Reviziita Raporto pri la Algoritma Lingvo ALGOL 60

dediĉita al la memoro pri William Turanski

de

J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, M. Woodger

redaktis Peter Naur

El la angla originalo tradukis Sergio Pokrovskij[1]

1 Unue tradukite en la jaro 1983-a. Reviziite en la jaro 2018-a.

Resumo

La raporto donas plenan difinan priskribon de la internacia algoritma lingvo Algolo 60. Tiu estas lingvo konvena por esprimi grandan klason da nombraj kalkulprocedoj sufiĉe koncize por rekta aŭtomata traduko en la lingvon de priprogrameblaj aŭtomataj komputiloj.

La Enkonduko resumas la preparan laboron, sekvigintan la finan konferencon, kie la lingvo estis difinita. Krome, la konceptoj de referenca lingvo, porpublikiga lingvo kaj maŝina prezento estas klarigataj.

La unua ĉapitro donas skizon de la bazaj konsistaj partoj kaj rimedoj de la lingvo, kaj klarigas *la formalan simbolaron* per kiu la sintaksa strukturo estas difinata.

La dua ĉapitro listigas ĉiujn bazajn simbolojn kaj difinas la sintaksajn unuaĵojn *nomo, nombro kaj signoĉeno*. Poste sekvas difinoj de kelkaj gravaj konceptoj, kiaj *grando* kaj *valoro*.

La tria ĉapitro klarigas la regulojn por formi *esprimojn* kaj la signifon de tiuj esprimoj. Estas tri tipoj de esprimoj: aritmetika, logika kaj marka.

La kvara ĉapitro priskribas agajn frazojn de la lingvo nomatajn *ordonoj*. La bazaj ordonoj estas; valoriza ordono (kalkulo laŭ formulo), saltordono (malimplica rompo de la laŭvica plenumo de ordonoj), malplena ordono kaj procedura ordono (vokas por plenumiĝo fermitan procezon difinitan per deklaro de proceduro). Estas klarigata la formado de malsimplaj strukturoj havantaj karakteron de ordonoj. Tiuj inkludas: kondiĉan ordonon, iteracion, opan ordonon kaj blokon.

La kvina ĉapitro difinas la frazojn nomatajn *deklaroj*, kiuj servas por difini nevariajn ecojn de la entoj aperontaj en procezo priskribata helpe de la lingvo.

La raporto finiĝas per du detalaj ekzemploj pri uzo de la lingvo kaj alfabeta indekso de difinoj.

Enkonduko

Historia skizo

Post la publikigo de prova raporto[1,2] pri la algoritma lingvo Algolo tia kia ĝi estis preparita ĉe la Zurika konferenco en la 1958-a jaro, ekestis granda interesiĝo pri tiu lingvo.

Post neoficiala kunveno en Majenco en novembro 1958, proksimume 40 interesitoj partoprenis konferencon pri Algolo en Kopenhago en februaro 1959. Rezulte kreiĝis teknika grupo por kuna laboro ĝis la nivelo de trubenda kodo. Tiu konferenco ankaŭ kondukis al ekpublikigo de *Algol-Bulteno* redaktata de Peter Naur. La bulteno servis kiel forumo por plua diskutado. Dum la Pariza konferenco ICIP en junio 1959 oni aranĝis kelkajn renkontiĝojn, oficialajn kaj neoficialajn. Kvankam tiuj evidentigis kelkajn miskomprenojn koncerne la celon de la grupo kiu ĉefe responsis pri la difino de la lingvo, tamen estis klare ke ĝia laboro estas alte taksata. Oni decidis okazigi internacian kunvenon en januaro 1960 por plibonigi la lingvon Algolo kaj prepari la definitivan raporton. En novembro 1959 en Parizo okazis Eŭropa konferenco pri Algolo, kiun partoprenis 50 homoj. Tiam sep Eŭropaj delegitoj estis elektitaj por partopreni la konferencon en januaro 1960. Ili reprezentis la jenajn organizaĵojn: *Association française de Calcul, British Computer Society, Gesellschaft für Angevandte Mathemalik und Mechanik*, kaj nederlandan *Rekenmachine Genootschap.*

Dume en Usono, ĉiu deziranta ŝanĝi aŭ korekti la lingvon povis sendi siajn rimarkojn al la revuo «Komunikaĵoj de ACM» (*Communications of the ACM*), kie ili estis publikigataj. Tiuj rimarkoj poste formis la bazon por konsidero pri ŝanĝoj en Algolo. La organizaĵoj SHARE kaj USE kreis laborgrupojn pri Algolo, kaj ambaŭ organizaĵoj estis reprezentataj en la Komitato pri programlingvoj de ACM. Tiu komitato kunvenis en Vaŝingtono en novembro 1959 kaj ekzamenis ĉiujn rimarkojn pri Algolo senditajn al la «Komunikaĵoj de ACM». Krome, sep delegitoj estis elektitaj por partopreni la internacian konferencon en januaro 1960. Fina prepara renkontiĝo de la sep delegitoj okazis en Bostono en decembro 1959.

1, Preliminary Report — International Algebraic Language. *Comm. Assoc. Comp. Mach.* 1, № 12 (1958), 8;

2 Report on the Algorithmic Language Algol by the ACM Committee on Program Languages and the GAMM Committee on programming, edited by A. J. Perlis and K. Samelson. *Numer. Math.* Bd. 1, S. 41–60 (1959).

Konferenco en januaro 1960

La 13 delegitoj[3] el Anglujo. Francujo, Germanujo, Nederlando, Svisujo kaj Usono kunsidis en Parizo de 11 ĝis 16 Januaro 1960. Antaŭ la konferenco Peter Naur verkis tute novan proponon de raporto sin gvidante per la Prova raporto kaj la rekomendoj de la preparaj renkontiĝoj. La Pariza konferenco akceptis tiun novan formon kiel bazon por sia definitiva raporto kaj procedis al la ekzameno de ĉiu parto de la raporto. Ĉi tiu raporto estas la kunaĵo de ideoj de la komitatanoj kaj la komunaĵo de iliaj akordoj.

Konferenco en aprilo 1962

(Redaktis M. Woodger)

Kunveno de iuj el la aŭtoroj de Algolo 60 okazis de 2 ĝis 3 aprilo 1962 en Romo, danke al la rimedoj kaj gastamo de la Internacia Komputa Centro. Ĉeestis:

Aŭtoroj

> F. L. Bauer, J. Green, G. Katz, R. Kogon (reprezentis J.W. Backus-on), P. Naur, K. Samelson, J. H. Wegstein, A. van Wijngaarden, M. Woodger;

Konsilantoj

> M. Paul, R. Franciotti, P. Z. Ingerman, G. Seegmüller, R. E. Utman, P. Landin;

Observanto

> W. L. van der Poel (Prezidanto de la laborgrupo pri Algolo ĉe IFIP TC 2.1).

La celo de la kunveno estis korekti la konatajn erarojn, forigi evidentajn plursencaĵojn kaj alimaniere pliklarigi la Raporton pri Algolo 60. Oni ne konsideris etendojn dum tiu kunveno. Diversaj proponoj pri korekto kaj pliklarigo, kiujn la interesitaj grupoj prezentis responde al la demandaro el Algol-Bulteno n-ro 14, servis kiel gvido.

Ĉi tiu raporto[4] estas aldono al la «Raporto pri Algolo 60» kaj celas solvi kelkajn el ties problemoj. Ne ĉiujn demandojn pri la originala raporto oni sukcesis solvi. Konsiderante, ke hastaj konkludoj pri kelkaj subtilaj punktoj povus krei novajn plursencaĵojn, la komitato decidis raporti nur pri la punktoj kiujn laŭ ĉies opinio oni povis formuli klare kaj unusence.

---

3 W. Turanski el la usona grupo pereis en akcidento ĵus antaŭ la konferenco.

4 *Editora noto.* La angla originalo de la «Reviziita raporto» estas teksto aprobita de la Konsilio de IFIP. Ĝi estas la teksto akceptita en la Januara konferenco de 1960ª jaro, kun la ŝanĝoj aprobitaj en la Aprila konferenco de la jaro 1962ª; pri ĉi tiuj ŝanĝoj temas la komentata loko; ili estas faritaj en la teksto sekvanta la Enkondukon.

En la espero ke kurantaj laboroj pri novaj programlingvoj donos pli bonajn solvojn, la demandoj koncernantaj la jenajn temojn estis lasitaj por plua konsidero fare de la laborgrupo 2.1 ĉe IFIP:

1. Kromefikoj de funkcioj
2. La koncepto de transigo per esprimo
3. *Own* : ĉu statika aŭ dinamika
4. Iteracio: ĉu statika aŭ dinamika
5. Malakordo inter specifo kaj deklaro.

La aŭtoroj de la raporto pri Algolo 60 kiuj partoprenis la Roman konferencon, konstatante la estiĝon de laborgrupo pri Algolo ĉe IFIP, akceptis ke ĉia kolektiva responso kiun ili povus havi pri la evoluigo, specifado kaj plibonigo de la lingvo Algolo de post tiam transiĝu al tiu grupo.

Ĉi tiun raporton reviziis «IFIP TC2 on Programming Languages» en aŭgusto 1962 kaj aprobis la «Council of the International Federation for Information Processing».

Kiel en la prova raporto pri Algolo, oni distingas tri malsamajn nivelojn, nome la referencan lingvon, la porpublikigan lingvon kaj diversajn maŝinajn prezentojn.

Referenca lingvo

1. Ĝi estas la labora lingvo de la Komitato.
2. Ĝi estas la difina lingvo.
3. Ĝiaj signoj estas elektitaj por plifaciligi interkomprenon, sendepende de la komputilaj limigoj aŭ simbolaroj uzataj de programistoj aŭ en pura matematiko.
4. Ĝi estas la ĉefa referenco kaj gvidilo por la konstruantoj de tradukiloj.
5. Ĝi estas la gvidilo por ĉiuj maŝinaj prezentoj.
6. Ĝi estas la gvidilo por la aliliterigo el referenca lingvo en ĉiun lokan maŝinan prezenton.
7. La ĉefaj publikaĵoj pri la lingvo mem uzos la referencan prezenton.

Porpublikiga lingvo

1. La porpublikiga lingvo aperas kiel variaĵo de la referenca lingvo, konforma al kutimoj de presado kaj matematika manskribado (ekz. subaj indicoj, spacetoj, eksponentoj, grekaj literoj).
2. Ĝi estas uzata por formulado kaj komunikado.
3. La signoj uzataj en diversaj landoj povas diferenci, tamen oni devas garantii unusignifan interrespondon kun la referenca prezento.

Maŝinaj prezentoj

1. Ĉiu el ili estas variaĵo de la referenca lingvo, reduktita konforme al la

aro da signoj de la disponebla enigilo.
2. Ĉiu el ili uzas la signaron de aparta komputilo kaj estas la lingvo akceptata de tradukilo por tiu komputilo.
3. Ĉiu el ili devas esti garnita per speciala aro da reguloj por traduki el la porpublikiga aŭ referenca lingvo.

Koncerne al la traduko inter la referenca lingvo kaj lingvo konvena por publikigaĵoj, la jenaj reguloj, inter aliaj, estas rekomendataj:

Referenca lingvo		*Porpublikiga lingvo*
indicaj krampoj	[ ]	mallevu la linion inter la krampoj, forigu la krampojn
eksponento	↑	levu la eksponenton
rondaj krampoj	( )	iu ajn speco de krampoj: rondaj, rektaj, kunigaj
bazo dek	$_{10}$	levu la dekon kaj la sekvantan entjeron, intermetu la subkomprenatan signon de multipliko.

Priskribo de la Referenca Lingvo

> Kio estas iel direbla, estas direbla klare; kaj pri kio oni ne povas paroli, pri tio oni silentu.
>
> *Ludwig Wittgenstein*

1. Strukturo de la lingvo

Kiel oni vidis en la enkonduko, la algoritma lingvo havas tri diversajn formojn de prezento (referenca, maŝina kaj porpublikiga) kaj la plua priskribo uzas [nur] la rimedojn de la referenca prezento. Tio signifas, ke ĉiujn objektojn difinatajn en la lingvo oni prezentas helpe de fiksita signaro, — kaj sole per la elekto de la signoj povas diferenci la du aliaj prezentoj. La strukturo kaj la enhavo devas esti identaj por ĉiuj prezentoj.

La algoritma lingvo estas destinita por priskribo de komputaj procezoj. La baza koncepto uzata por priskribi regulojn de kalkulado estas la bone konata aritmetika esprimo, kiu havas kiel erojn nombrojn, variablojn kaj funkciojn. El tiaj esprimoj konstruiĝas, laŭ reguloj de aritmetika kunmeto, memstaraj frazoj de la lingvo — malimplicaj formuloj, nomataj *valorizaj ordonoj.*

Por indiki la fluon de komputaj procezoj, estas aldonitaj nearitmetikaj ordonoj kaj kondiĉoj por ordonoj. Ili ebligas priskribon de (ekz.) alternativoj aŭ ciklaj ripetoj de komputaj ordonoj. Ĉar la funkciado de tiuj ordonoj postulas, ke unu ordono referencu alian, ordonojn eblas marki. Vico da ordonoj povas esti enfermita inter la ordonajn krampojn **begin** kaj **end** formante opan ordonon.

La ordonojn apogas deklaroj. Tiuj ne estas instrukcioj por komputado, sed anstataŭe ili informas la tradukilon pri la ekzisto kaj certaj ecoj de objektoj referencataj el ordonoj, kiaj la speco de nombroj kiujn variablo povas havi kiel valorojn, la dimensio de nombra tabelo, aŭ eĉ la aro da reguloj difinantaj funkcion. Vico da deklaroj kaj postiranta vico da ordonoj enfermite inter **begin** kaj **end** konsistigas *blokon*. Ĉiu deklaro aperas tiel en bloko kaj validas nur por tiu bloko.

Programo estas bloko aŭ opa ordono ne entenata de alia ordono kaj ne uzanta aliajn ne entenatajn en ĝi ordonojn.

Malsupre sekvas la priskribo de sintakso kaj semantiko.[5]

5 Ĉiufoje kiam oni asertas, ke la aritmetika precizo ĝenerale ne estas specifita, aŭ kiam la finiĝon de procezo oni lasas aŭ anoncas nedifinita, tio estu komprenata jene: la programo plene difinas komputan procezon nur akompanate de instrukcio indikanta la uzendajn precizon, specon de aritmetiko kaj la agojn farendajn en ĉiuj okazoj, kiuj povas estiĝi dum la plenumo de la komputado.

1.1. Formala sistemo por sintaksa priskribo

La sintakso estos priskribata per metalingvistikaj formuloj[6] (B9). Ilian interpretadon plej bone klarigas ekzemplo:

‹ab› ::= (| [| ‹ab› (| ‹ab› ‹d›

Vicoj da signoj enfermitaj inter angulaj krampoj prezentas metalingvistikajn variablojn kies valoroj estas vicoj da simboloj. La metasimboloj ::= kaj | (ĉilasta signifas *aŭ*) estas metalingvistikaj ligiloj. Ĉiu formulero, kiu ne estas variablo aŭ ligilo, prezentas sin mem (aŭ la aron da al ĝi similaj simboloj).

Apudmeto de simboloj kaj (aŭ) variabloj en formulo signifas apudmeton de la prezentataj vicoj. Do, la supra formulo donas rekursian regulon por formi valorojn de la variablo ‹ab›. Ĝi indikas, ke (kaj [estas taŭgaj valoroj por ‹ab›, kaj ke el ĉiu tia valoro oni povas formi pluajn aldonante post ĝin la signon (aŭ iun ajn valoron de la variablo ‹d›. Se la valoroj de ‹d› estas dekumaj ciferoj, jen estas kelkaj valoroj por ‹ab›:

[((1(37((((
(12345([86

Por plifaciligi la studon, la simboloj elektitaj por distingi la metalingvistikajn variablojn (t.e. signovicoj inter la angulaj krampoj, kia ab en la antaŭa ekzemplo) estas vortoj, proksimume priskribantaj la sencon de la responda variablo. Kiam tiel aperintaj vortoj estas uzataj aliloke en la teksto, ili referencas la respondan sintaksan formulon. Krome, kelkaj formuloj estas donitaj en pli ol unu loko.

Difino:

‹malpleno› ::=

(t.e. nula signoĉeno).

2. Bazaj simboloj, nomoj, nombroj kaj signoĉenoj. Bazaj konceptoj

La referenca lingvo konstruiĝas el ĉi tiaj bazaj simboloj:

‹baza simbolo› ::= ‹litero› | ‹cifero› | ‹logika valoro› | ‹limaĵo›

2.1. Literoj

‹litero› ::= a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|
A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z

Oni povas arbitre redukti tiun alfabeton, aŭ aldoni al ĝi ajnan nemisklaseblan signon (t.e. signon diferencan de ĉiu cifero, logika valoro aŭ limaĵo).

6 J. W. Backus. The syntax and semantics of the proposed international algebraic language of the Zürich ACM-CAMM conference. ICIP Paris, June 1959.

Literoj ne havas propran signifon. Ili estas uzataj por formi nomojn kaj signoĉenojn (v. 2.4, «Nomoj»; 2.6, «Signoĉenoj»).[7]

2.2. Ciferoj kaj logikaj valoroj

2.2.1. Ciferoj

‹cifero› ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Oni uzas ciferojn por formi nombrojn, nomojn kaj signoĉenojn.

2.2.2. Logikaj valoroj

‹logika valoro› ::= **true** | **false**

La logikaj valoroj havas evidentan signifon (*vera* — *malvera*).

2.3. Limaĵoj

‹limaĵo› ::= ‹operacisimbolo› | ‹intersimbolo› | ‹krampo› | ‹deklarilo› |
‹specifilo›

‹operacisimbolo› ::= ‹aritmetika operacisimbolo› | ‹rilatosimbolo› |
‹logika operacisimbolo› | ‹plenumorda operacisimbolo›

‹aritmetika operacisimbolo› ::= + | − | × | / | ÷ | ↑

‹rilatosimbolo ::= < | ≤ | = | ≥ | > | ≠

‹logika operacisimbolo› ::= ≡ | ⊃ | ∨ | ∧ | ¬

‹plenumorda operacisimbolo› ::= **go to** | **if** | **then** | **else** | **for** | **do**[8]

‹intersimbolo› ::= , | . | ₁₀ | : | ; | := | ␣ | **step** | **until** | **while** | **comment**

‹krampo› ::= (|) | [|] | ‘ | ’ | **begin** | **end**

‹deklarilo› ::= **own** | **Boolean** | **integer** | **real** | **array** | **switch** | **procedure**

‹specifilo› ::= **string** | **label** | **value**

La limaĵoj havas fiksitan signifon. Por la plimulto el ili ĝi estas evidenta[9], la ceteraj estos klarigitaj poste en la konvena loko.

Presaj aranĝoj, kiaj spacetoj inter vortoj aŭ ŝanĝo de linio, ne havas signifon en la referenca lingvo. Tamen ili estas uzeblaj por plifaciligi la legadon de programo.

---

7 Menciendas, ke ĉie en la referenca lingvo la ~~substrekado~~ grasigo servas por apartigi memstarajn bazajn simbolojn (v. 2.2.2 kaj 2.3). Ĉi-lastaj neniel rilatas al la unuopaj literoj el kiuj ili estas kunmetitaj. En ĉi tiu Raporto la ~~substrekado~~ grasigo ne estas uzata por alia celo.

8 **do** aperas en iteracioj. Ĝi neniel rilatas al la **do** el la prova raporto, kia **do** ne estas parto de Algolo 60.

9 Evidenta por anglalingvano. Tradukon en Esperanton vidu en la Indekso. (Noto de la tradukinto.)

Por ebligi aldonon de rimarkoj en programon, validas la jenaj konvencioj pri *komentoj:*

La vico da bazaj simboloj:	*estas samsignifa kun:*
; comment ‹ajna vico ne entenanta **;** ›	**;**
begin comment ‹ajna vico ne entenanta **;** ›	**begin**
end ‹ajna vico entenanta nek **end** nek **;** nek **else** ›	**end**

Tiu tabelo de samsignifeco indikas, ke iu ajn el la tri strukturoj de la maldekstra kolumno, se ĝi aperas ekster signoĉeno, estas anstataŭigebla je la simbolo el la dekstra kolumno en la sama linio. Tia anstataŭigo neniel efikas la funkciadon de la programo. Kompreneble, la komentostrukturon renkontitan pli frue dum la legado dekstren oni anstataŭigu antaŭ ol la postajn.

2.4. Nomoj

2.4.1. Sintakso

‹nomo› ::= ‹litero› | ‹nomo› ‹litero› | ‹nomo› ‹cifero›

2.4.2. Ekzemploj

q	V17a	Marinjo
supo	a34kTMNs	

2.4.3. Semantiko

Nomoj ne havas propran signifon. Ili servas por indiki simplajn variablojn, tabelojn, markojn, komutilojn kaj procedurojn. Oni povas arbitre elekti tiajn nomojn (tamen v. 3.2.4, «Antaŭdifinitaj funkcioj»).

Unu saman nomon ne eblas uzi por prezenti du malsamajn grandojn — krom se la videblejoj de tiuj grandoj, difinitaj per la deklaroj en la programo, havas malplenan komunaĵon (v. 2.7, «Grandoj, specoj kaj videblejoj» kaj 5, «Deklaroj»).

2.5. Nombroj

2.5.1. Sintakso

‹sensignuma entjero› ::= ‹cifero› | ‹sensignuma entjero› ‹cifero›

‹entjero› ::= ‹sensignuma entjero› | + ‹sensignuma entjero› |
    − ‹sensignuma entjero›

‹dekuma frakcio› ::= . ‹sensignuma entjero›

‹skalfaktoro› ::= $_{10}$ ‹entjero›

‹dekuma nombro› ::= ‹sensignuma entjero› | ‹dekuma frakcio› |
    ‹sensignuma entjero› ‹dekuma frakcio›

‹sensignuma nombro› ::= ‹dekuma nombro› | ‹skalfaktoro› |
‹dekuma nombro› ‹skalfaktoro›

‹nombro› ::= ‹sensignuma nombro› | + ‹sensignuma nombro› |
− ‹sensignuma nombro›

2.5.2. Ekzemploj

0	−200.084	$-.083_{10}-02$
177	$+07.43_{10}8$	$-1_{10}7$
.5384	$9.34_{10}+10$	$_{10}-4$
+0.7300	$2_{10}-4$	$+1_{10}+5$

2.5.3. Semantiko

Dekumaj nombroj havas sian kutiman signifon. La skalfaktoro esprimiĝas kiel entjera potenco de 10.

2.5.4. Tipoj

Entjeroj havas la tipon **integer** . Ĉiuj aliaj nombroj havas la tipon **real** (v. 5.1, «Deklaro pertipa»).

2.6. Signoĉenoj

2.6.1. Sintakso

‹simpla ĉeno› ::= ‹ajna vico da bazaj simboloj entenanta nek ‘ nek ’› |
‹malpleno›

‹malfermita ĉeno› ::= ‹simpla ĉeno› | ‘ ‹simpla ĉeno› ’ | ‹malfermita ĉeno›
‹malfermita ĉeno›

‹signoĉeno› ::= ‘ ‹malfermita ĉeno› ’

2.6.2. Ekzemploj

‘5k,,−‘[((‘∧=/:’Tt’’
‘..Tio␣estas␣‘signocheno’’

2.6.3. Semantiko

La citiloj de signoĉeno estas provizitaj por igi la lingvon kapabla manipuli arbitrajn vicojn da bazaj simboloj. La simbolo ␣ prezentas spaceton. Ekster signoĉeno ĝi estas sensignifa.

Signoĉeno povas esti fakta parametro de proceduro (v. 3.2. «Indiko de funkcio» kaj 4.7, «Procedura ordono»).

2.7. Grandoj, specoj kaj videblejoj

Oni distingas la jenajn specojn de *grandoj*[10] simplaj variabloj, tabeloj, markoj, komutiloj, proceduroj.

La videblejo de grando estas la aro da ordonoj kaj esprimoj en kiuj validas la deklaro de la nomo kun ĝi asociita. Pri markoj vidu 4.1.3.

2.8. Valoroj kaj tipoj

Valoro estas ordigita aro da nombroj (aparta okazo: unuopa nombro), ordigita aro da logikaj valoroj (aparta okazo: unuopa logika valoro), aŭ marko.

Pri iuj sintaksaj unuaĵoj oni diras, ke ili posedas valorojn. Ĝenerale la valoroj ŝanĝiĝas dum la plenumo de la programo. La valoroj de esprimoj kaj ties konsistigaj partoj estas difinitaj en la ĉapitro 3. La valoro de nomo de tabelo estas la ordigita aro da valoroj de la responda tabelo el indicitaj variabloj (v. 3.1.4.1).

La diversaj *tipoj* (**integer**, **real**, **Boolean**) esence indikas ecojn de valoroj. La tipoj asociitaj kun sintaksaj unuaĵoj rilatas al ties valoroj.

3. Esprimoj

En la lingvo, la bazaj partoj de programo priskribantaj algoritmajn procedojn estas aritmetikaj, logikaj kaj markaj esprimoj. Krom iuj limaĵoj, eroj de tiuj esprimoj estas logikaj valoroj, nombroj, variabloj, indikoj de funkcio, elementaj aritmetikaj, rilataj, logikaj kaj plenumordaj operacisimboloj. Ĉar la sintaksa difino pri variablo kaj indiko de funkcio entenas esprimojn, la difino pri esprimo kaj ties konsistaj partoj estas esence rekursia.

‹esprimo› ::= ‹aritmetika esprimo› | ‹logika esprimo› | ‹marka esprimo›

3.1. Variabloj

3.1.1. Sintakso

‹nomo de variablo› ::= ‹nomo›

‹simpla variablo› ::= ‹nomo de variablo›

‹indica esprimo› ::= ‹aritmetika esprimo›

‹indicaro› ::= ‹indica esprimo› | ‹indicaro› , ‹indica esprimo›

‹nomo de tabelo› ::= ‹nomo›

‹indicita variablo› ::= ‹nomo de tabelo› [‹indicaro›]

10 RR uzas la matematikan terminon *quantity* (al kiu en Esperanto respondas «grando») por la nocio, kiun en la modernaj programlingvoj oni nun nomas «objekto» (*object*). (Noto de la tradukinto.)

3.1.2. Ekzemploj

epsilo tab17 X[sin(n × pi / 2), Q[3, n, 4] detA Q[7,2]

3.1.3. Semantiko

Variablo estas indiko de unuopa valoro. Oni povas uzi tiun valoron en esprimoj por formi aliajn valorojn, kaj oni povas laŭdezire ŝanĝadi ĝin per valorizaj ordonoj (4.2). La tipo de la valoro de ĉiu variablo estas difinata en la deklaro por la variablo mem (v. 5.1, «Deklaro pertipa») aŭ por la responda nomo de tabelo (v. 5.2, «Deklaro de tabeloj»).

3.1.4. Indicoj

3.1.4.1. Indicitaj variabloj indikas valorojn kiuj estas eroj de multedimensiaj tabeloj (v. 5.2, «Deklaro de tabeloj»). Ĉiu aritmetika esprimo el la indicaro okupas unu indican pozicion de la indicita variablo, kaj estas nomata *indico*. La plena listo da indicoj enfermiĝas inter ia indicaj krampoj [ ]. La tabeleron referencatan per indicita variablo determinas la fakta nombra valoro de ties indicoj (v. 3.3, «Aritmetikaj esprimoj»).

3.1.4.2. Ĉiu indica pozicio efikas kvazaŭ variablo de la tipo **integer**, kaj oni rigardas la kalkulon de la indico egalsignifa al valorizo de tiu fikcia variablo (v. 4.2.4). La valoro de la indicita variablo estas difinita nur se la valoro de la indica esprimo kuŝas inter la indicaj limoj de la tabelo (v. 5.2, «Deklaro de tabeloj»).

3.2. Indiko de funkcio

3.2.1. Sintakso

‹procedurnomo› ::= ‹nomo›

‹fakta parametro› ::= ‹signoĉeno› | ‹esprimo› | ‹nomo de tabelo› |
‹nomo de komutilo› | ‹procedurnomo›

‹literĉeno› ::= ‹litero› | ‹literĉeno› ‹litero›

‹interparametra disigilo› ::= , |) ‹literĉeno› :(

‹listo da faktaj parametroj› ::= ‹fakta parametro› |
‹listo da faktaj parametroj› ‹interparametra disigilo› ‹fakta parametro›

‹fakta parametraro› ::= ‹malpleno› | (‹listo da faktaj parametroj›)
‹indiko de funkcio› ::= ‹nomo de funkcio› ‹fakta parametraro›

3.2.2. Ekzemploj

sin (a − b) S(s − 5)Temperaturo:(T)Premo:(P))
J(v + s, n) Traduku (‘:=’)Stako:(0)
R

3.2.3. Semantiko

Indiko de funkcio difinas unuopan nombran aŭ logikan valoron rezultontan el la apliko de determinita aro da reguloj, kiujn difinas deklaro de proceduro (v. 5.4, «Deklaro de proceduro») al fiksita aro da faktaj parametroj. La regulojn por determini la faktajn parametrojn vidu en 4.7, «Procedura ordono». Ne ĉiu deklaro de proceduro difinas valoron por indiko de funkcio.

3.2.4. Antaŭdifinitaj funkcioj

Endas rezervi kelkajn nomojn por la tradiciaj funkcioj de analitiko, kiuj havos la formon de proceduroj. Oni rekomendas la jenan liston:

abs($\mathcal{E}$)	por la modulo (absoluta valoro) de la valoro de la esprimo $\mathcal{E}$
sign($\mathcal{E}$)	por la signumo de la valoro de $\mathcal{E}$ ($+1$ por $\mathcal{E} > 0$, 0 por $\mathcal{E} = 0$, -1 por $\mathcal{E} < 0$)
sqrt($\mathcal{E}$)	por la kvadrata radiko de la valoro de $\mathcal{E}$
sin($\mathcal{E}$)	por la sinuso de la valoro de $\mathcal{E}$
cos ($\mathcal{E}$)	por la kosinuso de la valoro de $\mathcal{E}$
arctan($\mathcal{E}$)	por la ĉefa valoro de la arktangento de la valoro de $\mathcal{E}$
ln($\mathcal{E}$)	por la natura logaritmo de la valoro de $\mathcal{E}$
exp($\mathcal{E}$)	por la eksponencialo de la valoro de $\mathcal{E}$ ($e^{\mathcal{E}}$)

Tiuj funkcioj taŭgu kaj por reelaj, kaj por entjeraj argumentoj. Ili liveros valoron de la tipo **real** — krom sign($\mathcal{E}$), kiu havos valorojn de la tipo **integer**. En aparta maŝina prezento tiuj funkcioj povas esti disponeblaj sen malimplica deklaro (v. 5, «Deklaroj»).

3.2.5. Konvertaj funkcioj

Oni povas difini konvertajn funkciojn por ĉiu duopo da valoroj aŭ grandoj. Inter la antaŭdifinitaj funkcioj oni speciale rekomendas unu, nome

entier($\mathcal{E}$)

kiu «konvertas» esprimon de la reela tipo al tiu de la entjera tipo. La valoro de entier($\mathcal{E}$) estas la plej granda entjero ne superanta la valoron de $\mathcal{E}$.

3.3. Aritmetikaj esprimoj

3.3.1. Sintakso

‹adiciranga operacisimbolo› ::= + | −

‹multiplikranga operacisimbolo› ::= × | / | ÷

‹pimara esprimo› ::= ‹sensignuma nombro› | ‹variablo› |
‹indiko de funkcio› | (‹aritmetika esprimo›)

‹faktoro› ::= ‹primara esprimo› | ‹faktoro› ↑ ‹primara esprimo›

‹termo› ::= ‹faktoro› | ‹termo› ‹multiplikranga operacisimbolo› ‹faktoro›

‹simpla aritmetika esprimo› ::= ‹termo› |
‹adiciranga operacisimbolo› ‹termo› |
‹simpla aritmetika esprimo› ‹adiciranga operacisimbolo› ‹termo›

‹kondiĉo› ::= **if** ‹logika esprimo› **then**

‹aritmetika esprimo› ::= ‹simpla aritmetika esprimo› |
‹kondiĉo› ‹simpla aritmetika esprimo› **else** ‹aritmetika esprimo›

3.3.2. Ekzemploj

Primaraj esprimoj:

$7.394_{10}-8$
Sumo
w[i + 2.8]
cos(y + z × 3)
(a − 3 / y + vu ↑ 8);

Faktoroj:

omego
sumo ↑ cos(y + z × 3)
$7.394_{10}-8$ ↑ W[i + 2.8] ↑ (a − 3 / y + vu ↑ 8)

Termoj:

U
omega × sumo ↑ cos(y + z × 3) / $7.394_{10}-8$ ↑ W[i+2, 8] ↑ (a − 3/y + vu↑8)

Simpla aritmetika esprimo:

U − Yu + omego×sumo↑cos(y+z×3) / $7.394_{10}-8$↑W[i+2,8]↑(a−3/y+ vu↑8)

Aritmetikaj esprimoj:

w × u − Q(S + Cu)↑2
if q > 0 **then** S + 3 × Q / A **else** 2 × S + 3 × q
if a < 0 **then** U+V **else if** a×b > 17 **then** U/V **else if** k ≠ y **then** U/V **else** 0
a × sin(omego × t)
$0.57_{10}12$ × a[N × (N − 1) / 2, 0]
(A × arctan(y) + Z) ↑ (7 + Q)
if q **then** n − 1 **else** n
if a < 0 **then** A / B **else if** b = 0 **then** B / A **else** z

3.3.3. Semantiko

Aritmetika esprimo estas regulo por kalkuli nombran valoron. Por simplaj aritmetikaj esprimoj, tiun valoron oni ricevas plenumante la indikitajn aritmetikajn operaciojn super la faktaj nombraj valoroj de la konsistigaj primaraj esprimoj (v. la detalan klarigon en 3.4.4 malsupre). La fakta nombra

valoro de primara esprimo estas evidenta en la okazo de nombroj. Por variabloj ĝi estas la kuranta valoro (tiu de la lasta valorizo) kaj por indikoj de funkcio ĝi estas la valoro kiun determinas la reguloj de kalkulo difmantaj la proceduron (v. 5.4.4, «Valoro de indiko de funkcio») por la kurantaj valoroj de la proceduraj parametroj donitaj en la esprimo. Fine, por interkrampigita aritmetika esprimo necesas, uzante rekursian analizon, esprimi ĝian valoron ekirante de la valoroj de la primaraj esprimoj, apartenantaj al la tri aliaj specoj.

En pli ĝeneralaj aritmetikaj esprimoj, kiuj entenas kondiĉojn, unu sola el kelkaj aritmetikaj esprimoj estas elektata laŭ la faktaj valoroj de la logikaj esprimoj (v. 3.4, «Logikaj esprimoj»). Tiu elekto fariĝas jene: la logikaj esprimoj de la kondiĉoj estas prikalkulataj laŭvice dekstren, ĝis troviĝos iu havanta la valoron **true**. Tiam la valoro de la aritmetika esprimo estas la valoro de la unua aritmetika esprimo kiu sekvas tiun ĉi logikan esprimon (t.e. de la plej longa aritmetika esprimo trovebla en tiu pozicio). La konstruaĵo

> **else** ‹simpla aritmetika esprimo›

estas egalsignifa al

> **else if true then** ‹simpla aritmetika esprimo›

3.3.4. Operacisimboloj kaj tipoj

Escepte la logikajn esprimojn en kondiĉoj, la konsistaj partoj de simplaj aritmetikaj esprimoj devas havi la tipon **real** aŭ **integer** (v. 5.1, «Deklaro pertipa»). La signifon de la bazaj operacisimboloj kaj la tipon de la esprimoj kiujn ili formas determinas jenaj reguloj:

3.3.4.1. La operacisimboloj +, − kaj × havas la kutiman signifon (adicio, subtraho kaj multipliko). La tipo de la esprimo estos **integer** se ambaŭ argumentoj havas tiun tipon, aliokaze ĝi estos **real**.

3.3.4.2. La operacioj „‹termo› / ‹faktoro›“ kaj „‹termo› ÷ ‹faktoro›“ prezentas dividon. Tiun ĉi oni konceptu kiel multiplikon de la termo per la inverso de la faktoro, plenumendan observante la regulon pri rangoj de operacisimboloj (v. 3.3.5). Tial ekz.

$$a / b \times 7 / (p - q) \times v / s$$

signifas

$$((((a \times (b^{-1})) \times 7) \times ((p - q)^{-1})) \times v) \times (s^{-1})$$

La operacisimbolo / estas difinita por ĉiuj kombinaĵoj de la tipoj **real** kaj **integer**; ĉiuokaze ĝi liveras reelan rezulton. La operacisimbolo ÷ estas

difinita nur se ambaŭ argumentoj estas entjeraj, kaj liveras entjeran rezulton matematike difinitan jene:

a ÷ b = sign(a / b) × entier(abs(a / b))

(v. 3.2.4 kaj 3.2.5).

3.3.4.3. La operacio ‹faktoro› ↑ ‹primara esprimo› prezentas la potencigon, kie la faktoro estas la bazo kaj la primara esprimo estas la eksponento. Tial ekz.

2↑n↑k signifas $(2^n)^k$,

dum

2↑(n↑m) signifas $2^{(n^m)}$.

Se *i* indikas nombron de la tipo **integer**, *r* nombron de la tipo **real** kaj *a* nombron de ajna el tiuj tipoj, tiam la rezulton determinas la jenaj formuloj:

a↑*i* : SE *i* > 0, TIAM *a*×*a*×...×*a* (*i*-oble), de la sama tipo kiel *a*;
SE *i* = 0, TIAM
SE *a* ≠ 0, TIAM 1, de la sama tipo kiel *a*;
SE *a* = 0, TIAM nedifinita;
SE *i* < 0, TIAM
SE *a* ≠ 0, TIAM 1/(*a*×*a*×...×*a*)
(la dividanto havas *i* multiplikantojn), de la tipo **real**;
a↑*r* : SE *a* > 0, TIAM exp(*r* × ln(*a*)), de la tipo **real**;
SE *a* = 0, TIAM
SE *r* > 0, TIAM 0.0, de la tipo **real**
SE *r* < 0, TIAM nedifinita;
SE *a* < 0, TIAM nedifinita por ĉiuj *r*

3.3.5. Rangoj de operacioj

La operacioj en unu esprimo ĝenerale estas plenumataj de maldekstre dekstren, tamen observante la jenajn regulojn:

3.3.5.1. Konforme al la sintakso el 3.3.1, la laŭranga ordo de la operacisimboloj estas la jena:

1. ↑
2. × / ÷
3. + −

3.3.5.2. La interkrampaj esprimoj estas prikalkulataj aparte kaj iliaj valoroj estas uzataj en postaj kalkuloj. Tial oni ĉiam povas aranĝi la deziratan plenumordon per konvena intermeto de krampoj.

3.3.6. Reela aritmetiko

La sencon de nombroj kaj variabloj de la tipo **real** oni komprenu laŭ la maniero de la komputa analitiko, t.e. kiel grandojn havantajn nur limigitan nombran precizon. Same, en aritmetikaj esprimoj oni toleras certan devion disde la matematike difinita rezulto, kvankam la preciza aritmetiko ne estas specifita, kaj ĉe malsamaj maŝinaj prezentoj la valoroj de aritmetikaj esprimoj povas diferenci. Eblas reguli la eventualajn sekvojn de tiuj diferencoj per la metodoj de la komputa analitiko. Tiu regulado estu rigardata kiel parto de la priskribata procezo, do ĝi estu esprimata per la rimedoj de la lingvo mem.

3.4. Logikaj esprimoj

3.4.1. Sintakso

‹rilatosimbolo› ::= < | ≤ | = | ≥ | > | ≠

‹rilato› ::=
    ‹simpla aritmetika esprimo› ‹rilatosimbolo› ‹simpla aritmetika esprimo›

‹primara logika esprimo› ::= ‹logika valoro› | ‹variablo› |
    ‹indiko de funkcio› | ‹rilato› | (‹logika esprimo›)

‹sekundara logika esprimo› ::= ‹primara logika esprimo› |
    ¬ ‹sekundara logika esprimo›

‹logika faktoro ::= ‹sekundara logika esprimo› |
    ‹logika faktoro› ∧ ‹sekundara logika esprimo›

‹logika termo› ::= ‹logika faktoro› | ‹logika termo› ∨ ‹logika faktoro›
    ‹implikacio› ::= ‹logika termo› | ‹implikacio› ⊃ ‹logika termo›

‹simpla logika esprimo› ::= ‹simpla logika esprimo› |
    ‹implikacio› ≡ ‹simpla logika esprimo›

‹logika esprimo› ::= ‹simpla logika esprimo› |
    ‹kondiĉo› ‹simpla logika esprimo› **else** ‹logika esprimo›

3.4.2. Ekzemploj

    x = 2
    y>V ∨ z<q
    a+b > −5 ∧ z−d > q↑2
    P ∧ Q ∨ x ≠ y
    g ≡ ¬ a ∧ b ∧ ¬ c ∨ d ∨ e ⊃ ¬ f
    if k < 1 **then** s > w **else** h ≤ c
    if if if a **then** b **else** c **then** d **else** f **then** g **else** h < k

3.4.3. Semantiko

Logika esprimo estas regulo por kalkuli logikan valoron. La principoj de la kalkulado estas tute similaj al tiuj por la aritmetikaj esprimoj, donitaj en

3.3.3.

3.4.4. Tipoj

Variabloj kaj indikoj de funkcio aperantaj kiel primaraj logikaj esprimoj devas esti deklaritaj **Boolean** (v. 5.1, «Deklaro pertipa» kaj 5.4.4, «Valoro de indiko de funkcio»).

3.4.5. La operacisimboloj

Rilato liveras la valoron **true** se la responda logika nlato estas vera por la argumentaj esprimoj, aliokaze ĝi liveras la valoron **false**.

La signifon de la logikaj operacisimboloj ¬ (ne), ∧ (kaj), ∨ (aŭ), ⊃ (sekvigas) kaj ≡ (ekvivalentas) donas jena funkcia tabelo:

b1	**false**	**false**	**true**	**true**
b2	**false**	**true**	**false**	**true**
¬ b1	**true**	**true**	**false**	**false**
b1 ∧ b2	**false**	**false**	**false**	**true**
b1 ∨ b2	**false**	**true**	**true**	**true**
b1 ⊃ b2	**true**	**true**	**false**	**true**
b1 ≡ b2	**true**	**false**	**false**	**true**

3.4.6. Rangoj de operacisimboloj

La operacioj en unu esprimo ĝenerale estas plenumataj de maldekstre dekstren, tamen observante la sekvajn aldonajn regulojn.

3.4.6.1. Konforme al la sintakso de 3.4.1, la laŭranga ordo de la operacisimboloj estas jena:

1. aritmetikaj esprimoj laŭ 3.3.5
2. < ≤ = ≥ > ≠
3. ¬
4. ∧
5. ∨
6. ⊃
7. ≡

3.4.6.2. La uzo de krampoj havas la saman sencon kiel en 3.3.5.2.

3.5. Markaj esprimoj

3.5.1. Sintakso

‹marko› ::= ‹nomo› | ‹sensignuma entjero›

‹nomo de komutilo› ::= ‹nomo›

‹indiko de komutilo› ::= ‹nomo de komutilo› [‹indica esprimo›]

‹simpla marka esprimo› ::= ‹marko› | ‹indiko de komutilo› |
(‹marka esprimo›)

‹marka esprimo› ::= ‹simpla marka esprimo› |
‹kondiĉo› ‹simpla marka esprimo› **else** ‹marka esprimo›

3.5.2. Ekzemploj

17
p9
Elektu[n − 1]
Urbo[**if** y<0 **then** n **else** n + 1]
if Ab < c **then** 17 **else** q[**if** w ≤ 0 **then** 2 **else** n]

3.5.3. Semantiko

Marka esprimo estas regulo por ricevi markon de ordono (v. 4, «Ordonoj»). Ankaŭ ĉi tie la principo de kalkulo estas tute simila al tiu por aritmetikaj esprimoj (v. 3.3.3). En la ĝenerala okazo, la logikaj esprimoj de la kondiĉoj elektos simplan markan esprimon. Se ĝi estas marko, la serĉata rezulto estas trovita. Aliokaze indiko de komutilo referencos la respondan deklaron de komutilo (v. 5.3, «Deklaroj de komutilo»), kaj per la fakta nombra valoro de sia indica esprimo elektos unu el la markaj esprimoj listigitaj en la deklaro de komutilo, numerante ilin dekstren. Ĉar la tiel elektita marka esprimo povas denove estt indiko de komutilo, tiu kalkulado estas procezo rekursia.

3.5.4. Indica esprimo

La prikalkulo de indica esprimo similas tiun por indicitaj variabloj (v. 3.1.4.2). La valoro de indiko de komutilo estas difinita nur se la indica esprimo liveras unu el la pozitivaj valoroj 1, 2, 3, ..., n, kie n estas la nombro de eroj en la komuta.

3.5.5. Nombroformaj markoj

Sensignumaj entjeroj prezentantaj markojn havas tiun econ, ke la antaŭirantaj nuloj ne efikas al ilia signifo, ekz. 00217 prezentas la saman markon kiel 217.

4. Ordonoj

La frazoj de la lingvo prezentantaj agojn nomiĝas *ordonoj*. Normale ili estas plenumataj unu post alia laŭ la teksta ordo. Tamen tiun vicon de agoj povas rompi saltordonoj, kiuj malimplice difinas sian sekvonton; ĝin ankaŭ povas mallongigi kondiĉaj ordonoj, ebligantaj preterigi certajn ordonojn.

Por ke oni povu difini specialan dinamikan sekvigon, eblas provizi ordonojn je markoj.

Ĉar la ordonoj povas ariĝi formante opajn ordonojn kaj blokojn, la difino pri ordono estas principe rekursia. Krome, ĉar la deklaroj, priskribataj en la ĉapitro 5, estas esenca parto de la sintaksa strukturo, tial la sintaksa difino pri ordono devas supozi la deklarojn jam difinitaj.

4.1. Opa ordono kaj bloko

4.1.1. Sintakso

‹senmarka baza ordono› ::= ‹valoriza ordono› | ‹saltordono› |
‹malplena ordono› | ‹procedura ordono›

‹baza ordono› ::= ‹senmarka baza ordono› | ‹marko› : ‹baza ordono›

‹senkondiĉa ordono› ::= ‹baza ordono› | ‹opa ordono› | ‹bloko›

‹ordono› ::= ‹senkondiĉa ordono› | ‹kondiĉa ordono› | ‹iteracio›

‹opo kaj fermo› ::= ‹ordono› **end** | ‹ordono› ; ‹opo kaj fermo›

‹blokmalfermo› ::= **begin** ‹deklaro› | ‹blokmalfermo› ; ‹deklaro›

‹senmarka opo› ::= **begin** ‹opo kaj fermo›

‹senmarka bloko› ::= ‹blokmalfermo› ; ‹opo kaj fermo›

‹opa ordono› ::= ‹senmarka opo› | ‹marko› : ‹opa ordono›

‹bloko› ::= ‹senmarka bloko› | ‹marko› : ‹bloko›

‹programo› ::= ‹bloko› | ‹opa ordono›

Ĉi tiun sintakson oni povas prezenti jene. Se Ω, Δ kaj μ indikas arbitrajn ordonojn, deklarojn kaj markojn respektive, tiam la bazaj sintaksaj frazoj havas la formojn:

- opa ordono:
  μ: μ: ... **begin** Ω; Ω; ... Ω; Ω **end**
- bloko:
  μ: μ: ... **begin** Δ; Δ; ... Δ; Ω; Ω; ... Ω; Ω **end**

Oni atentu, ke ĉiu el la ordonoj Ω povas mem esti plena opa ordono aŭ bloko.

4.1.2. Ekzemploj

- Bazaj ordonoj:

  a := p + q
  go to Napolo
  EK: PLU: w := 7.993

- Opa ordono:

  begin
    x := 0;
    for y := 1 **step** 1 **until** n **do** x := x + A[y];
    if x > q **then go to** Halt **else if** x > w − 2 **then go to** S;
  Aw: St: W := x + svingilo
  end

- Bloko:

  Q: **begin integer** i, k; **real** w;
      for i := 1 **step** 1 **until** m **do**
        for k:=i+l **step** 1 **until** m **do**
          begin
            w := A[i, k]; A[i, k] := A[k, i]; A[k, i] := w
          end cikloj por i kaj k
  end bloko Q

4.1.3. Semantiko

Ĉiu bloko aŭtomate enkondukas novan nivelon de nomaro. Tio realiĝas jene: ĉiu nomo aperanta en la bloko povas esti specifita, per konvena deklaro (v. 5, «Deklaroj»), kiel loka por la koncerna bloko. Tio signifas ke

1. la objekto prezentata per tia nomo ene de la bloko, ekster tiu ĉi ne ekzistas, kaj ke
2. ĉiu objekto prezentata per tia nomo ekster la bloko estas neniel atingebla en ĉi-lasta.

Nomoj (krom tiuj prezentantaj markojn) aperantaj en bloko kaj en ĝi ne deklaritaj, estos nelokaj por ĝi, t.e. ili prezentos unu saman objekton kaj en la bloko, kaj je la senpere ekstera nivelo. Marko apartigita disde ordono per dupunkto, t.e. markanta la ordonon, kondutas kvazaŭ ĝi estus deklarita en la malfermo de la plej malvasta bloko ĝin entenanta, t.e. la plej malvasta bloko kies krampoj **begin** kaj **end** ĝin enfermas. Tiurilate procedurkorpo estu konsiderata kvazaŭ enkrampigita inter **begin** kaj **end** kaj traktata kiel bloko.

Ĉar ordono en bloko povas denove esti bloko, la ideojn de lokeco kaj nelokeco oni konceptu rekursie. Tial nomo, kiu estas neloka por bloko $\mathcal{A}$ povas, sed ne devas, esti neloka por la bloko $\mathcal{B}$ unu el kies ordonoj estas $\mathcal{A}$.

4.2. Valorizaj ordonoj

4.2.1. Sintakso

‹valorizoto› ::= ‹variablo› := | ‹procedurnomo› :=

‹valorizotaro› ::= ‹valorizoto› | ‹valorizotaro› ‹valorizoto›

‹valoriza ordono› ::= ‹valorizotaro› ‹aritmetika esprimo› |
‹valorizotaro› ‹logika esprimo›

4.2.2. Ekzemploj

s := q[0] := n := n + 1 + s
n := n + 1
A := B / C − v − q × S
s[v, k + 2] := 3 − arctan(s × dzeto)
W := Q > Y ∧ Z

4.2.3. Semantiko

Valorizaj ordonoj servas por doni la valoron de esprimo al unu aŭ pluraj variabloj aŭ procedurnomoj. Procedurnomon oni povas valorizi nur en la korpo de proceduro difinanta valoron por indiko de funkcio (v. 5.4.4). En la ĝenerala okazo la procezo estas tripaŝa:

4.2.3.1. Ĉiuj indicaj esprimoj aperantaj en la variabloj estas prikalkulataj laŭvice dekstren.

4.2.3.2. La *esprimo* de la ordono estas prikalkulata.

4.2.3.3. La valoro de la esprimo estas donata al ĉiuj valorizotoj uzante por la indicoj la valorojn kalkulitajn je la paŝo 4.2.3.1.

4.2.4. Tipoj

Ĉiuj variabloj kaj procedurnomoj el valorizotaro devas havi unu saman tipon. Se ĝi estas **Boolean**, la esprimo devas esti logika. Se la tipo estas **real** aŭ **integer**, la esprimo devas esti aritmetika. Se la aritmetika esprimo havas tipon alian ol tiu de la variabloj kaj procedurnomoj, tiam implica voko al konvenaj konvertaj funkcioj estas subkomprenata. La konverto el la reela al la entjera tipo devas liveri rezulton egalvaloran al

entier($\mathcal{E}$ + 0.5)

kie $\mathcal{E}$ estas la valoro de la esprimo. La tipon asociitan kun procedurnomo donas la deklarilo aperanta kiel la unua simbolo en la responda deklaro de proceduro.

4.3. Saltordono

4.3.1. Sintakso

‹saltordono› ::= **go to** ‹marka esprimo›

4.3.2. Ekzemploj

go to 8
go to elirejo[n + 1]
go to Urbo [**if** y < 0 **then** N **else** N + 1]
go to if Ab < c **then** 17 **else** q[**if** w < 0 **then** 2 **else** n]

4.3.3. Semantiko

Saltordono rompas la normalan vicon de agoj, difinatan per la teksta ordo de ordonoj: ĝi indikas sian sekvonton malimplice per la valoro de marka esprimo. Do, la sekva ordono plenumenda estas tiu havanta la menciitan valoron kiel sian markon.

4.3.4 Limigo

Ĉar markoj ĉiam estas lokaj, saltordono ne povas konduki en blokon el ekstere. Tamen saltordono povas konduki el ekstere en opan ordonon.

4.3.5. Salto kun nedifinita indiko de komutilo

Se la marka esprimo de saltordono estas indiko de komutilo kies valoro estas nedifinita, tian la saltordono ekvivalentas al malplena ordono.

4.4. Malplena ordono

4.4.1. Sintakso

‹malplena ordono› ::= ‹malpleno›

4.4.2. Ekzemploj

M:
begin ...; Johano: **end**

4.4.3. Semantiko

Malplena ordono estigas nenian agon. Ĝi povas servi por loki markon.

4.5. Kondiĉa ordono

4.5.1. Sintakso

‹kondiĉo› ::= **if** ‹logika esprimo› **then**

‹senkondiĉa ordono› ::= ‹baza ordono› | ‹opa ordono› | ‹bloko›

‹kondiĉita ordono› ::= ‹kondiĉo› ‹senkondiĉa ordono›

‹kondiĉa ordono› ::= ‹kondiĉita ordono› |‹kondiĉita ordono› **else** ‹ordono› |
‹kondiĉo› ‹iteracio› | ‹marko› : ‹kondiĉa ordono›

4.5.2. Ekzemploj

if x > 0 **then** n := n + 1

if v > u **then** V: q := n + m **else go to** R

if s < 0 ∨ P ≤ Q **then**
AA: **begin** if q < v **then** a := v / s **else** y : = 2 × a **end**
else if v > s **then** a : = v − q **else if** v > s − 1 **then go to** S

4.5.3. Semantiko

Kondiĉaj ordonoj ebligas preterigi aŭ plenumi iujn ordonojn depende je la kuranta valoro de la provizitaj logikaj esprimoj.

4.5.3.1. Kondiĉita ordono

La senkondiĉa ordono de kondiĉita ordono plenumiĝos se la logika esprimo de la kondiĉo estas vera. Aliokaze ĝi estos preterlasita kaj la procezon daŭrigos la tekste sekva ordono.

4.5.3.2. Kondiĉa ordono

Laŭ la sintakso estas du malsamaj formoj de kondiĉaj ordonoj, kiujn oni povas ekzemple prezenti tiel:

if *ℒ1* **then** Ω1 **else if** *ℒ2* **then** Ω2 **else** Ω3; Ω4

kaj

if *ℒ1* **then** Ω1 **else if** *ℒ2* **then** Ω2 **else if** ℒ3 **then** Ω3; Ω4

Ĉi tie *ℒ1* kaj *ℒ2* estas logikaj esprimoj, Ω1, Ω2, Ω3 estas senkondiĉaj ordonoj. Ω4 estas la ordono postiranta la tutan kondiĉan ordonon.

La plenumon de kondiĉa ordono oni povas priskribi jene. La logikaj esprimoj de la kondiĉoj estas prikalkulataj po unu laŭ la ordo dekstren ĝis troviĝos iu liveranta la valoron **true** (vera). Tiam la senkondiĉa ordono sekvanta tiun logikan esprimon estas plenumata. Se ĝi ne difinas sian sekvonton malimplice, la sekva plenumenda ordono estos Ω4, la ordono postiranta la tutan kondiĉan ordonon. Do oni povas priskribi la efikon de la limaĵo **else** tiel: por la antaŭiranta ĝin ordono **else** difinas kiel sekvonton la ordonon kiu situas malantaŭ la tuta kondiĉa ordono.

La konstruaĵo

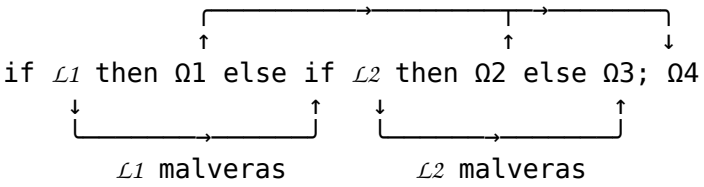
else ‹senkondiĉa ordono›

estas egalsignifa al

else if true then ‹senkondiĉa ordono›

Se neniu el la logikaj esprimoj en la kondiĉoj estas vera, la efiko de la tuta kondiĉa ordono ekvivalentas al tiu de la malplena ordono.

Kiel plua klarigo povas utili jena skemo:



4.5.4. Salto en kondiĉan ordonon

La efiko de saltordono kondukanta en kondiĉan ordonon senpere sekvas el la supra klarigo pri la efiko de **else**.

4.6. Iteracio

4.6.1. Sintakso

‹ero de ripetila listo› ::= ‹aritmetika esprimo› |
‹aritmetika esprimo› **step** ‹aritmetika esprimo›
until ‹aritmetika esprimo› |
‹aritmetika esprimo› **while** ‹logika esprimo›

‹ripetila listo› ::= ‹ero de ripetila listo› | ‹ripetila listo› , ‹ero de ripetila listo›

‹ripetilo› ::= **for** ‹variablo› := ‹ripetila listo› **do**

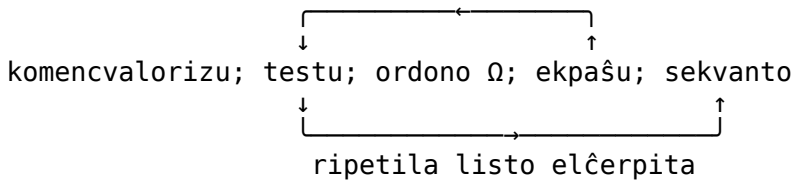
‹iteracio› ::=‹ripetilo› ‹ordono› | ‹marko› : ‹iteracio›

4.6.2. Ekzemploj

for q := 1 **step** s **until** n **do** A[q] := B[q]
for k := 1, V1 × 2 **while** V1 < N **do**
  for j := I + G, L, 1 **step** 1 **until** N, C + D **do**
    A[k, j] := B[k, j]

4.6.3. Semantiko

Ripetilo kaŭzas plenumadon de la ordono Ω kiun ĝi antaŭas nul aŭ pli fojojn. Krome ĝi realigas serion da valorizoj por la variablo kiun ĝi regas. Oni povas bildigi la procezon per la jena skemo:



Ĉi tie la vorto «komencvalorizu» signifas: Plenumu la unuan valorizon laŭ la ripetilo. «Ekpaŝu» signifas: Plenumu la sekvan valorizon laŭ la ripetilo. «Testu» ekzamenas, ĉu la lasta valorizo jam okazis. Se jes, oni pasas al la sekvanto de la iteracio. Se ne, plenumendas la ordono staranta malantaŭ la ripetilo.

4.6.4. La eroj de ripetila listo

La ripetilo provizas regulon por determini la valorojn laŭvice donatajn al la nombrilo. Tiun vicon da valoroj liveras la eroj de la ripetila listo, kiujn oni uzas unu post alia laŭ la teksta ordo. La vicon de valoroj produktataj per ĉiu el la tri specoj de eroj de ripetila listo kaj la respondan plenumon de la ordono Ω priskribas jenaj reguloj:

4.6.4.1. Aritmetika esprimo

La ero produktas unu valoron, tiun de la aritmetika esprimo prikalkulita ĵus antaŭ la responda plenumo de la ordono Ω.

4.6.4.2. Ero **step** ... **until**

La signifon de la ero « $\mathcal{A}$ **step** $\mathcal{B}$ **until** $\mathcal{C}$ », kie $\mathcal{A}, \mathcal{B}, \mathcal{C}$ estas aritmetikaj esprimoj, plej koncize priskribas samsignifaj algolaj ordonoj:

$\mathcal{V} := \mathcal{A}$;
$\mathcal{M}1$: **if** $(\mathcal{V} - \mathcal{C}) \times \text{sign}(\mathcal{B}) > 0$ **then go to** *Ero elĉerpita*;
Ordono Ω;
$\mathcal{V} := \mathcal{V} + \mathcal{B}$;
go to $\mathcal{M}1$;

Ĉi tie $\mathcal{V}$ estas la nombrilo de la ripetilo kaj *Ero elĉerpita* markas la kalkulon laŭ la sekva ero de la ripetila listo aŭ, se la ero **step** ... **until** estas la lasta en la listo, la sekvan ordonon en la programo.

4.6.4.3. Ero **while**

La efikon de ero havanta la formon $\mathcal{E}$ **while** $\mathcal{F}$, kie $\mathcal{E}$ estas aritmetika esprimo kaj $\mathcal{F}$ estas logika esprimo, plej koncize priskribas samsignifaj algolaj ordonoj:

$\mathcal{M}3$: $\mathcal{V}$:= $\mathcal{E}$;
if ¬ $\mathcal{F}$ **then go to** *Ero elĉerpita*;
Ordono Ω;
go to $\mathcal{M}3$;

kie la simbolaro estas la sama kiel en 4.6.4.2.

4.6.5. Elira valoro de la nombrilo

Se la ordono Ω estas opa, tiam tuj post eliro el Ω per saltordono la valoro de la nombrilo estos la sama kia ĝi estis ĵus antaŭ la plenumo de la saltordono. Male, se la eliron kaŭzis elĉerpiĝo de la ripetila listo, la valoro de la nombrilo post la eliro estas nedifinita.

4.6.6. Salto en ciklon

La efiko de saltordono kiu troviĝas ekster iteracio kaj referencas markon enan por ĉi-lasta, estas nedifinita.

4.7. Procedura ordono

4.7.1. Sintakso

‹fakta parametro› ::= ‹signoĉeno› | ‹esprimo› | ‹nomo de tabelo› |
‹nomo de komutilo› | ‹procedurnomo›
‹literĉeno› ::= ‹litero› | ‹literĉeno› ‹litero›
‹interparametra disigilo› ::= , |) ‹literĉeno› :(
‹listo da faktaj parametroj› ::= ‹fakta parametro› |
‹listo da faktaj parametroj› ‹interparametra disigilo› ‹fakta parametro›
‹fakta parametraro› ::= ‹malpleno› | (‹listo da faktaj parametroj›)
‹procedura ordono› ::= ‹procedurnomo› ‹fakta parametraro›

4.7.2. Ekzemploj

Spuro (A) Ordo: (7) Rezulton en: (V)
Transponu(W, v + 1)
AbsMax(A, N, M, Yy, I, K)
Skalara produto(A[t, P, u], B[p], 10, P, Y)

Tiuj ekzemploj respondas al la ekzemploj el 5.4.2.

4.7.3. Semantiko

Procedura ordono estigas (elvokas) la plenumon de procedurkorpo (v. 5.4, «Deklaroj de proceduro»). Se la procedurkorpo estas esprimita en Algolo, tiu plenumo efikas same kiel la jenaj agoj aplikitaj al la programo je la tempopunkto de plenumo de la procedura ordono.

4.7.3.1. Valorizo (transigo per valoro)

Ĉiuj formalaj parametroj menciitaj en la pervalora listo ricevas valorojn (v. 2.8, «Valoroj kaj tipoj») de la respondaj faktaj parametroj, kvazaŭ per plenumo de malimplicaj valorizoj antaŭ la eniro en la procedurkorpon. Tio efikas kiel aldona bloko, kiu entenus la procedurkorpon, kaj en kiu la valorizoj estus farataj por variabloj lokaj en tiu fikcia bloko kaj havantaj la tipojn laŭ la specifoj (v. 5.4.5). Tial variabloj transigataj per valoro estu rigardataj kiel nelokaj por la procedurkorpo, sed lokaj por la fikcia bloko (v. 5.4.3).

4.7.3.2. Anstataŭigo de nomoj (transigo per esprimo)

Ĉie en la procedurkorpo ĉiun parametron ne menciitan en la pervalora listo ekanstataŭas la responda fakta parametro, enkrampigite, se la sintakso allasas tion. Eventualaj kolizioj inter la nomoj tiel enkondukataj kaj la nomoj jam estantaj en la procedurkorpo foriĝas per konvenaj sistemecaj ŝanĝoj de tiuj ĉi formalaj aŭ lokaj nomoj.

4.7.3.3. Anstataŭigo je procedurkorpo kaj plenumo

Fine la tiel modifita procedurkorpo ekanstataŭas la proceduran ordonon kaj plenumiĝas. Se la proceduro estas vokata el ekster la videblejo de grando neloka por la procedurkorpo, la kolizioj inter la nomoj aperantaj dum la anstataŭigo je procedurkorpo kaj la nomoj kies deklaroj validas en la loko kie situis la procedura ordono aŭ la indiko de funkcio foriĝas per konvenaj sistemecaj ŝanĝoj de tiuj lastaj nomoj.

4.7.4. Interrespondo de faktaj kaj formalaj parametroj

La interrespondo de la faktaj parametroj el la procedura ordono kaj la formalaj parametroj el la procedurĉapo estas determinata tiel: La listo da faktaj parametroj en la procedura ordono devas havi la saman nombron de eroj kiel la listo da formalaj parametroj en la ĉapo de la deklaro de proceduro. La interrespondo rezultas el tio ke oni prenas la erojn de la du listoj laŭ sama ordo.

4.7.5. Limigoj

Por ke procedura ordono havu sencon, kompreneble necesas ke la transformoj de la procedurkorpo difinitaj en 4.7.3.1 kaj 4.7.3.2 konduku al

korekta algola ordono. Tio sekvigas la limigon ke la speco kaj tipo de ĉiu fakta parametro konformu al la tipo kaj speco de la responda formala parametro. Jen kelkaj gravaj konsekvencoj de tiu ĝenerala regulo:

4.7.5.1. Signoĉeno provizita kiel fakta parametro al procedura ordono aŭ indiko de funkcio kies procedurkorpo estas algola ordono (kontraste al nealgola kodo, kp 4.7.8) uzeblas en la procedurkorpo nur kiel fakta parametro por pluaj procedurvokoj. Finfine ĝin povas uzi nur procedurkorpo esprimita en nealgola kodo.

4.7.5.2. Formala parametro uzata en la procedurkorpo kiel valorizoto kaj transigata ne per valoro povas respondi nur al fakta parametro kiu estas variablo (aparta okazo de esprimo).

4.7.5.3. Formala parametro uzata en la procedurkorpo kiel nomo de tabelo povas respondi nur al fakta parametro kiu estas nomo de tabelo havanta la saman dimension. Krome, se la formala parametro transiĝas per valoro, la loka tabelo kreata dum la voko havos la samajn indicajn limojn kiel la fakta tabelo.

4.7.5.4. Formala parametro transigata per valoro ĝenerale ne povas respondi al nomo de komutilo aŭ procedurnomo aŭ signoĉeno, ĉar tiuj ne posedas valorojn (la escepto estas procedurnomo kies deklaro havas malplenan formalan parametraron — v. 5.4.1 — kaj difinas la valoron por indiko de funkcio — v. 5.4.4; tia procedurnomo ja mem estas plena esprimo).

4.7.5.5. Kun ĉiu formala parametro povas asociiĝi limigoj pri la tipoj de la responda fakta parametro (tiujn limigojn oni povas, sed ne devas, indiki per specifoj en la procedurĉapo). Kompreneble, la procedura ordono devas konformi al tiaj limigoj.

4.7.6. Forigita

4.7.7. Interparametraj disigiloj

Ĉiuj interparametraj disigiloj estas samsignifaj. Nenia respondeco de la interparametraj disigiloj uzataj en la procedura ordono kaj tiuj uzataj en la procedurĉapo estas postulata, krom ke samu ilia nombro. Do, la informoj prezentataj per la pli malsimpla formo estas nedevigaj.

4.7.8. Koda procedurkorpo

Kompreneble, la limigojn koncerne al procedura ordono, vokanta proceduron kies korpo estas esprimita en nealgola kodo, determinas la ecoj de la uzata kodo kaj la celoj de la uzanto, do ili kuŝas ekster la kadro de la referenca lingvo.

5. Deklaroj

La deklaroj servas por difini ecojn de la grandoj uzataj en la programo kaj por doni al ili nomojn. Ĉiu deklaro de nomo validas por unu bloko. Ekster tiu bloko oni rajtas uzi la nomon por aliaj celoj (v. 4.1.3).

Rultempe tio sekvigas jenon. Je la momento de eniro en blokon (tra la **begin**, ĉar la enaj markoj estas lokaj, do neatingeblaj de ekstere) ĉiuj nomoj deklaritaj por la bloko akceptas la signifon konforman al la naturo de la donitaj deklaroj. Se tiuj nomoj estis jam difinitaj ekstere per aliaj deklaroj, ili provizore ricevas novan signifon. Male, nomoj ne deklaritaj por la bloko konservas sian antaŭan signifon.

Je la momento de eliro el bloko (ĉu tra **end**, ĉu per saltordono) ĉiuj nomoj deklaritaj por la bloko perdas sian signifon.

Deklaro povas havi la aldonan deklarilon **own**. Ĝi efikas jene: Tuj post denova eniro en la blokon la valoro de la grandoj persistaj (t.e. deklaritaj kun **own**) estas nešanĝita rilate al ilia valoro ĉe la antaŭa eliro, kaj male, la valoro de la deklaritaj variabloj ne indikitaj kiel persistaj estas nedifinita. Escepte la markojn, la formalajn parametrojn en deklaro de proceduro kaj eble la nomojn de antaŭdifinitaj funkcioj (v. 3.2.4 kaj 3.2.5), ĉiuj nomoj en programo estas deklarendaj. Nomo ne povas esti deklarita pli ol unu fojon en unu sama blokmalfermo.

Sintakso

‹deklaro› ::= ‹deklaro pertipa› | ‹deklaro de tabeloj› | ‹deklaro de komutilo› |
‹deklaro de proceduro›

5.1. Deklaro pertipa

5.1.1. Sintakso

‹listo da samtipaĵoj› ::= ‹simpla variablo› |
‹simpla variablo› , ‹listo da samtipaĵoj›

‹tipo› ::= **real** | **integer** | **Boolean**

‹loka aŭ persista tipo› ::= ‹tipo› | **own** ‹tipo›

‹deklaro pertipa› ::= ‹loka aŭ persista tipo› ‹listo da samtipaĵoj›

5.1.2. Ekzemploj

integer p, q, s
own Boolean Acryl, n

5.1.3. Semantiko

Deklaro pertipa servas por deklari, ke iuj nomoj reprezentas simplajn variablojn de indikita tipo. Variablo deklarita reela povas akcepti nur pozitivajn kaj negativajn valorojn inklude de nul, variablo deklarita entjera povas akcepti nur pozitivajn kaj negativajn entjerajn valorojn inklude de nul. Variablo deklarita logika povas akcepti nur la valorojn **true** kaj **false**. En aritmetikaj esprimoj ĉiun pozicion, kiun povas okupi variablo deklarita reela, rajtas okupi variablo deklarita entjera. Pri la semantiko de **own** vidu ĉi-supre la kvaran alineon de la ĉapitro kvina (p. 32).

5.2. Deklaro de tabeloj

5.2.1. Sintakso

‹malsupra limo› ::= ‹aritmetika esprimo›

‹supra limo› ::= ‹aritmetika esprimo›

‹lima paro› ::= ‹malsupra limo› : ‹supra limo›

‹listo da limaj paroj› ::= ‹lima paro› | ‹listo da limaj paroj› , ‹lima paro›

‹grupo da tabeloj› ::= ‹nomo de tabelo› [‹listo da limaj paroj›] |
‹nomo de tabelo› , ‹grupo da tabeloj›

‹listo da tabeloj› ::= ‹grupo da tabeloj› | ‹listo da tabeloj› , ‹grupo da tabeloj›

‹deklaro de tabeloj› ::= **array** ‹listo da tabeloj› | ‹loka aŭ persista tipo›
array ‹listo da tabeloj›

5.2.2. Ekzemploj

array a, b, c[7 : n, 2 : m], s[−2 : 10]
own integer array A [**if** c < 0 **then** 2 **else** 1 : 20]
real array q[−7 : −1]

5.2.3. Semantiko

Deklaro de tabeloj indikas, ke unu aŭ pluraj nomoj prezentas multedimensiajn tabelojn el indicitaj variabloj kaj donas la dimension de la tabeloj, la limojn de la indicoj kaj la tipon de la variabloj.

5.2.3.1. Indicaj limoj

La limojn de indicoj por ĉiu tabelo donas, per listo da limaj paroj, la unua indica interkrampo sekvanta la nomon de la tabelo. Ĉiu ero de la listo indikas la supran kaj malsupran limojn de indico per du aritmetikaj esprimoj disigitaj per dupunkto. La listo da limaj paroj donas la limojn por ĉiuj indicoj laŭ la ordo dekstren.

5.2.3.2. Dimensio

La dimension donas la nombro de eroj en la listo da limaj paroj.

5.2.3.3. Tipo

Ĉiuj tabeloj deklaritaj en unu sama deklaro havas la saman indikitan tipon. Se deklarilo de tipo ne estas indikita, oni subkomprenu la tipon **real**.

5.2.4. Limaj esprimoj

5.2.4.1. La limaj esprimoj estas prikalkulataj laŭ la sama maniero kiel indicaj esprimoj (v. 3.1.4.2).

5.2.4.2. La esprimoj povas dependi nur je tiuj variabloj kaj proceduroj kiuj estas nelokaj por la bloko en kiu validas la deklaro de tabeloj. Tial en la plej ekstera bloko de programo la deklaroj de tabeloj povas havi nur konstantajn limojn.

3.2.4.3. Tabelo estas difinita nur se la valoro de ĉiu malsupra limo ne superas la respondan supran limon.

5.2.4.4. La esprimoj estas prikalkulataj po unu fojon por ĉiu eniro en la blokon.

5.2.5. Idento de indicitaj variabloj

La idento de indicita variablo neniel rilatas al la indicaj limoj indikitaj en la deklaro de tabeloj. Tamen, se tabelo estas deklarita persista, la valoroj de la respondaj indicitaj variabloj estos, ĉiufoje, difinitaj nur por tiuj el la variabloj kiuj havos la indicojn inter la plej ĵuse kalkulitaj limoj.

5.3. Deklaro de komutilo

5.3.1. Sintakso

‹komuta listo› ::= ‹marka esprimo› | ‹komuta listo›, ‹marka esprimo›
‹deklaro de komutilo› ::= **switch** ‹nomo de komutilo› ::= ‹komuta listo›

5.3.2. Ekzemploj

switch S := S1, S2, Q[m], **if** v > −5 **then** S3 **else** S4
switch Q := p1, w

5.3.3. Semantiko

Deklaro de komutilo difinas la aron da valoroj por la respondaj indikoj de komutilo. Tiuj valoroj estas donitaj unu post alia kiel la valoroj de la markaj esprimoj konsistigantaj la komutan liston. Kun ĉiu el la markaj esprimoj asociiĝas pozitiva entjero 1,2,..., ricevata per la numerado de la listeroj

dekstren. La valoro de la indiko de komutilo respondanta al donita valoro de la indica esprimo (v. 3.5, «Markaj esprimoj») estas la valoro de la marka esprimo el la komuta listo havanta la donttan valoron kiel sian numeron.

5.3.4. Prikalkulo de esprimoj en la komuta listo

Esprimoj en la komuta listo estas prikalkulataj ĉiun fojon kiam oni referencas la eron de la listo enhavantan la esprimon, uzante [en la kalkulo] la kurantajn valorojn de ĉiuj bezonaj variabloj.

5.3.5. Influo de videblejoj

Se indiko de komutilo aperas ekster la videblejo de iu grando uzata en la komuta listo, kaj se prikalkulo de la indiko de komutilo elektas tiun markan esprimon, tiam la kolizioj inter la nomoj por la grandoj el tiu esprimo kaj la nomoj kies deklaroj validas en la loko kie situas la indiko de komutilo foriĝas per konvenaj sistemecaj ŝanĝoj de tiuj lastaj nomoj.

5.4. Deklaro de proceduro

5.4.1. Sintakso

‹formala parametro› ::= ‹nomo›

‹listo da formalaj parametroj› ::= ‹formala parametro› | ‹listo da formalaj parametroj› ‹interparametra disigilo ‹formala parametro›

‹formala parametraro› ::= ‹malpleno› | (‹listo da formalaj parametroj›)

‹nomlisto› ::= nomo› | ‹nomlisto› , ‹nomo›

‹pervalora listo› ::= **value** ‹nomlisto›; | ‹malpleno›

‹specifanto› ::= **string** | ‹tipo› | **array** | ‹tipo› **array** | **label** | **switch** | **procedure** | ‹tipo› **procedure**

‹specifaro› ::= ‹malpleno› | ‹spectfanto› ‹nomlisto›; | ‹specifaro› ‹specifanto› ‹nomlisto› ;

‹procedurĉapo› ::= ‹procedurnomo› ‹formala parametraro›;

‹procedurkorpo› ::= ‹ordono› | ‹kodaĵo›

‹deklaro de proceduro› ::= **procedure** ‹procedurĉapo› ‹procedurkorpo› | ‹tipo› **procedure** ‹procedurĉapo› ‹procedurkorpo›

5.4.2. Ekzemploj

(vidu ankaŭ la ekzemplojn en la fino de la raporto)

```
procedure Spuro (a) Ordo: (n) Rezulto: (s);
  value n; array a; integer n; real s;
  begin integer k;
    s := 0;
    for k := 1 step 1 until n do s := s + a[k, k]
  end

procedure Transponu (a) 0rdo:(n);
  value n; array a; integer n;
  begin real W; integer l, k;
    for i := 1 step 1 until n do
        for k := 1 + i step 1 until n do
          begin
            w:=a[i,k]; a[i,k]:=a[k,i]; a[k,i]:=w
          end
end Transponu

integer procedure Shtupo(u); real u;
       Shtupo := if 0 ≤ u ∧ u ≤ 1 then 1 else 0

procedure AbsMax (a) amplekso: (m, n) rezulto: (y) indicoj: (i, k);
   comment La absolute plej grandan eron de la matrico a, havanta la
     amplekson m×n, sendu al y, kaj la indicojn de tiu ero sendu al i kaj k;
  array a; integer n, m, i, k; real y;
  begin integer p, q;
    y := 0;
    for p := 1 step 1 until n do
       for q := 1 step 1 until m do
          if abs(a[p,q]) > y then
            begin y:=abs(a[p,q]); i:=p; k:=q end
   end AbsMax

procedure Skalara produto (a, b) Ordo: (k, p) Rezulto: (y);
       value k; integer k, p; real y, a, b;
  begin real S; S := 0;
    for p := 1 step 1 until k do s := s + a×b;
    y := s end Skalara produto
```

5.4.3. Semantiko

Deklaro de proceduro servas por difini la proceduron asociotan al nomo de

proceduro. La ĉefa parto en deklaro de proceduro estas ordono aŭ koda teksto, la procedurkorpo, aktivigebla per proceduraj ordonoj kaj (aŭ) indikoj de funkcio el aliaj partoj de la bloko en kies malfermo situas la deklaro de proceduro. La korpon antaŭas procedurĉapo, kiu specifas la nomojn aperantajn en la korpo por prezenti formalajn parametrojn. Ĉiufoje kiam ia proceduro estos aktivigata (v. 3.2, «Indiko de funkcio» kaj 4.7, «Procedura ordono»), la formalaj parametroj en la procedurkorpo ricevos la valorojn de, aŭ anstataŭiĝos je, la faktaj parametroj. La nomoj en la procedurkorpo kiuj ne estas formalaj, estos aŭ lokaj aŭ nelokaj, laŭ tio, ĉu ili estas deklaritaj en la korpo aŭ ne. Tiuj el ili kiuj estas nelokaj en la korpo povas, ekzemple, esti lokaj por la bloko en kies malfermo situas la deklaro de proceduro. La procedurkorpo ĉiam kondutas kiel bloko, eĉ se ĝi ne havas ties formon. Tial la videblejo de marko markanta ordonon en la korpo aŭ la korpon mem ne povas etendiĝi ekster la procedurkorpon; krome, se la nomo de formala parametro estas redeklarita en la procedurkorpo (inklude la okazon de ĝia uzo por marko kiel en 4.1.3), tiam ĝi ekhavas lokan signifon kaj pro tio la faktaj parametroj respondaj al ĝi estas neatingeblaj [per ĝi] en la videblejo de tiu ena loka grando.

5.4.4. Valoro de indiko de funkcio

Por ke deklaro de proceduro difinu la valoron por indiko de funkcio, necesas ke en la procedurkorpo estu unu aŭ pli da malimplicaj valorizaj ordonoj havantaj la procedurnomon kiel valorizoton. Almenaŭ unu el ili devas esti plenumita. La tipo atribuata al la procedurnomo devas esti deklarita per deklarilo de tipo aperanta kiel la unua simbolo en la deklaro de proceduro. La valoro de la plej malfrua tia valorizo estas uzata por prikalkuli plu la esprimon en kia aperis la indiko de funkcio.

Ĉiu apero de procedurnomo en la korpo de la proceduro alie ol kiel valorizoto prezentas vokon de la proceduro.

5.4.5. Specifoj

La procedurĉapo povas enhavi specifaron, kiu per memkomprenebla simbolaro informas pri la specoj kaj tipoj de la formalaj parametroj. Nenia formala parametro povas aperi tie pli ol unu fojon. Specifoj por pervaloraj parametroj (v. 4.7.3.1) estas nepraj, por peresprimaj parametroj (v. 4.7.3.2) eblas ilin ellasi.

5.4.6. Koda procedurkorpo

Oni allasas ke la procedurkorpo estu esprimita en lingvo alia ol Algolo. Ĉar la intenco estas, ke la uzo de tiu rimedo koncernu nur la maŝinan prezenton, tial en la referenca lingvo ne eblas doni pluajn regulojn pri tiu koda lingvo.

Ekzemploj pri deklaroj de proceduro

Ekzemplo unua

```
procedure euler (fkt, sum, eps, foj);
   value eps, foj; integer foj; real procedure fkt; real sum, eps;
   comment euler kalkulas la sumon de fkt(i) por i ekde 0 ĝis la malfinio per
      konvene modifita Eŭlera transformo. La sumado ĉesos kiam en la
      transformita serio foj fojojn tujsekve aperos termoj kies absoluta valoro
      estas pli malgranda ol eps. Do, oni provizu funkcion fkt kun unu entjera
      argumento, supran limon eps kaj entjeron foj. La rezulto estas la sumo
      sum.

      euler estas precipe bona por malrapide konverĝa aŭ diverĝa signum-
      alterna serio;
begin integer i, k, n, t; array m[0:15]; real mn,mp,ds;
   i := n := t := 0; m[0] := fkt(0); sum := m[0] / 2;
Plua termo:
   i := i + 1; mn := fkt(i);
   for k := 0 step 1 until n do
      begin mp:= (mn + m[k]) / 2; m[k] := mn; mn := mp end helpvaloroj;
   if (abs(mn) < abs(m[n])) ∧ (n < 15) then
      begin ds := mn / 2; n := n + 1; m[n] : = mn end akcepte
   else ds := mn;
   sum := sum + ds;
   if abs(ds) < eps then t := t + 1 else t := 0;
   if t < foj then go to Plua termo
end euler
```

Ekzemplo dua[11]

```
procedure RK(x, y, n, fkt, eps, eta, xE, yE, unue);
value x, y; integer n; Boolean unue;
real x,eps,eta,xE; array y, yE; procedure fkt;
comment : RK integralas la sistemon
```

$$y'_k = f_k(x, y_1, y_2, ..., y_n), k = 1, 2, ... n$$

da diferencialaj ekvacioj per la metodo de Runge—Kutta, aŭtomate elektante konvenan integraladan paŝon. La parametroj estas:

11 Ĉi tiu *RK-programo* enhavas kelkajn novajn ideojn kiuj originas el S. GILL: “A process for the step by step integration of differential equations in an automatic computing machine”, *Proc. Camb. Phil. Soc.*, Vol. 47 (1951) p. 96, kaj E. FRÖBERG, “0n the solution of ordinary differential equations with digital computing machines”, *Fysiograf. Sallsk. Lund, Forhdl.* 20 Nr 11 (1950) pp. 136–152. Tamen oni atentu, ke ĝi povas esti ne optima rilate kalkultempon kaj erarojn de rondigo, kaj ke ĝi ne estas kontrolita per komputila testo.

- La komencaj valoroj x kaj y[k] por *x* kaj la nekonataj funkcioj $y_k(x)$.
- La ordo n de la sistemo.
- La proceduro *fkt*(x,y,n,z) kiu prezentas la integralendan sistemon, t.e. la aron da funkcioj f_k.
- La toleritaj eraroj *eps* kaj *eta*, regulantaj la precizon de la komputa integralado.
- La fino de la integralintervalo *xE*.
- La logika variablo *unue* kiu devas esti vera ĉe unuopa aŭ unua eniro en *RK*. Male, se la funkcioj y estas determinendaj por kelkaj maŝpunktoj x[0], x[1], ... x[n], tiam la proceduro estu ripete vokata (kun x=x_k, xE = x_{k+1}, por k = 0,1, ..., n−1) kaj tiam la postaj vokoj povos havi *unue* = **false**, kio ŝparos kalkultempon.

La eniraj parametroj de *fkt* devas esti x, y, n, la elira parametro z reprezentas la aron da derivaĵoj z[k] = f[k](x, y[1], y[2], ..., y[n]) por x kaj la faktaj y-oj.

Proceduro *komp* aperas kiel neloka nomo;

```
begin
  array z, y1, y2, y3[1:n]; real x1, x2, x3, H;
  Boolean eksteren; integer k, j;
  own real s, Hs;
  procedure RK1P(x, y, h, xe, ye); real x, h, xe; array y, ye;
      comment : RK1P integralas unuopan paŝon de Runge-Kutta kun
        komencaj valoroj x, y[k]. Tio donas la elirajn parametrojn xe=x+h
        kaj ye[k], kie la lasta estas la solvo je xe.
        G r a v a : La parametroj n, fkt, z aperas en RK1P kiel objektoj
        nelokaj;
    begin
      array w[1:n], a[1:5]; integer k,j;
      a[1] := a[2] := a[5] := h / 2; a[3] := a[4] := h; he := x;
        for k := 1 step 1 until n do ye[k] := w[k] := y[k];
        for j := 1 step 1 until 4 do
          begin
            fkt (xe,v,n,z); xe := x + a[j];
            for k := 1 step 1 until n do
              begin
                w[k] := y[k] + a[j] × z[k];
                ye[k] := ye[k] + a[j + 1] × z[k] / 3;
              end k
        end j
    end RK1P;
Komenco de la programo:
      if  unue  then begin H := xE − x; s := 0 end
                else H := Hs;
```

```
      eksteren := false;
AA:   if (x + 2.01 × H − xE > 0) ≡ (H > 0) then
        begin Hs := H; eksteren := true; H := (xE − x) / 2 end if;
      RK1P(x, y, 2 × H, x1, y1);
BB:   RK1P(x, y, H, x2, y2); RK1P(x2, y2, H, x3, y3);
      for k := 1 step 1 until n do
        if komp(y1[k], y3[k], eta) > eps then go to CC;
          comment : komp(a,b,c) estas indiko de funkcio liveranta la
            absolutan valoron de la diferenco inter la mantisoj de a kaj b
            kalkulita ĉe la eksponentoj de tiuj grandoj egaligitaj al la plej
            granda el la eksponentoj de la donitaj parametroj a, b, c;
      x := x3;
      if eksteren then go to DD;
      for k := 1 step 1 until n do y[k]:=y3[k];
      if s = 5 then begin s := 0; H := 2 × H end if;
      s :=s + 1; go to AA;
CC:   H := 0.5 × H; eksteren := false; x1 := x2;
      for k := 1 step 1 until n do y1[k] := y2[k];
      go to BB;
DD:   for k := 1 step 1 until n do yE[k] := y3[k]
end RK
```

Indekso pri difinoj de konceptoj kaj sintaksaj unuaĵoj

Ĉiuj referencoj estas indikitaj per sekcinumeroj. Estas tri specoj de referencoj:

df tian mallongigon sekvas referenco al la sintaksa difino (se tiu ekzistas)

sn tian mallongigon sekvas referencoj al la aperoj en la metalingvistikaj formuloj sintaksaj. La referencoj jam donitaj en la df-grupo ne estas ripetataj

tk tian mallongigon sekvas la referencoj al la difinoj donitaj en la teksto la bazaj simboloj prezentataj alie ol per substrekitaj vortoj estas grupigitaj en la komenco. Kompilante la indekson oni malatentadis la ekzemplojn.

Simboloj matematikaj kaj interpunkciaj

Simbolo	*Vidu ĉe:*	*Simbolo*	*Vidu ĉe:*
+	plus	₁₀	deko
−	minus	:	dupunkto
×	oble	;	punktokomo
/ ÷	divido	:=	estu
↑	potencigo	␣	spaceto
< ≤ = ≥ > ≠	‹rilatosimbolo›	( )	rondaj krampoj
≡ ⊃ ∨ ∧ ¬	‹logika operacisimbolo›	[ ]	indicaj krampoj
,	komo	‘ ’	citiloj de signoĉeno
.	dekuma punkto		

Indekso angla

[*Rimarko de la tradukinto*. La anglaj vortosimboloj estas kolektitaj en ĉi-sekcian indekson anglan. La dua kolumno kun la esperantaj tradukoj ne estas parto de la originala dokumento kaj estas aldonita por oportuno de esperantistoj.]

Angle	*Esperante*	*Referencoj*
array	**tabelo**	sn 2.3 5.2.1 5.4.1
begin	**starto**	sn 2.3 4.1.1
Boolean	**bulea**	sn 2.3 5.1.1 tk 5.1.3
comment	**komento**	sn 2.3
do	**faru**	sn 2.3 4.6.1
else	**alie**	sn 2.3 3.3.1 3.4.1 3.5.1 4.5.1 tk 4.5.3.2
end	**fino**	sn 2.3 4.1.1
false	**malvera**	sn 2.2.2
for	**por**	sn 2.3 4.6.1

Angle	*Esperante*	*Referencoj*
go to	[iru] **al**	sn 2.3 4.3.1
if	**se**	sn 2.3 3.3.1 4.5.1
integer	**entjera**	sn 2.3 5.1.1 tk 5.1.3
label	**marko**	sn 2.3 5.4.1
own	**persista**	sn 2.3 5.1.1 tk 5 5.2.5; v. persista
procedure	**proceduro**	sn 2.3 5.4.1
real	**reela**	sn 2.3 5.1.1 tk 5.1.3
step	**paŝo**	sn 2.3 4.6.1 tk 4.6.4.2
string	**ĉeno**	sn 2.3 5.4.1
true	**vera**	sn 2.2.2
until	**ĝis**	sn 2.3 4.6.1 tk 4.6.4.2
value	**valore**	sn 2.3 5.4.1
while	**kiam**	sn 2.3 4.6.1 tk 4.6.4.3

Indekso esperanta

[*Rimarko de la tradukinto*. La originalaj anglaj terminoj estas indikitaj inter rektaj krampoj. Se en la franca, germana aŭ rusa tradicioj iu termino baziĝas sur alia ideo ol la angla termino, mi aldonas ankaŭ tiun nacian tradukon post la komenclitero de la lingvo kaj dupunkto. — Krome, mi enkondukis en la indekson kelkajn teknikajn esprimojn, kiujn mi markis per la signo «•».]

Permesilo

La publikaĵon en la revuo *Communications of the ACM* akompanas jena noto (p. 17):

> NOTE: This Report is published in the *Communications of the ACM*, in *Numerische Mathematik*, and in the *Computer Journal*. Reproduction of this Report for any purpose is explicitly permitted; reference should be made to this issue of the *Communications* and to the respective issues of *Numerische Mathematik* and the *Computer Journal* as the source.

Tio estas:

> Ĉi tiu Raporto estas publikigita en la revuoj *Communications of the ACM; Numerische Mathematik* kaj *Computer Journal*. Reproduktado de ĉi tiu Raporto estas eksplicite permesita, kun referenco (RR) al la fontoj en ĉi tiu numero de la «Komunikaĵoj» kaj en la respektivaj numeroj de *Numerische Mathematik* kaj *Computer Journal*.

Bibliografio

5L Kvinlingva vortaro pri komputiko. Budapeŝto 1972.

AR Report on the algorithmic language Algol 60. *Numer. Math.* 2 (1960) pp. 106–137; *Comm. Ass. Comp. Mash.* 3.5 (May 1960) pp. 299–314.

ĈT J.W. Backus. Programování v jazyku „Algol 60“, Praha 1963, pp. 43–104; J. Raichl. Programování v Algolu. Praha: Academia, 1977, pp. 182–227.

FT E. Boliet k.a. Un nouveau langage scientifique algol. Manuel pratique. Paris, 1964, pp. 134–170.

GT Bericht über die algorithmische Sprache Algol 60. ABS N-r 3, redigiert von I.O. Kerner. Zentralinstitut für Automatisirung, Jena;
T. Ekman, C.-E. Fröberg. Lehrnbuch in Algol Programmierung. Lund, Schweden, 1969, pp. 137–181

MR R.M. De Morgan k.a. A supplement to the Algol 60 Revised Report. *Comp. J.* 19, pp. 276–287 (1976);
ACM SIGPLAN Notices, 12, No. 1, pp. 52–67 (1977);
Алгоритмический язык Алгол 60. Модифицированное сообщение. Москва: Мир, 1982.

MV EG-Wörterbuch mathematischer Begriffe (EK-vortaro de matematikaj terminoj). R. Hilders-Yashovardhan Hrsg., Europäische Reihe: Entnationalisierte Wissenschaft, Band 4. Leuchtturm-Verlag, 1950.

RR Revised Report on the algorithmic language Algol 60. *Numer. Math.* 4, pp. 420–453 (1963);
Comm. Ass. Comp. Mach. 6, pp. 1–17 (1963);
Computer J. 5, 349–367 (1963); *Annual review in automatic programming*, 4 — Pergamon press, 1964, pp. 217–255.

RT Алгоритмический язык Алгол-60. Пересмотренное сообщение. Москва: Мир, 1965.

Enhavtabelo