INFORMATICA 3/1984

PROGRAMSKI JEZIK PASCAL II.

M. GAMS (1),
I. BRATKO (2,1),
V. BATAGELJ (3),
R. REINHARDT (1),
M. MARTINEC (1),
M. ŠPEGEL (1),
P. TANCIG (1)

UDK: 519.682.8

(1) Institut „JOŽEF STEFAN",
(2) FAKULTETA ZA ELEKTROTEHNIKO, UNIVERZA E. KARDELJA
(3) FAKULTETA ZA NARAVOSLOVJE IN TEHNOLOGIJO

V članku so najprej opisana pravila lepega programiranja v Pascalu. Večji del članka je posvečen slabostim Pascala in oceni kritik v literaturi. Sledi opis primernih in neprimernih področij uporabe, zaključna ocena pa vsebuje celotno analizo.

Programming Language Pascal II (detailed analysis of Pascal's shortcomings). First we very shortly describe what is "good" programming in Pascal and than we devote special care to Pascals shortcomings and critics in literature. Finally we describe which problem domains are suitable for Pascal and which not.

1. Uvod

Vsakemu jeziku lahko očitamo kopico pomanjkljivosti. Tudi Pascal ni izjema. Nekaterim slabostim se da izogniti z doslednim programiranjem, nekatere slabosti pa so neprijetne za uporabnika. Zato smo najprej navedli pravila lepega programiranja v Pascalu. Šele nato smo skušali zbrati vse smiselne pripombe in jih oceniti. Na koncu so zbrani rezultati ankete o Pascalu. Vse skupaj naj da uporabnikom dodatno znanje o tem, kdaj uporabiti Pascal in kdaj ne, snovalce novih jezikov pa naj opozori na nekatere napake, ki naj jih pri današnjem stanju računalniške znanosti ne bi smeli ponavljati. Vse to razkrivanje napak pa naj ne vrže slabe luči na Pascal, saj si ta izvrstni jezik zasluži več pohval kot kritik in najbrž enega prvih mest med algoritmičnimi jeziki /13/.

Radi bi se zahvalili vsem, ki so sodelovali pri popravljanju članka, predvsem pa: Janezu Žerovniku, Marku Bohancu, Henriku Krnecu, Damjanu Bojadžijevu in Igorju Mozetiču.

2. Pravila lepega programiranja v Pascalu

Vsak jezik bolj podpira (in vsiljuje) nekatere stile programiranja. Z uporabo človeku bolj naravnega stila programiranja se poveča produktivnost programerja. Program večkrat beremo, kot pa ga pišemo. Vsak jezik, še posebej pa splošno namenski jezik, je mogoče "zlorabiti", to je uporabiti na neprimeren način. Npr. programerji, ki s Fortrana preidejo na Pascal, pogosto brez potrebe uporabljajo "goto" stavek. Podobno lahko zlorabimo Pascal z nepremišljeno uporabo globalnih spremenljivk /1/.

2.1. Strukturiranje

Pascal omogoča učinkovito strukturiranje, tj. pravilno strukturo zapisa algoritma. Osnovni so trije principi strukturirane gradnje programov:

a) od zgoraj navzdol (top-down)

b) od spodaj navzgor (bottom-up)

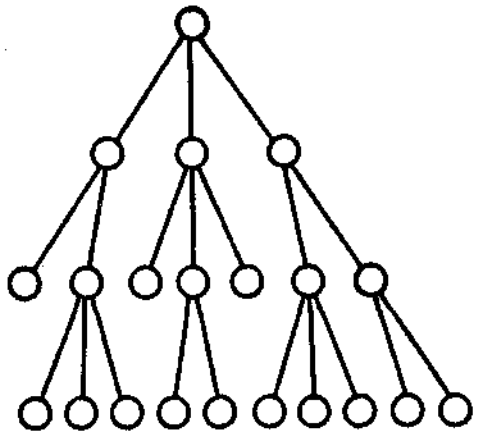
c) kombinacija a in b.

Princip "od zgoraj navzdol" gradnje programov je v tem, da obsežne heterogene kose delimo na manjše in bolj kompaktne dele. Manjše dele lažje obvladamo, preglednost raste.

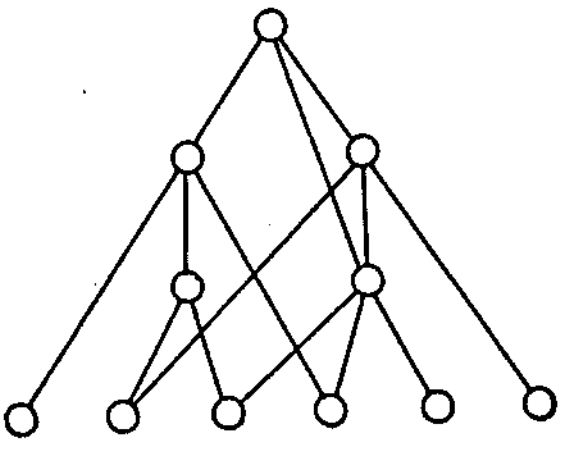
Princip "od spodaj navzgor" gradnje programov je v tem, da se nekatera osnovna opravila - atomarne funkcije - pogosto ponovijo. Te atomarne operacije najprej zakodiramo, nato pa čimveč ostalih podprogramov gradimo s čimveč klici že kodiranih podprogramov. Tako lahko realiziramo programe z manj kode, "moč" kode naraste.

V praksi se najbolj pogosto srečujemo z uporabo obeh stilov, tako da sproti razbijamo večje kose v manjše, kadar pa naletimo na osnoven podprogram, ga takoj zakodiramo. Med kodiranjem vedno preverjamo, ali lahko kakšno nalogo rešimo tako, da uporabimo že zakodiran (malce popravljen) podprogram. Podprogram naj bo dolg okoli ene strani in naj vsebuje največ 7 podprogramov. Podprogram mora imeti omejen pretok informacij preko čimmanj parametrov. Za globalne spremenljivke glej pravilo številka 2.2. Kadar bo program obsežen, moramo pred kodiranjem vedeti, kaj vse bo sestavljalo en modul (za overlaye ali za vsebinsko povsem ločene module), drugače pa pod-

programov rajši ne gnezdimo, ampak jih gradimo "od spodaj navzgor" ali v obliki produkcijskih sistemov /2,3/.



Slika 1: Strukturiranje "od zgoraj navzdol": večje module razbijamo v manjše tako, da med moduli ni povezav. "Idealna" kontrolna struktura je drevo brez prekrivanja listov, gradimo ga od zgoraj navzdol.



Slika 2: Strukturiranje "od spodaj navzgor": osnovne podprograme čimprej zakodiramo in ostale podprograme gradimo s pomočjo že kodiranih podprogramov, ki so dostopni vsem podprogramom. Kontrolna struktura je neplanaren usmerjen graf, prekrivanja je precej.

Vsak izmed načinov kodiranja ima svoje prednosti in slabosti. Bolj pogosto srečamo kodiranje od zgoraj navzdol, ker lažje razbijamo velik problem na lažje podprobleme in zato je preglednost običajno zelo dobra. Strukturiranje od spodaj navzgor omogoča veliko izkoriščenost kode, vendar ni primerno za vsa področja. Kljub temu je po mnenju nekaterih ob pazljivi uporabi to izredno učinkovita metoda.

Poglejmo si rezultate strukturiranja na primeru iz literature/4/: Ko so obstoječi program v jeziku PL/I preoblikovali iz nestrukturirane oblike v strukturirano, so testirane osebe za razumevanje nestrukturiranega programa potrebovale dvakrat več časa kot za razumevanje strukturirane verzije. Tudi drugi faktorji kot ciklomatična kompleksnost (ustreza intuitivnemu pojmu zapletenosti ciklov) so se bistveno izboljšali pri praktično isti hitrosti izvajanja. Pač pa se je bistveno (za polovico) podaljšalo število vrstic programa, največ na račun deklaracij procedur.

2.2. Ne uporabljaj goto stavkov in globalnih spremenljivk, če to ni nujno potrebno.

2.2.1. Pogosta uporaba konstrukta "goto" zelo poslabša čitljivost kode /4/, zato ga uporabljaj le v izjemnih primerih, npr. pri nenadni prekinitvi izvajanja ali pri skoku iz zanke.

2.2.2. Globalne spremenljivke uporabljaj le takrat, kadar so pogosto uporabljane v večjem številu podprogramov, ali kadar so podatkovne strukture obsežne. V komentarju ob klicu podprograma in v glavi komentarja navedi, katere globalne spremenljivke nastopajo v podprogramu in kdaj (kako) spremenijo vrednost znotraj podprograma.

2.3. Zamikaj

Stile zamikanja si lahko ogledamo v literaturi. Pri zamikanju ne obstaja en splošno veljaven princip in vsak programer ima svojo inačico oblikovanja programa. Smisel zamikanja je v tem, da z obliko nakažemo strukturo programa, oziroma algoritma. Lepo zamikanje močno poveča čitljivost programa.

2.4. Uporabljaj mnemonična imena za spremenljivke, tipe, imena podprogramov, itd.

Tako se močno poveča čitljivost in samodokumentiranost programa. Mnemoničnih imen ne uporabljamo le za podatke, ki nimajo semantičnega pomena, npr. tabela "TAB" ali "A" nastopa v podprogramu za sortiranje, ali za pogosto rabljene spremenljivke kot npr. "i" za števec v zanki. Le izjemoma - ali sploh ne - krajšamo mnemonična imena npr. "število vrstic" v "stvr". Samostojne besede združujemo v eno ime tako, da prvo črko samostojne besede pišemo z veliko začetnico, npr. "številoVrstic" ali "ŠteviloVrstic".

2.5. Komentiraj, dokumentiraj

Obvezno opiši vlogo vsakega podprograma, kaj dela, kakšen vhod in izhod ima in vse posebnosti. S komentarjem loči podprograme ali obsežne stavke, npr. tako, da ima prvi "begin" in pripadajoči "end" pripisano ime podprograma ali ime konstrukta. Vsak podprogram naj ima vsaj en stavek komentarja.

3. Analiza kritik Pascala

3.1. Uvod

V strokovni literaturi za vsak programski jezik najdemo vsaj nekaj kritičnih člankov. Bistven problem snovalcev vsakega programskega jezika je najti pravo ravnotežje med sposobnostjo in preprostostjo. Tudi to je eden od razlogov, da noben jezik ne more biti primeren za vsa področja in za vse stile programiranja. Kritika je upravičena zlasti takrat, kadar opozori na očitno slabost ali kadar predlaga boljšo rešitev.

Programski jezik je običajno definiran na štirih nivojih:

a) abstraktna sintaksa (za Pascal ena A4 stran)
b) konkretna sintaksa (za Pascal štiri A4 strani sintaktičnih diagramov)
c) priročniki in ostala literatura o standardnem jeziku (Pascal User Manual and Report ima 167 strani)
d) prevajalnik za standardni jezik (okoli 150 - 200 strani listinga).

Običajno tudi velja, da se kontekstno svobodna gramatika konča z b) in v c) opišemo kontekstno občutljivo gramatiko.

Z nivojem narašča količina informacij in določenost jezika. Iz abstraktne sintakse je npr. že razvidna možnost stranskih efektov pri funkcijah. S konkretno sintakso so določene stvari kot hierarhija operatorjev, oblika kontrolnih konstruktov, itd. Izjeme, ki jih s konkretno sintakso ni mogoče lepo opisati, so opisane v priročnikih. Tak primer je IF - THEN - IF - THEN - ELSE konstrukt, za katerega iz sintaktičnih diagramov ni jasno, ali zadnji ELSE pripada prvemu ali drugemu IF-u. Zaželjeno je, da je takih izjem čim manj. Slabo je, kadar ostanejo take izjeme neopisane ali prepuščene implementatorju. V Pascalu je to primer za evaluacijo (ovrednotenje) Boolovih izrazov, ko ni jasno, ali se v logičnem izrazu "p1 and p2 and p3" vedno ovrednotijo vsi trije pogoji ali celoten izraz dobi vrednost "false", kadar npr. "p1" dobi vrednost "false". Kot posledica tega imajo nekateri prevajalniki en in drugi drug način evaluacije, to pa poslabša prenosljivost programov.

Večina kritik v literaturi je povzeta v nadaljevanju članka. Poleg kritik je narejena analiza in ocena upravičenosti kritike. Kritike so v veliki meri privzete po /5/, predvsem pa se skicujemo še na /6,7,8,9/ in /10/. Pri tem analiziramo iste slabosti Pascala, le da se ocene včasih razlikujejo.

3.2. Problemi na nivoju kodiranja

3.2.1. Nezaključen komentar. Pri komentiranju pogosto naredimo napako, da pozabimo zaključiti komentar, Recimo:

```
i := 1 - 1; (* nek komentar pozabimo zaključiti ??)
j := 1 ;    (* drug komentar *)
```

To napaka je pogosto neprijetna, saj je prevajalnik ne odkrije in lahko se zgodi, da program dela pravilno, razen za posebno kombinacijo vhodnih podatkov.

Komentar: Predlagana rešitev z vrstičnimi komentarji kot v Fortranu je slabša kot naslednji dve možnosti:

- dober listing programa nam pokaže, kateri stavki so komentarski
- prevajalniku bi lahko dodali opozorilo, kadar bi znotraj komentarja naleteli na simbol "(*" .

Obe rešitvi je mogoče enostavno realizirati.

3.2.2. Fiksiran red deklaracij - CONST, TYPE, VAR. Nekateri avtorji trdijo, da je bolj smiselno dodati možnost poljubnega ponavljanja teh treh deklaracij. Komentar: Smiselno bi bilo dodati poljuben vrstni red deklaracij. To zahteva minimalne popravke prevajalnika, olajša pa probleme pri ločenem prevajanju ali delu s knjižnicami.

3.2.3. Napake zaradi "default-a". Neprijetna napaka je npr. pozabljeni "var" pri deklariranju parameterov ali to, da pozabimo deklarirati spremenljivko znotraj podprograma in postane globalna. Primer:

```
procedure increment ((*var*) i: integer);
(*var i : integer; *)
begin
   i := i + 1
end ;
```

Komentar: Mogoče bi bilo smiselno izločiti "default" princip. Tako bi za globalne spremenljivke vedno deklarirali npr. "global" in za lokalne parametre npr. "value".

3.3. Problemi s tipi

3.3.1. Vrednost funkcije je lahko le nekaj strogo določenih tipov. Komentar: Brez večjih naporov bi lahko spremenili prevajalnik tako, da bi imele funkcije poljuben tip in nekatere izvedenke Pascala to že imajo. Omejitev vrednosti funkcij pa ni zelo neprijetna, saj si lahko pomagamo s klicem procedure.

3.3.2. Množice. Komentar: Množice so zelo uporaben podatkovni tip. Na žalost pa so vezane na dolžino besede in največja možna dolžina množic je tako najpogosteje vezana na konkreten računalnik, kar zelo otežkoča prenosljivost programov. Dokaz za boljšo implementacijo množic najdemo npr. v izpeljanki Pascala "P+" /5/. Možna rešitev bi bila, če bi v standardnem Pascalu dovolili deklaracijo dolžine množice na

način kot pri dinamičnih tabelah. Druga slabost množic je, da so v Pascalu lahko elementi množice samo statični objekti, torej se njihova vsebina s časom ne more spreminjati. Zaradi tega so množice učinkovito implementirane, žal pa to dostikrat onemogoča udobno programiranje. Zato bi bilo mogoče smiselno dodati poseben konstrukt, imenujmo ga "LIST", ki bi omogočal tudi konstrukte tipa "množica zapisov" (set of record). Tak konstrukt pa je razumljivo implementacijsko precej počasnejši od standardnih množic.

3.4. Manjkajoči konstrukti

3.4.1. Manjkajoči inverzni konstrukt "ORDu". Razen za znake (chr) inverzni konstrukt "ORDu" ne obstaja. Komentar: Smiselno bi bilo dodati inverzni konstrukt "ORDju", saj ne zahteva velike popravkov prevajalnika. Tako bi za vsak enostaven tip T = (v0, v1, v2, ..., vn) imeli funkcijo ORD: i := ORD(vi) in funkcijo za inverzno transformacijo: vi := Create(T,i). Brez te inverzne funkcije npr. ne moremo preprosto poiskati srednjega elementa za neceloštevilčne indekse:

```
srednjiElement :=
  Create(T,(ORD(leviElement) +
  + ORD(desniElement)) div 2));
```

Tak konstrukt je enostavno dodati v prevajalnik.

3.4.2. Manjkajoči "LOOP" in "EXIT" konstrukt. Nekateri avtorji predlagajo "LOOP" konstrukt, podobno kot v COBOLU, in s prekinitvijo zanke z "EXIT"<name> konstruktom (ta obstaja v nekaterih verzijah Pascala). Komentar: V Pascalu je zadovoljiv konstrukt

```
WHILE true DO
BEGIN
  ----
  IF alarm THEN EXIT (*ali izjemama 'GOTO 111'*);
  ----
END;
111:
```

"LOOP" konstrukt je po mnenju večine nepotreben.

3.4.3. Manjkajoči "ELSE" v "CASE" stavku. Komentar: Smiselno bi bilo dodati "ELSE" ali "OTHERWISE" konstrukt v "CASE" stavek, saj zahteva majenkosten popravek prevajalnika. Novejši prevajalniki ga večinoma imajo.

3.5. Slabosti

3.5.1. Vrstni red evaluacije Boolovih izrazov. V poglavju 3.1. smo omenili, da ni jasno, ali se ovrednotijo vsi podizrazi v Boolovem izrazu tipa "p1 and p2", ali se vrednotenje prekine čim prevajalnik ugotovi, da je "p1" false. V naslednjih treh primerih vidimo prednosti evaluacije s prekinitvijo:

```
WHILE (i <= MAX) AND (a[i] <> 0) DO ...
WHILE (p <> nil) AND (p^.vsebina <> iskanaVsebina) DO...
WHILE NOT eof(f) AND NOT (ft = '*') DO ...
```

Komentar: V priročniku bi morali določiti način ovrednotenja Boolovih izrazov. Precej boljše je ovrednotenje s prekinitvijo.

3.5.2. Datoteke. Standardni Pascal nima definiranega povezovanja logičnih datotek z datotekami v operacijskem sistemu. Nima sintakse za branje in pisanje po datotekah z direktnim dostopom. Nima indeksno sekvencialnih datotek. Okence v datotekah je vedno inicializirano na tekoče mesto v datoteki, kar otežkoča interaktivno delo. Pri branju numeričnih podatkov je otežkočano testiranje konca datoteke. Komentar: Opis odpiranja datotek ali branja po datotekah z direktnim dostopom je močno vezan na operacijski sistem, zato se sintaktično močno razlikuje. Večina teh problemov je zadovoljivo rešena na konkretnem prevajalniku. Hujša pomanjkljivost je pomanjkanje indeksno sekvencialnih datotek na večini prevajalnikov. Le redke verzije Pascala (Pascal R) jih imajo. Na splošno pa so datoteke šibka točka Pascala.

3.5.3. Prirejanje začetnih vrednosti. V Pascalu ne moremo uporabiti naslednjih izrazov:

```
CONST
    n = 10;
    m = 20;
    i = n * m;
VAR
    a: array[1..n*m] of integer;
BEGIN a := (3, 4, 5, ...)
```

Prav tako ne moremo deklarirati začetnih vrednosti spremenljivkam pri deklaraciji, ne obstaja ekvivalent "DATA" stavku iz Fortrana. Komentar: Prirejanje začetnih vrednosti bi lahko zelo enostavno dodali v prevajalnik. Nekateri prevajalniki imajo dano to možnost.

3.5.4. Nezmožnost direktnega dosega računalniških enot. Pascal nima posebnih konstruktov za sistemsko delo kot npr. Modula ali C. Komentar: Te naloge lahko realiziraš z zbirnikom, kar pa še vedno ni enakovredno jezikom za sistemsko delo.

3.5.5. Pomanjkanje "lastnih" spremenljivk. Pascal nima lastnih spremenljivk, torej podprogram ne more imeti svoje spremenljivke, ki bi obdržala vrednost do ponovnega klica podprograma, ne da bi bila vidna vsem ostalim podprogramom kot npr. globalne spremenljivke. Komentar: Zaradi velikega pome-

na skritih spremenljivk za ločeno prevajanje in knjižnice bi bilo smiselno dodati lastne spremenljivke. To ne bi zahtevalo veliko popravkov prevajalnika in pogosto najdemo tovrstne možnosti v novejših prevajalnikih.

3.5.6. Neučinkovitost knjižnic. Knjižnice so neučinkovite zaradi stroge kontrole prenosa parametrov (type checking) in pomanjkanja skritih spremenljivk. Zato je v večini obstoječih prevajalnikov npr. nemogoče napisati funkcijo, ki bi izračunala dolžino poljubnega niza (packed array (MINX .. MAXX) of char). Novi ISO standard /9/ to sicer omogoča, vendar samo za tabele. Problemi z dinamičnimi strukturami (seznami, drevesa) pa ostajajo nerešeni. Komentar: Pri ločenem prevajanju in pri klicanju podprogramov v zbirniku ni kontrole tipov, vendar to ne omogoča splošno uporabnih podprogramov npr. za procesiranje dreves. To je precejšnja slabost, ki pa je ni mogoče enostavno rešiti. Nekateri prevajalniki kot Pascal 2 pa omogočajo prenose tudi brez kontrole tipov.

3.5.7. Nezmožnost ločenega prevajanja in procesiranja v realnem času. V standardnem Pascalu ni govora o ločenem prevajanju ali o konstruktih za nadzorovanje procesov v realnem času. Komentar: Večina prevajalnikov omogoča ločeno prevajanje, vendar vsak na svoj način. Nezmožnost ločenega prevajanja je ena večjih slabosti Pascala. S pomočjo podprogramov v zbirniku lahko na večini računalnikov procesiramo v realnem času tako, da podprogrami v zbirniku kličejo podprograme v Pascalu. Ta možnost pa zahteva globlje poznavanje zbirnika, pascalskega prevajalnika in operacijskega sistema.

4. Anketa o Pascalu

V reviji Sigplan Notices so objavili rezultate ankete o Pascalu /11/. Tu številke pomenijo število pozitivnih odgovorov.

Prednosti Pascala:

30 kontrolne strukture
26 podatkovni tipi
20 stroga kontrola tipov
14 preprostost
8 prenosljivost
5 dobra čitljivost
4 rekurzija
4 kontrola mej

Slabosti Pascala:

7 nezmožnost ločenega prevajanja
6 omejeno branje/čitanje
5 stroga kontrola tipov
3 slabe možnosti procesiranja nizov

Zakaj uporabljate Pascal

13 ker je prenosljiv
12 ker je enostaven za uporabo
10 ker omogoča dobro strukturiranje
6 ker omogoča strogo kontrolo tipov
8 ne uporabljajo Pascala, ker obstaja boljši jezik (najpogosteje omenjeni je C)
(opomba - verjetno so vprašani mislili na uporabo jezika na sistemskem področju. C je verjetno bolj uporaben za sistemsko delo, manj pa za ostala področja).

5. Primerna in neprimerna področja uporabe

Standardni Pascal brez dodatkov je primeren za:
- pisanje prevajalnikov
- procesiranje tekstov
- pisanje uporabniških programov, npr. editorjev
- procesiranje nenumeričnih podatkov
- procesiranje dreves, seznamov in drugih kompleksnih podatkovnih tipov
- nekatere matematične probleme
- za učenje
- za pisanje splošno prenosljivih programov.

Brez dodatkov je manj primeren za:
- sistemsko programiranje
- aplikacije v realnem času in kontrolo procesov
- paralelno procesiranje
- konstrukcijo velikih programov
- numerično analizo
- aplikacije, ki zahtevajo indeksno sekvencialne datoteke
- nekatere poslovne aplikacije.

Velja poudariti, da lahko za večino manj primernih področij učinkovito izboljšamo lastnosti Pascala, tako da uporabimo podprograme v zbirniku ali Fortranu. Poleg tega lahko uporabimo bolj specializirano usmerjene verzije kot USCD Pascal za mikroračunalnike, Pascal PLUS za diskretne simulacije in Concurrent Pascal za aplikacije v realnem času. Novejši Pascali uspešno rešujejo večino tu omenjenih problemov.

6. Zaključna ocena

Nekaj slabosti Pascala bi lahko z minimalnim trudom odpravili, tako da bi imel prevajalnik malo popravkov, funkcijsko pa bi bil Pascal precej močnejši. In najbrž bi moralo vsakemu Pascalskemu prevajalniku dodati indeksno sekvencialne datoteke. Precej kritik je nemogoče razrešiti, ne da bi se prevajalnik in jezik pretirano razširila. Večino omenjenih problemov imajo novejši Pascali običajno dokaj elegantno rešenih. Kljub vsemu pa je Pascal po svojih lastnostih verjetno eden najboljših pred-

stavnikov algoritmičnih splošno namenskih jezikov. Jeziki kot FORTRAN, COBOL, BASIC ali PL/1 so v splošnem objektivno nekaj slabši, čeprav so primernejši za določena področja. Velja naslednje: Pascal (in podobni jeziki kot MODULA-2 ali inačice ADE) je verjetno eden najboljših splošno namenskih jezikov, čeprav skoraj na vsakem ožjem področju lahko najdemo jezike, ki so boljši kot Pascal. Učinkovitost novejših Pascalskih prevajalnikov tako glede dolžine kode generiranega programa in hitrosti izvajanja kode je približno taka kot pri novejših Fortranskih prevajalnikih /12/, torej običajno neprimerno boljša kot pri BASICu, COBOLu ali PL/1. Velika slabost Pascala ostaja slabo programersko okolje. Programer veliko časa porabi za testiranje programov. Jeziki (bolje rečeno programsko okolje), ki omogočajo ločeno prevajanje, uspešno testiranje (debugger) in imajo interpreter in prevajalnik, ter vse to integrirano in istočasno dostopno, so bistveno boljše orodje za testiranje, kot pa npr. Pascal. Tudi Pascalske knjižnice s splošno uporabnimi podprogrami so le redkokdaj komercialno dosegljive in jih mora uporabnik pisati sam, še zlasti kadar bi rad odpravil kakšno pomanjkljivost Pascala s podprogramom v zbirnem jeziku. Ravno to pa je področje, kjer lahko v naslednjih letih pričakujemo največje korake naprej.

7. Literatura

1. E.B. Levy: The Case Againt Pascal as a Teaching Tool, ACM SIGPLAN Notices, Vol. 17, Num. 11, str. 39-42, november 1982
2. D.A. Waterman, F. Hayes-Roth: An Overview of Pattern-Directed Inference Systems, Academic Press, 1978
3. M. Gams: Pomen in vloga znanja v sistemih za interakcijo z uporabnikom, magistrsko delo, junij 1982
4. J.L. Elshof, M. Marcotty: Improving Computer Program Readibility to Aid Modification, CACM, Vol. 25, Num. 8, str. 512-521, avgust 1982
5. R. Cailliau: How to Avoid Getting SCHLONKED by Pascal, ACM SIGPLAN Notices, Vol. 17, Num. 12, str. 31 - 41, december 1983
6. R.E. Sumner, R.E. Gleaves: Modula-2 -- A Solution to Pascal's Problems, ACM SIGPLAN Notices, Vol. 17, Num. 9, str. 28 - 34, september 1982
7. R. Cailliau: A Letter to Editor, ACM SIGPLAN Notices, Vol. 17, Num. 12, str. 10 - 11, december 1982
8. K. Jensen, N. Wirth: Pascal, User Manual and Report, Springer Verlag, 1978
9. Second draft proposal ISO/DP 7185 - Specification for the Computer Programming Language - Pascal, Pascal News, Num. 20, december 1980
10. N. Wirth: The Design of a Pascal Compiler, Software Practice and Experience, 1, str. 309 - 333, 1971
11. K. Mogel: A Report on a PASCAL Questionaire, ACM SIGPLAN Notices, Vol. 17, Num. 10, str. 23 - 33, oktober 1982
12. Benchmark test na Quicksortu, Special Software Limited, Informatica 3, str. 77, 1982
13. M. Gams, I. Bratko, V. Batagelj, R. Reinhardt, M. Martinec, M. Špegel, P. Tancig: PASCAL I (primerjava z ostalimi jeziki), Informatica 1, str. 22 - 26, 1984

Novice iz Instituta "Jožef Stefan"

Domač enokartični mikroračunalnik tipa DEC

Pod vodstvom M.M.Miletića je bil razvit enokartični mikroračunalnik MMA-11 z DECovim 16-bitnim mikroprocesorjem T-11, ki uporablja operacijski sistem RT-11. Ta računalnik ima 64k-zložni hitri pomnilnik, dvoje serijskih in ena paralelna vrata, krmilnik za upogljivi disk itd. Predviden je tudi krmilnik za trdni disk. Podobnost z mikroračunalnikom VT-150 je očitna.

Program za povezovanje na tiskanih vezjih

Na IJS je bil razvit program za iskanje povezav na tiskanih vezjih (M. Gams). Problem 500 povezav je npr. ta program razrešil v manj kot 4 urah (računalnik Delta 4850). Program je v paketu z rutinami za izrisovanje shem, filmov in izdelavo trakov za numerično krmiljeni vrtalni stroj.

A. P. Železnikar