed in LISP 2. In the meantime, we had to settle for LISP 1.5, developed at MIT, which corrected only the most glaring deficiencies.
>
> The LISP 2 project was a collaboration of SYSTEMS DEVELOPMENT CORPORATION and INFORMATION INTERNATIONAL INC. ... The project proved more expensive than expected, the collaboration proved more difficult than expected, and so LISP 2 was dropped. From a 1970s point of view, this was regrettable, because much more money has since been spent to develop LISPs with fewer features.

---

[4] **McCarthy**, History of Lisp, p. 181

Projekt je bio vrlo ozbiljan, o čemu svjedoči preko pedeset članaka, tehničkih izvještaja i memoa objavljenih u periodu između **1963**. i **1967**. Najvažniji od tih dokumenata je, vjerojatno, članak grupe autora „Lisp 2 programming language and system.[5]"

U članku se svrha jezika opisuje sa

> LISP 2 is a new programming language designed for use in problems that require manipulation of highly complex data structures as well as lengthy arithmetic operations.[6]

---

[5] **Abrahams**, **Barnett** et al., *Lisp 2 programming language and system*, Proceedings of the Fall Joint Computer Conference, 1966, pp. 661-676

[6] Ibid., p. 661.

Velika promjena je uvođenje dva različita nivoa jezika: Source Language (SL) i Intermediate Language (IL). Nivoi imaju različitu sintaksu ali istu semantiku. Sintaksa ta dva jezika liči na sintaksu Algola i Lispa 1.5, redom. **Abrahams** i ostali navode primjer:

```
% RANDOM COMPUTES A RANDOM
  NUMBER IN THE INTERVAL (A,B)
 OWN INTEGER Y;
 REAL FUNCTION RANDOM(A,B);
    REAL A,B;
 BEGIN Y ← 3125*Y;
       Y ← Y\67108864;
       RETURN (Y/67108864.0 * (B - A) + A)
 END;
```

```
(DECLARE (Y OWN INTEGER))
(FUNCTION (RANDOM REAL)
          ((A REAL) (BREAL))
          (BLOCK NIL
                 (SET Y (TIMES 3125 Y))
                 (SET Y (REMAINDER Y 67108864))
                 (RETURN (PLUS (TIMES (QUOTIENT Y 6.7108E+7)
                                      (DIFFERENCE B A ))
                               A))))
```

Najmanja jedinica programa su tokeni, koji se dijele na: delimitere (`(`, `)`, `[`, `]` i prelazak u novi red), brojeve (nekoliko vrsta), jednostavne stringove (npr. `#AB(C)D#`, počinje i završava sa `#`, a znak `'` omogućuje ubacivanje posebnih znakova u stringove, npr `#'##` ili `#ISN''T#`), identifikatore i operatore. Stringovi se mogu konvertirati u identifikatore, npr `%#A(B)C#` je identifikator koji sadrži zagrade.

Opća forma podatka je S-izraz, izgrađen od atoma koji mogu biti brojevi, stringovi, identifikatori, specifikatori funkcija i matrice.

S-izrazi su definirani kao i u Lispu 1.5, uključujući i liste. Definirane su osnovne funkcije `CAR`, `CDR` i `CONS`.

IL programi su S-izrazi i mogu biti tretirani kao podaci.

I SL programi mogu biti tretirani kao podaci, tj. stringovi - iako njihovo procesiranje nije tako pogodno. Matrice se smatraju atomima jer `CAR` i `CDR` za njih nisu definirani. Zapisuju se, primjerice, `[2 5 1]` za jednodimenzionalnu i `[[A B] [1 2]]` za dvodimenzionalnu matricu.

Druga velika promjena je uvođenje deklaracija tipova, kao i nekih novih tipova podataka, uključujući matrice i *stringove*. Najavljuje se implementacija „packed data tables“ u budućnosti. Deklariranje tipova je neobavezno.

Iako su u Lispu 2 osnovni podaci S-izrazi, neki od podskupova S-izraza imaju svoja posebna „imene tipova:“ **BOOLEAN** (**TRUE**, **FALSE**, **NIL** i **()**, posljednja tri su sinonim), **INTEGER**, **OCTAL**, **REAL**, **FUNCTIONAL**, **SYMBOL** (to su, neočekivano, svi S-izrazi) i **ARRAY**.

*Izrazi* se definiraju kao oznake podataka (engl. designation of a datum). Podatak je vrijednost izraza. Izrazi se sastoje od konstanti, varijabli i operacionalnih formi (engl. operational forms).

*Naredbe* su akcije koje trebaju biti provedene i razlikuju se od izraza; svi izrazi su naredbe ali nisu sve naredbe izrazi. Naredbe mogu imati jednu ili više oznaka (engl. label) na koje se može prenijeti kontrola naredbom `GO`.

*Konstanta* je podatak koji označava sam sebe i ne može promijeniti vrijednost tijekom izvršavanja programa. Konstante u SL jeziku se pišu, primjerice, `'ALPHA` i `'(L1 L2)`. U IL-u se pišu `(QUOTE ALPHA)` ili `(QUOTE (L1 L2))`.

*Varijabla* je također elementarna oznaka podatka, koja može mijenjati vrijednost tijekom izvršavanja. Svakoj varijabli je asociran niz vezivanja (eng. binding), tj. lokacija i vrijednosti.

Vezivanja se formiraju deklaracijama - koje se mogu pojaviti u blokovima, funkcijama ili na nivou nadglednika (engl. supervisor level.)

Nadalje, sa svakom varijablom je asociran tip (tip **SYMBOL** dozvoljava sve vrijednosti), način spremanja (engl. storage mode) i način prijenosa (engl. transmission mode).

Postoje tri načina spremanja: fluidni, vlastiti i leksički.

Fluidne varijable se mogu pozivati izvan programa u kojima su definirane, a leksičke ne mogu. Fluidne varijable su općenitije ali i izloženije konfliktu imena. Uglavnom se koriste za komunikaciju između odvojeno kompajliranih programa.

Konačno, vlastite varijable su dizajnirane za komunikaciju između programa.

Varijable mogu označavati podatak direktno ili indirektno. Ako varijabla označava podatak direktno, onda označava vrijednost podatka. Ako označava podatak indirektno, onda označava adresu podatka. Ta je razlika posebno važna kada se podatak prosljeđuje kao argument funkcije.

Operacijske forme (engl. operational forms) se koriste za primjenu argumenata na funkciju, poziv makro transformacije, promjena tijeka programa ili lociranje elemenata u matrici.

Operacijska forma je u SL-u pisana

$$f(e1, e_2, \ldots, e_n)$$

a u IL-u

$$(f\ e_1\ e_2\ \ldots\ e_n).$$

gdje je $f$ operator a $e_i$ operandi. Ako je operator funkcija onda se operandi prvo izračunavaju a nakon toga je funkcija primjenjena na dobivene vrijednosti.

Operator može biti i matrica, a onda se operandi izračunavaju u indekse matrice. Matrica se ponaša slično.

*Funkcija* se asocira sa tipom vrijednosti i skupom tipova argumenata. Ako argumenti nisu pravog tipa, pokušat će se konverzija.

Redoslijed evaluacije argumenata nije garantiran. Prednost je u mogućoj optimizaciji, pogotovo u vezi sa razvojem paralelnih računala. No, primjeni li se operator **ORDER** na operacijsku formu, argumenti će se izračunavati redom pojavljivanja. (Nedostaje li **DISORDER** - K.M.)

Dozvoljeni su *makroi* u IL-u, i oni se izračunavaju jednako kao Hartovi makroi. Postoje predefinirani makroi, a i programer može definirati vlastite.

Ostali izrazi. U SL-u se koriste prefix i infix operatori na način sličan kao uobičajenoj algebarskoj notaciji. Primjerice, mogući su izrazi poput

$$a < b < c$$

gdje su *a*, *b* i *c* bilo koji izrazi SL.

*Uvjetni izrazi* u SL-u i IL imaju oblik

**IF** $p_1$ **THEN** $e_1$ **ELSE IF** $p_2$ **THEN** $e_2$ **ELSE** ... **IF** $p_n$ **THEN** $e_n$
**ELSE** $e_{n+1}$

**(IF** $p_1$ $e_1$ $p_2$ $e_2$ ... $p_n$ $e_n$ $e_{n+1}$**)**.

Način izračunavanja uvjetnih izraza je očigledan.

*Blok naredba* u SL i IL imaju oblik

**BEGIN** $d_1$**;** $d_2$**;** ... **;** $d_k$**;** $s_1$**;** $s_2$**;** ... **;** $s_n$ **END**

**(BLOCK (**$d_1$ $d_2$ ... $d_k$**)** $s_1$ $s_2$ ... $s_n$**)**

gdje su *di* deklaracije internih varijabli, vezanih dok je blok aktivan a *si* su naredbe. Deklaracije nisu obavezne i ako ih nema tada se blok-izraz naziva složeni izraz.

Izvršavanje blok-izraza završava ili naredbom **RETURN** ili izvršavanjem zadnje naredbe u bloku-izrazu.

Deklaracije internih varijabli u SL imaju oblik

$$p_1\ p_2\ p_3\ s_1, s_2, \ldots, s_n$$

gdje su $p_1$, $p_2$ i $p_3$ tip, način spremanja i način transmisije u bilo kojem redu, a $s_1$, ..., $s_n$ su imena varijabli ili naredbe za pridruživanje oblika $v \leftarrow e$. Deklaracije internih varijabli u SL imaju oblik

$$(v\ p_1\ p_2\ p_3\ p_4)$$

gdje su $p_1$, $p_2$, $p_3$ i $p_4$ tip, način spremanja i način transmisije te inicijalna vrijednost varijable, a sve te vrijednosti se mogu izostaviti i tada se koriste zadane (engl. default) vrijednosti.

Case-izrazi u SL i IL imaju oblik

**CASE**$(s,e_1,e_2,\ldots,e_n)$

**(CASE** $s$ $e_1$ $e_2$ ... $e_n$**)**.

Vrijednost gornjeg izraza je vrijednost $e_i$, gdje je $i$ vrijednost izraza $s$.

Naredbe za pridruživanje (assignement operator) u SL i IL imaju oblik

$v \leftarrow e$

**(SET** $v$ $e$**)**.

Prvo se izračunava $e$ i dobivena vrijednost se pridružuje $v$.

Opći oblik naredbe za pridruživanje dozvoljava da se na lijevoj strani nalazi „lokativni izraz“. Primjerice,

```
A ← '(MARY DOE)
CAR A ← 'JOHN
```

pridružuje varijabli **A** vrijednost **(JOHN DOE)**.

For-naredba u SL ima oblik

**FOR** *v* **IN** *x* **DO** *s*

Može se dodati i ograničenje

**UNLESS** *b*,

gdje je *b* logički izraz.