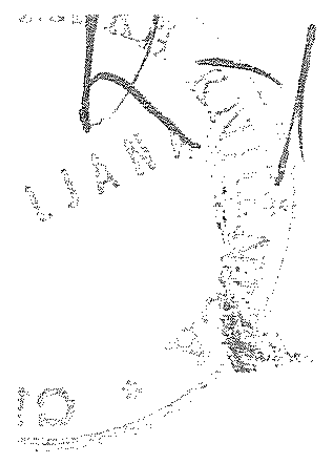




1997

Letnik 44

6

OBZORNIK ZA MATEMATIKO IN FIZIKO



OBZORNIK ZA MATEMATIKO IN FIZIKO

LJUBLJANA, NOVEMBER 1997

letnik 44, številka 6, strani 161–192

Glasilo Društva matematikov, fizikov in astronomov Slovenije, 1001 Ljubljana, Jadranska c. 19, p.p. 2964, telefonska št. (061) 17-66-553, žiro račun 50106-678-47233, devizna nakazila SKB banka d.d. Ljubljana, val-27621-42961/9, Ajdovščina 4, Ljubljana.

Uredniški odbor: Miran Černe (glavni urednik), Boris Lavrič (urednik za matematiko in odgovorni urednik), Martin Čopič (urednik za fiziko), Boštjan Jaklič (tehnični urednik).

Jezikovno pregledal Janez Juvan, računalniško oblikoval Martin Zemljič.

Člani društva prejema Obzornik brezplačno. Celoletna članarina 2.500 SIT. Naročnina v knjigarnah in za ustanove 5.000 SIT, za študente 1.250 SIT, za tujino 50 DEM. Posamezna številka za člane 480 SIT, stare številke 320 SIT.

Tisk: Tiskarna KURIR. Naklada 1500 izvodov.

Revijo sofinancirata Ministrstvo za znanost in tehnologijo ter Ministrstvo za šolstvo in šport.

Po mnenju MZT št. 415-52/92 z dne 5.2.1992 šteje revija med proizvode iz 13. točke tarifne št. 3 zakona o prometnem davku, za katere se plačuje 5% davek od prometa proizvodov.

DMFA je včlanjeno v Evropsko matematično društvo (EMS), v Mednarodno matematično unijo (IMU), v Evropsko fizikalno društvo (EPS) in v Mednarodno združenje za čisto in uporabno fiziko (IUPAP). DMFA ima pogodbo o recipročnosti z Ameriškim matematičnim društvom (AMS).

© 1997 DMFA Slovenije – 1334

Poštšina plačana na pošti 1102 Ljubljana

VSEBINA — CONTENTS

Članki – Articles

Str.–Page

Nenavadni interferenčni poskusi: Uklon na fraktalih — Unusual interference experiment: Diffraction by fractals (Janez Strnad) 161–165

Programski jezik oberon: nov pristop k objektno usmerjenemu programiranju — Programming language Oberon: a new approach to object-oriented programming (Viljan Mahnič) 166–182

O zgodovini algebre — On the history of algebra (Borut Zalar) 183–190

Vesti — News

Ob odkritju doprsnega kipa dr. Franca Močnika (Bojan Hvala) 191–III

Popravek 182

Letno kazalo III–IV

Na ovitku: Claude Monet, *Impresija, vzhajajoče sonce*, 1873

NENAVADNI INTERFERENČNI POSKUSI

UKLON NA FRAKTALIH

JANEZ STRNAD

PACS 42.25.Hz

Z lasersko svetlobo je mogoče opazovati zanimive interferenčne slike fraktalov, na primer Kochovih snežink.

UNUSUAL INTERFERENCE EXPERIMENT

DIFFRACTION BY FRACTALS

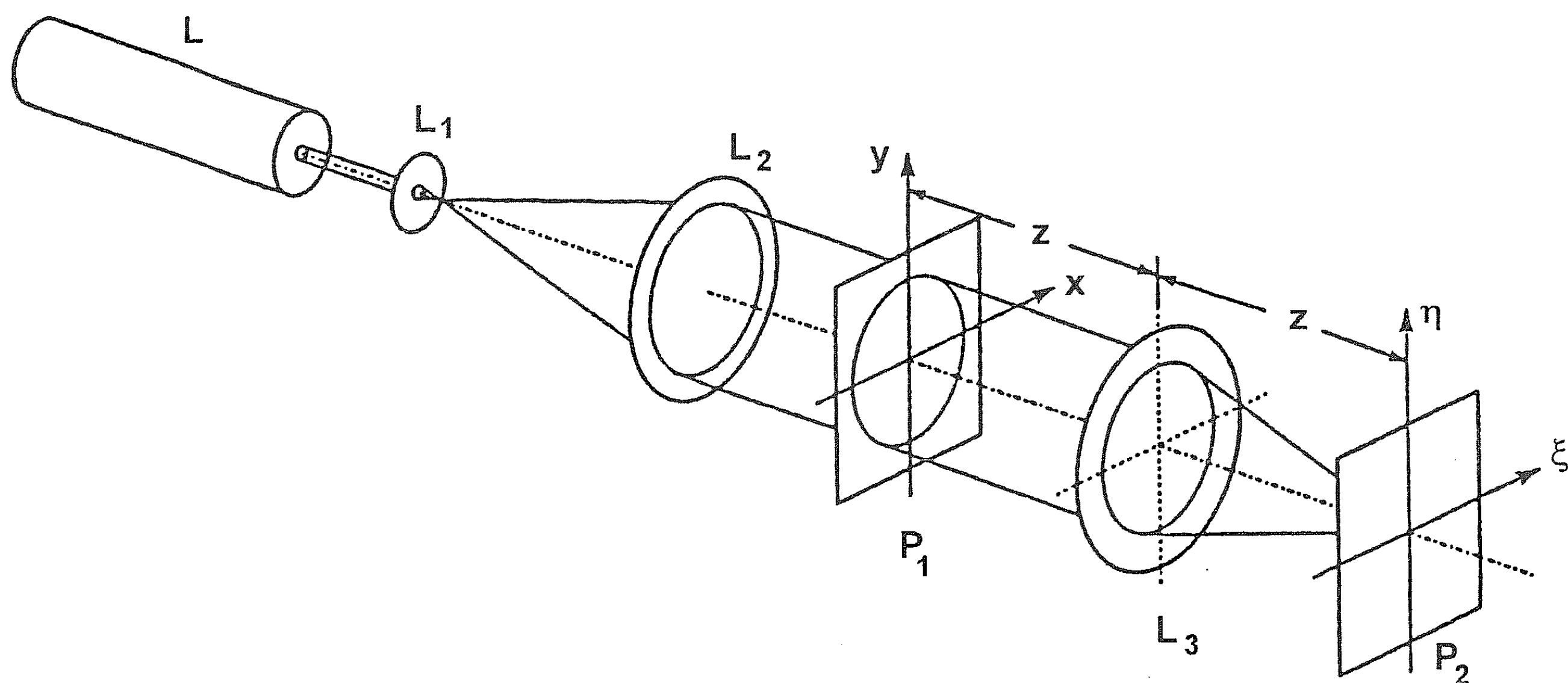
With laser light one can observe interesting interference patterns of fractals, e.g. Koch curves.

Fraktali so pritegnili precejšnjo pozornost, tako da so z njimi v šoli poskusili povečati zanimanje za matematiko. O nekaterih od njih, Cantorjevi množici, Kochovi krivulji ter trikotniku Sierpinskega je poročal Obzornik [1]. Za poučevanje fizike niso tako zanimivi kot deterministični kaos, s katerim so se skupaj pojavili. Ena od možnosti za uporabo v šoli je uklon svetlobe na fraktalih. Interferenčne slike so privlačne in zanimive tudi same po sebi in jih je mogoče uporabiti pri spoznavanju fraktalov na eni strani in uklona na drugi. Opisali so uklonske poskuse na Cantorjevi množici v obliki črt [2], na Kochovi snežinki [3] in na množici trikotnikov Sierpinskega [4].

Za poučevanje v prvem letu na univerzi je uklon svetlobe na Kochovi snežinki izkoristila skupina z univerze Joensuu na Finskem in univerze Hokkaido v Sapporu na Japonskem [5], [6]. Fraktale zaradi lepih slik študenti večinoma že poznajo, zato o njih ni treba povedati veliko več kot to, da so sebi podobni, to je invariantni na podobnostno preslikavo. Značilna zanje je fraktalna dimenzija D , večja od 1, ki ustreza daljici, in manjša od 2, ki ustreza ravninskemu liku. Za Cantorjevo množico je $D = \log 2 / \log 3 = 0,63$, za Kochovo krivuljo $D = \log 4 / \log 3 = 1,26$. Pri uklonu obdelamo le preprostejši Fraunhoferjev način opazovanja, pri katerem na predmet pravokotno pade vzporedni curek enobarvne svetlobe, in opazujemo interferenčno sliko na neskončno oddaljeni ravnini. V tem primeru smemo vzeti, da niha jakost električnega polja v vseh točkah predmetne ravnine sočasno in prispevajo k interferenčni sliki vzporedni šopi žarkov. Da ni treba opazovati na neskončno oddaljeni ravnini, uporabimo zbiralno lečo in opazujemo sliko na zaslonu v njeni goriščni ravnini.

Za poskus potrebujemo najprej fraktal. V ta namen ga z računalnikom izračunamo, navadno z rekurzijsko zvezo po določenem številu korakov. S tiskalnikom ga narišemo na list papirja. Na drobnozrnat fotografski film posnamemo nekaj manj kot centimeter velik negativ. S fotografskim objektivom razširimo curek helij-neonskega laserja in z zbiralno lečo naredimo vzporeden curek, s katerim pravokotno osvetlimo film. V curek prepuščene

svetlobe postavimo drugo zbiralno lečo in v njeno goriščno ravnino zaslon, na katerem opazujemo interferenčno sliko (sl. 1). Pri neposrednem opazovanju je slika dokaj šibka, če uporabimo miliwattni helij-neonski laser. Bolje jo je posneti na fotografski film ali neposredno opazovati s polprevodniško slikovno kamero. Da bi prišli do uporabnih fotografij, je potrebnih nekaj izkušenj pri delu z laserjem. Poskusi, da bi jih dobili na hitro, so se izjalovili. Pri zahtevnejših poskusih pogosto uporabijo ionski argonski laser pri valovni dolžini 488 nm z močjo več sto miliwattov.

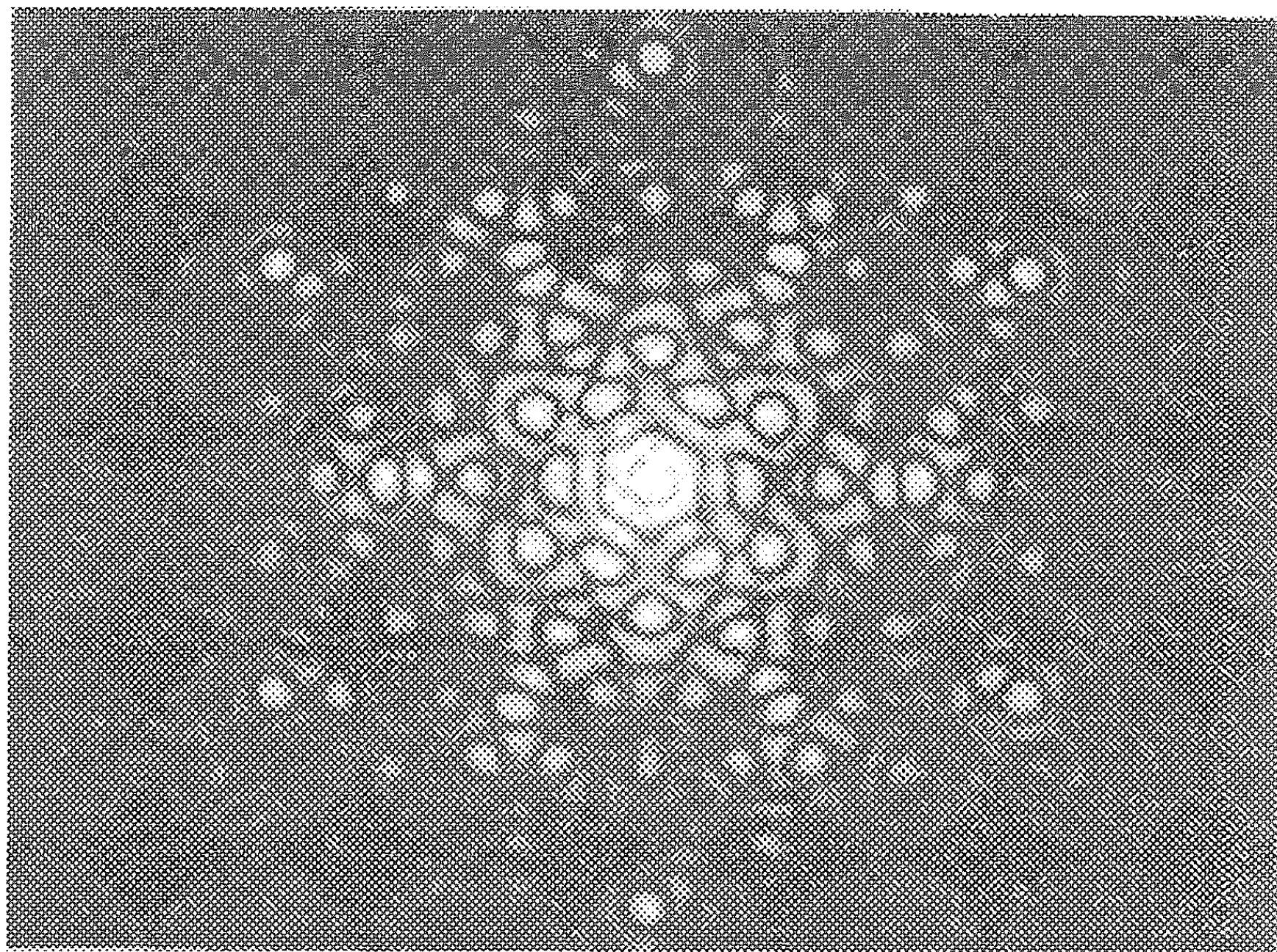
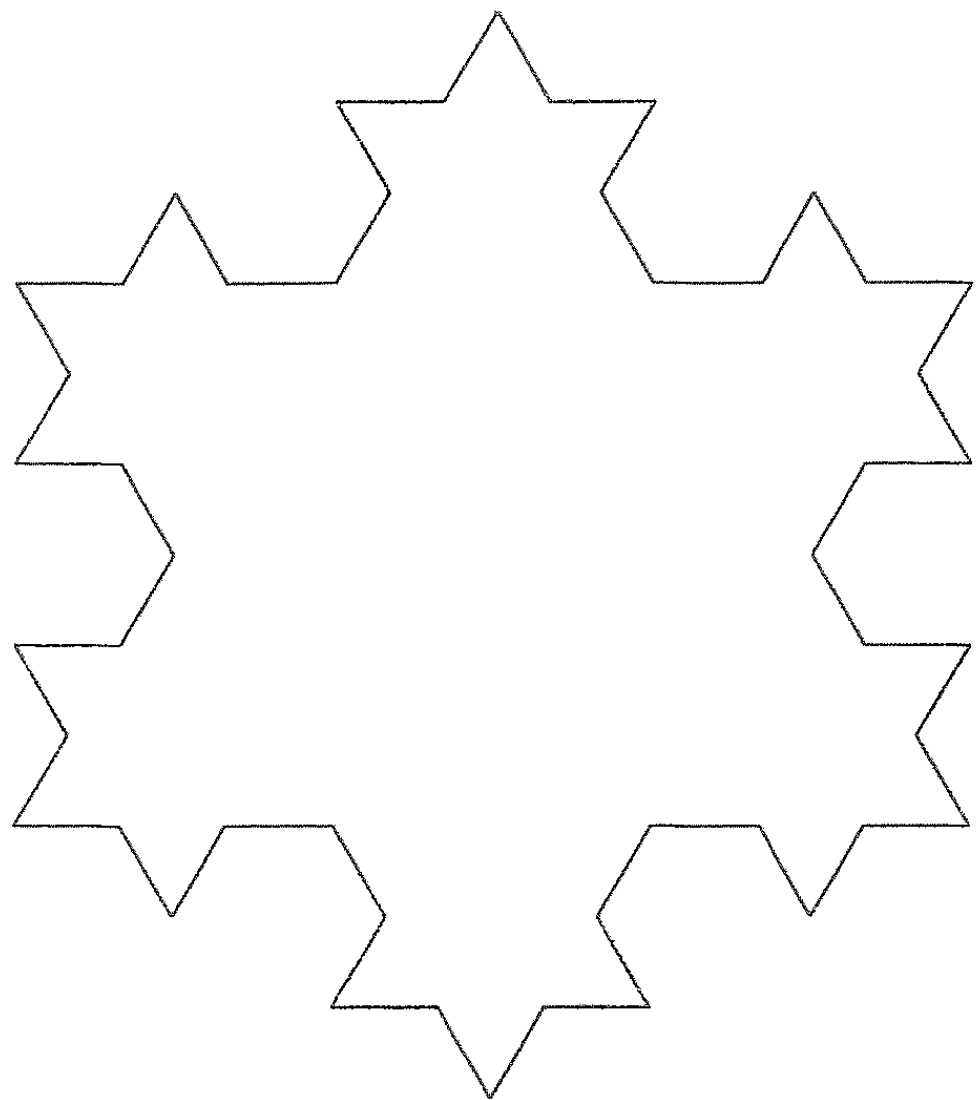


Slika 1. Običajna razporeditev naprav pri uklonu: L laser, L_1 , L_2 , L_3 zbiralne leče, P_1 predmetna ravnina, P_2 slikovna ravnina.

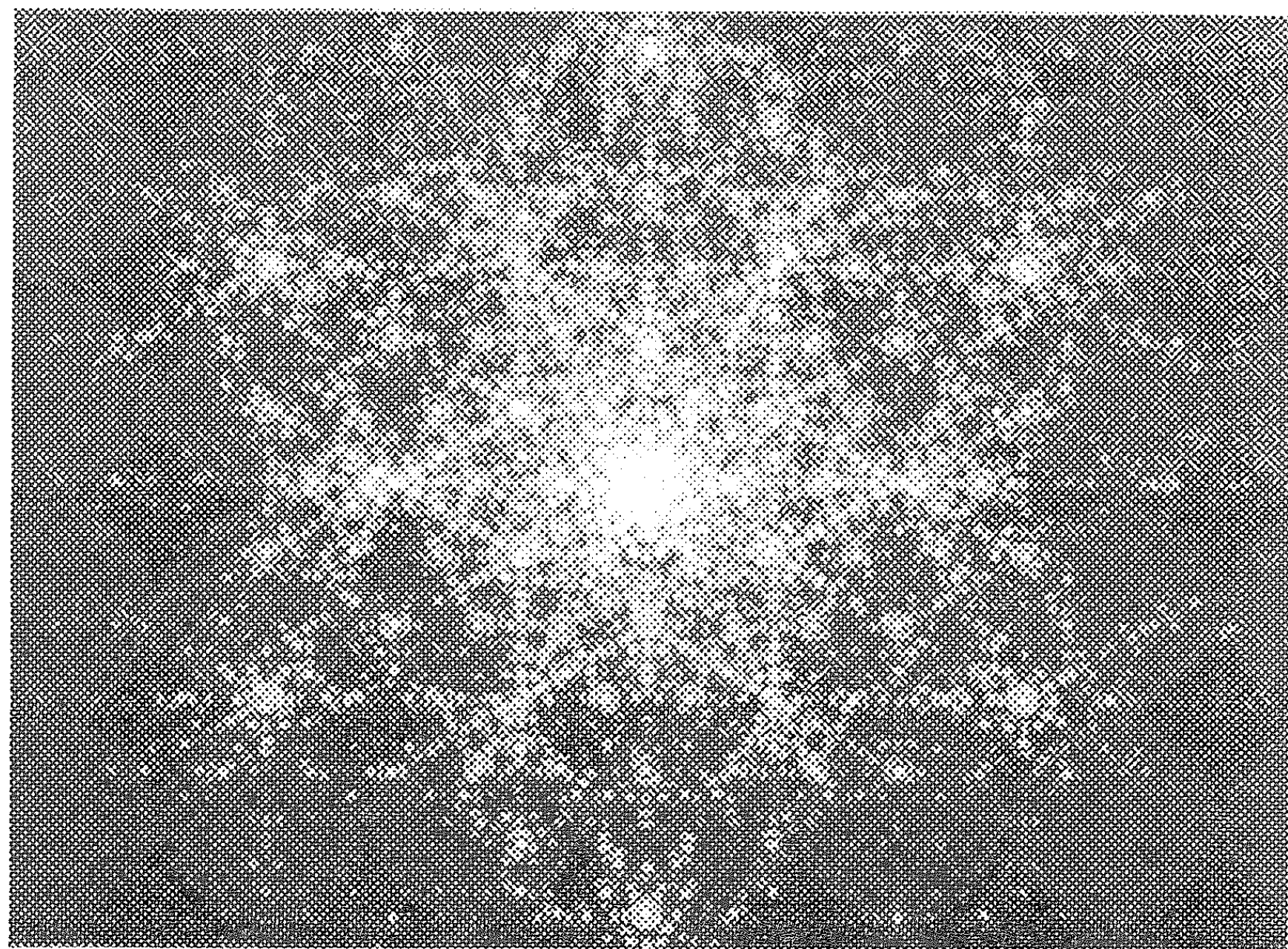
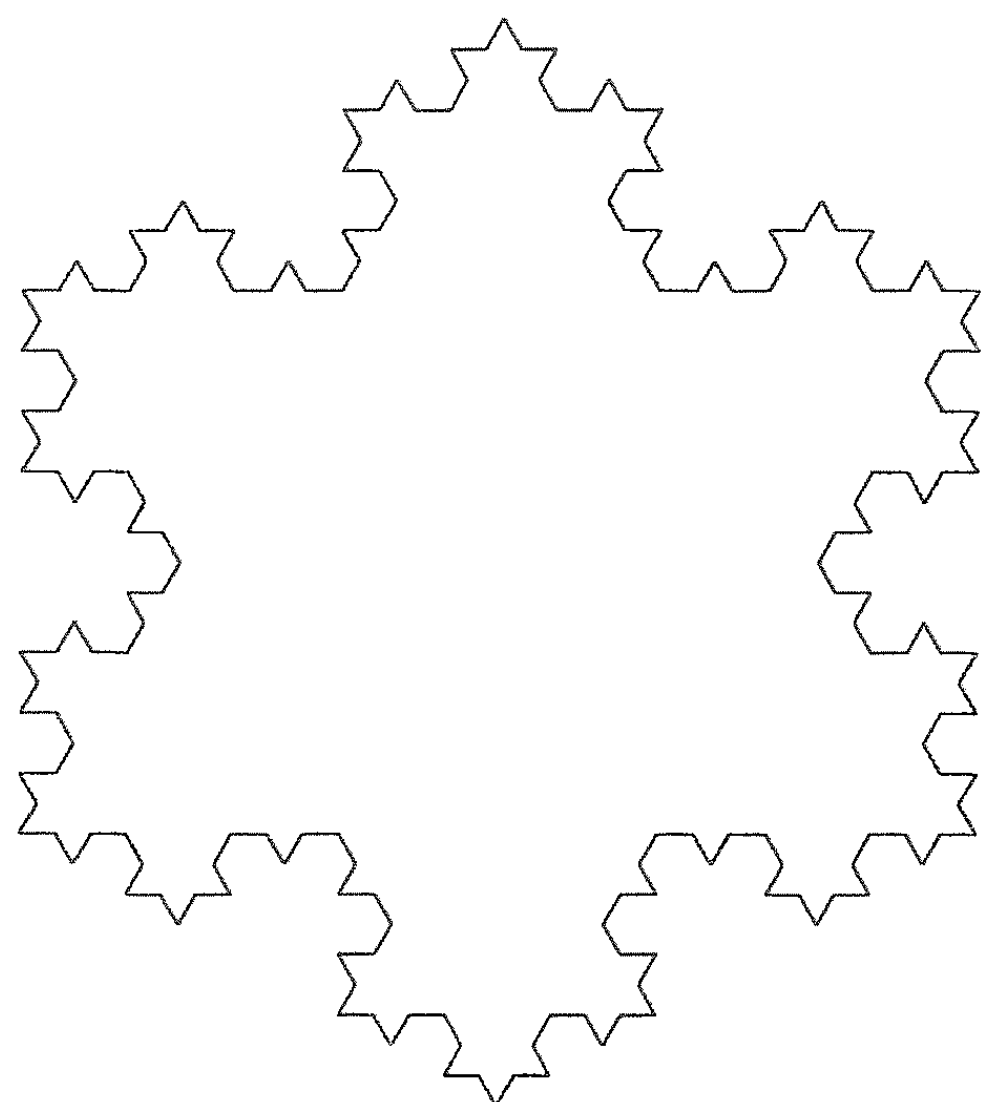
Zanimivo je primerjati interferenčne slike Kochovih snežink, dobljene po 3, 4 in 5 korakih (sl. 2). Interferenčna slika se precej spremeni pri vsakem nadaljnjem koraku. Pri majhnem številu korakov je predmet majhen in je zgradba interferenčne slike groba. Po večjem številu korakov je predmet večji in prispeva k interferenčni sliki več delnih valovanj, zato njena zgradba kaže več podrobnosti. Velikost slike pa ni odvisna od števila korakov. Interferenčna slika treh Kochovih snežink, dobljenih po 3, 4 in 5 korakih s skupnim središčem z velikostmi, sorazmernimi z 1, 3 in 9, kaže še posebno bogato zgradbo (sl. 3). Z izpopolnjeno napravo so posneli hologram Kochove snežinke (sl. 4). Razširjeni curek argonskega ionskega laserja so razdelili na dva delna curka. Enega od njiju so skozi lečo speljali naravnost na zaslon, drugi je dosegel zaslon po odboju na zrcalu.

Nazadnje pogledjmo, kaj se da o interferenčni sliki fraktalov pri Fraunhoferjevem načinu opazovanja ugotoviti s preprostim računom. Zamislimo si predmetno ravnino in v veliki oddaljenosti l od nje vzporedno slikovno ravnino. Točko na predmetni ravnini zaznamujemo s koordinatama x in y , točko na slikovni ravnini s koordinatama ξ in η . Iz izhodišča predmetne ravnine naj izhaja žarek pod kotom α proti osi x in pod kotom β proti osi y . Žarek seka slikovno ravnino v točki s koordinatama $\xi = l \tan(\frac{1}{2}\pi - \alpha) \approx \approx l \sin(\frac{1}{2}\pi - \alpha) = l \cos \alpha$ in $\eta = l \tan(\frac{1}{2}\pi - \beta) \approx \approx l \sin(\frac{1}{2}\pi - \beta) = l \cos \beta$. Kota α in β sta zaradi velike oddaljenosti l dovolj majhna, da se njuna tangensa zanemarljivo malo razlikujeta od sinusov. K jakosti električnega polja v tej

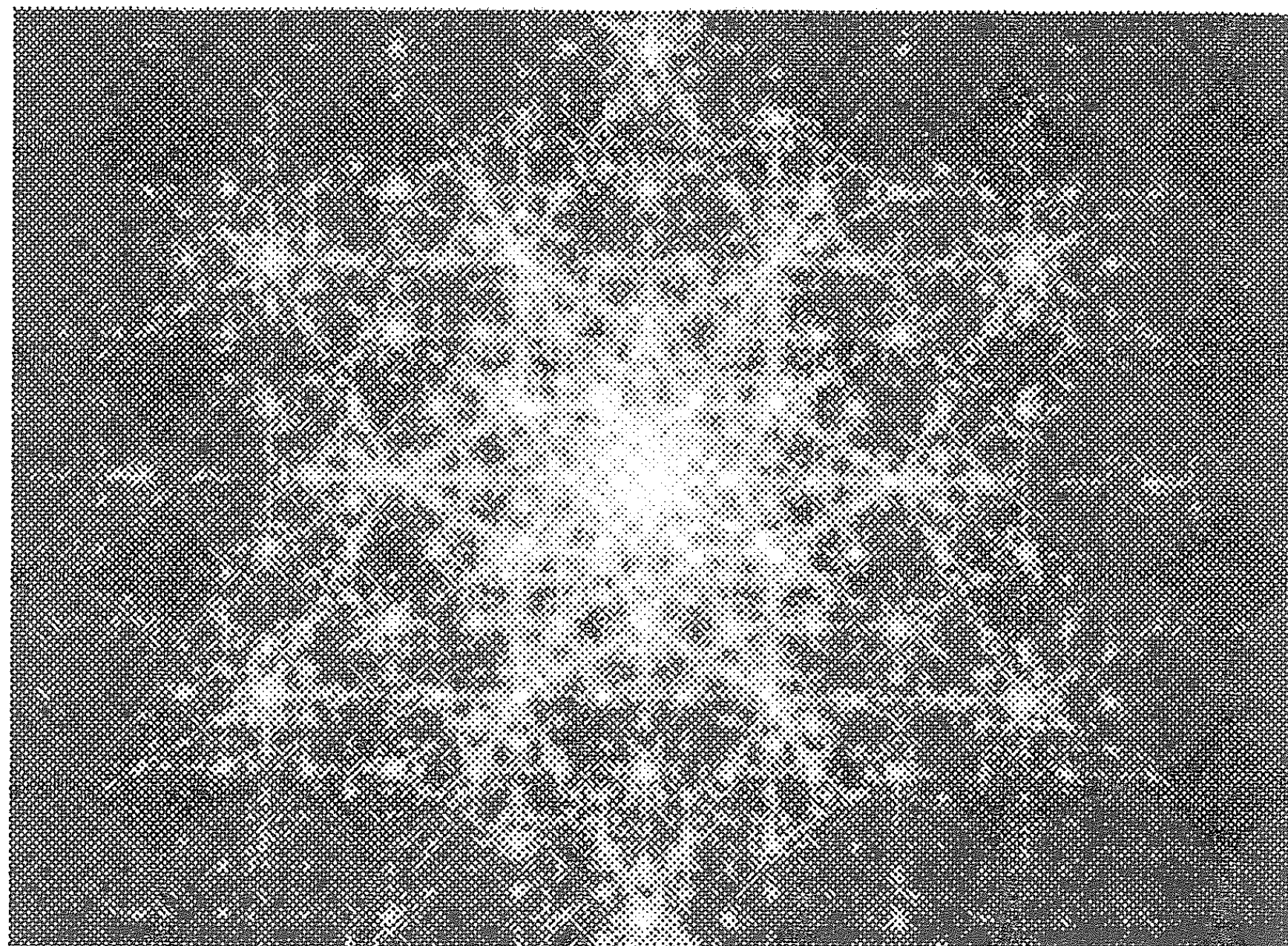
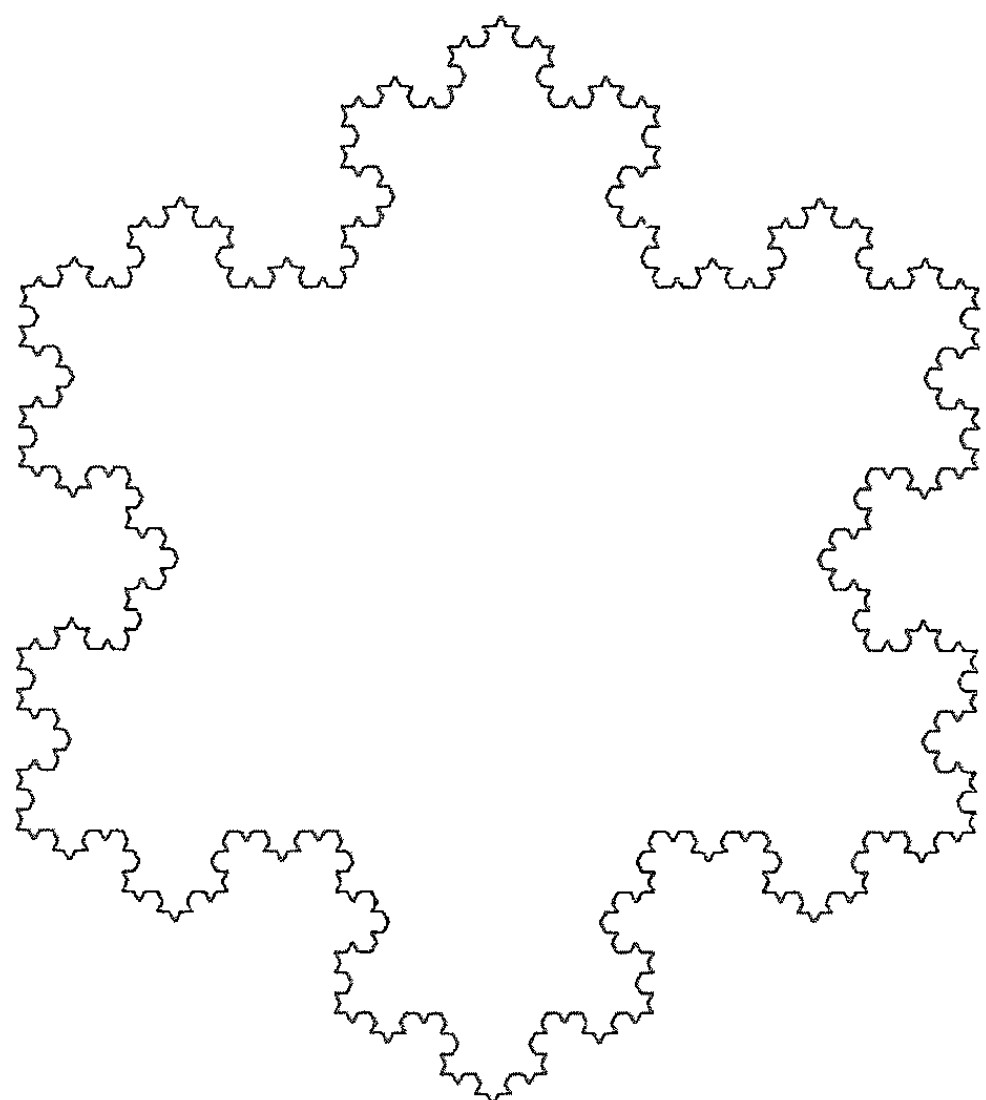
(a)



(b)

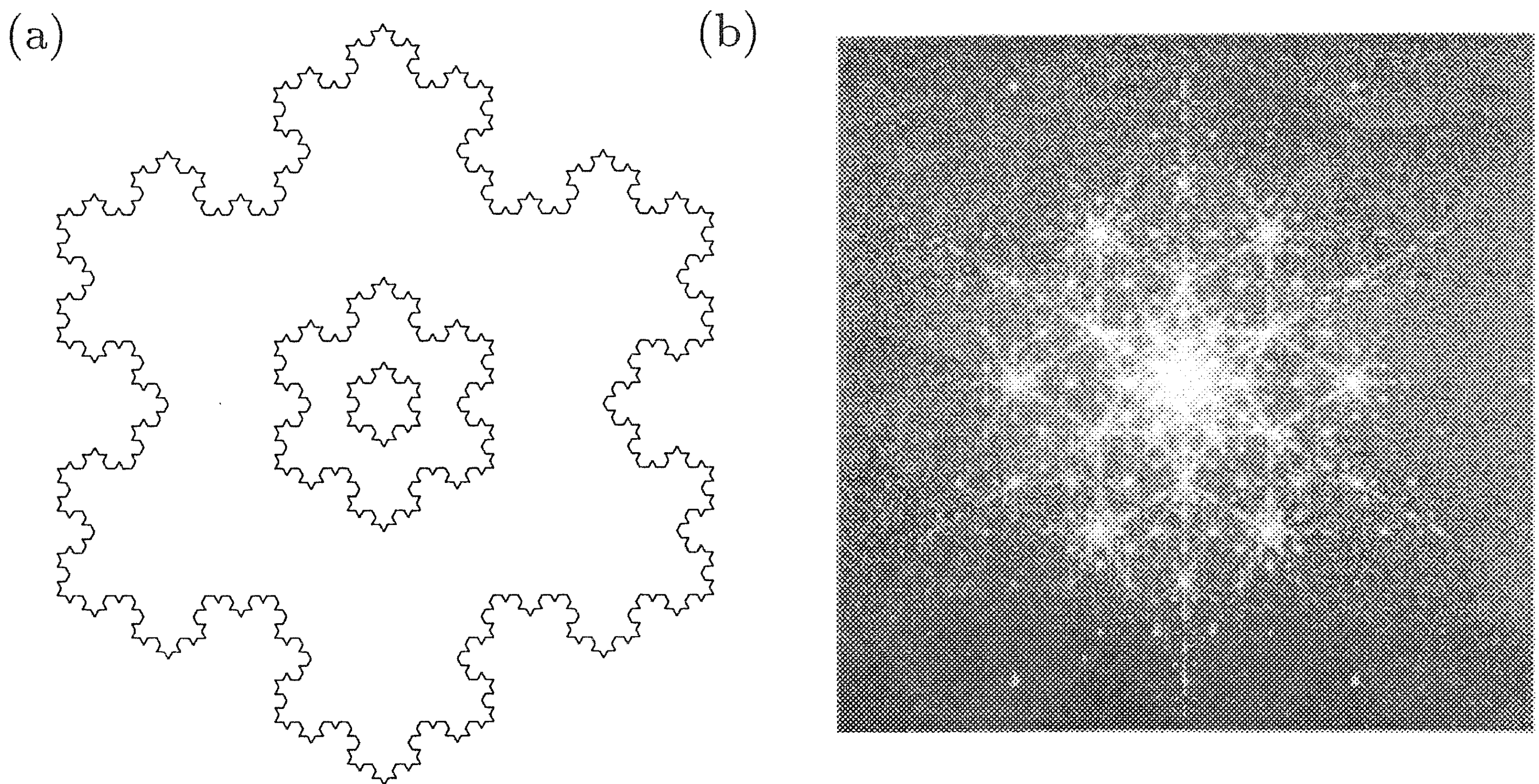


(c)



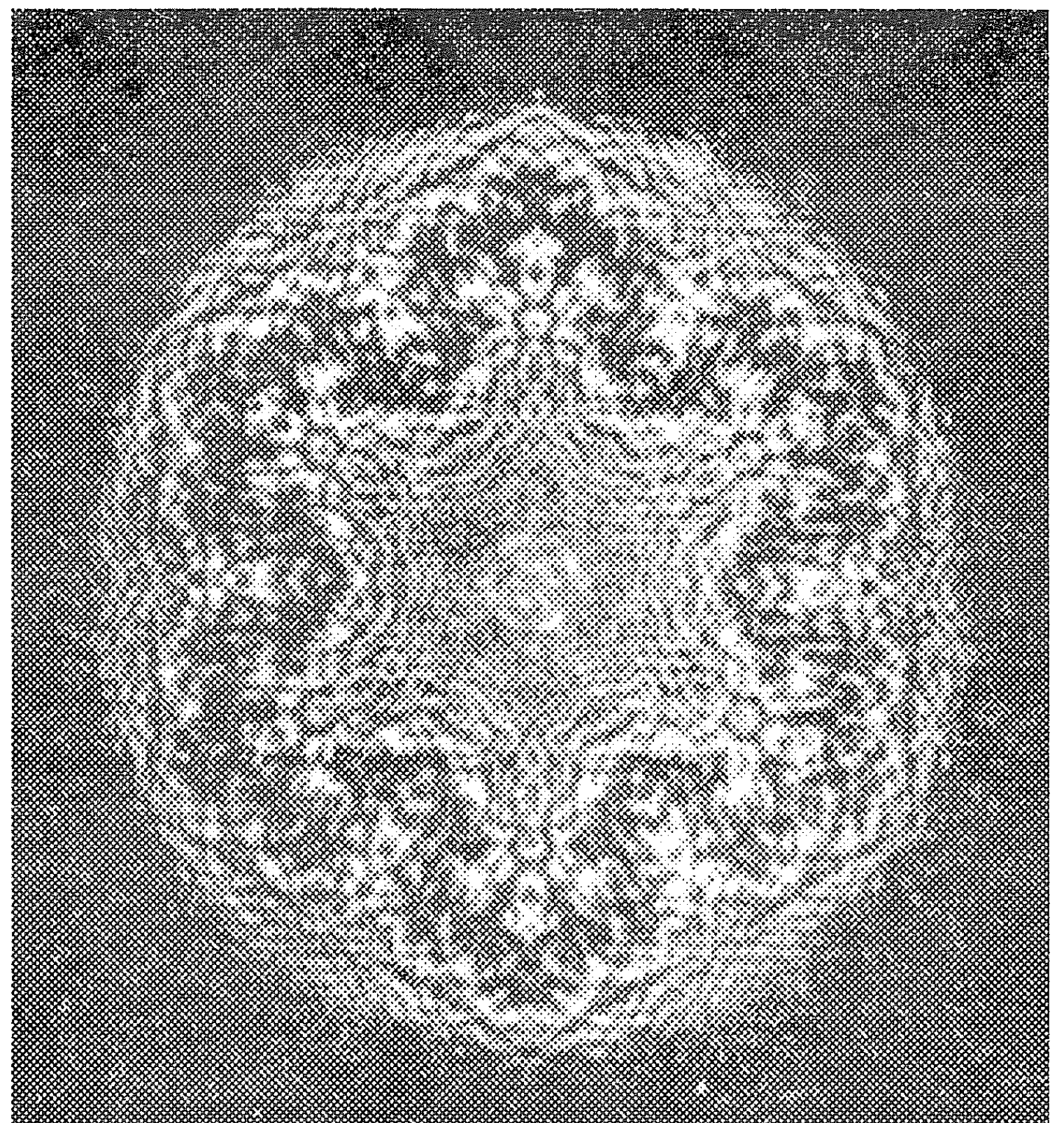
Slika 2. Kochove snežinke po 3 (a), 4 (b) in 5 korakih (c) (levo) in njihove interferenčne slike (desno).

točki slikovne ravnine prispevajo še delna valovanja iz vseh točk predmetne ravnine, ki so vzporedna z žarkom iz izhodišča. Toda ta valovanja imajo za $x \cos \alpha + y \cos \beta$ daljšo pot, ki ji ustreza fazna razlika $\delta = 2\pi(x \cos \alpha + y \cos \beta)/\lambda$. Zato moramo $e^{i\delta}$ pomnožiti s sploškovnim elementom predme-



Slika 3. Tri Kochove snežinke po 3, 4 in 5 korakih s skupnim središčem v razmerju velikosti 1 : 3 : 9 (a) in njihova interferenčna slika.

Slika 4. Hologram Kochove snežinke, dobljen z argonskim laserjem. Vse slike so z ljubeznivim dovoljenjem profesorja K.-H. Peiponena in profesorja J. Uozumija vzete iz prispevka [5].



tne ravnine $dx dy$ in sešteti vse prispevke po predmetni ravnini. Naposled moramo upoštevati, da jakost električnega polja v točkah predmetne ravnine ni konstantna, ampak jo določa prepustnost filma $g(x, y)$. Tako dobimo $E'(\xi, \eta) \propto \int \int g(x, y) e^{i\delta} dx dy$. Vpeljemo novi spremenljivki $u = 2\pi\xi/l\lambda$ in $v = 2\pi\eta/l\lambda$ z enoto m^{-1} , ki ju imenujemo krajevni frekvenci. Z njima dobimo:

$$E'(u, v) \propto \int \int g(x, y) e^{i(ux+vy)} dx dy. \quad (1)$$

Enačba pokaže, da kompleksni amplitudi $E'(u, v)$ in $g(x, y)$ povezuje Fourierova transformacija [7]. Gostoti svetlobnega toka na interferenčni sliki in v predmetni ravnini sta tedaj $j'(u, v) \propto |E'(u, v)|^2$ in $j(x, y) \propto |g(x, y)|^2$.

Pokaže se, da velja enačba, ki se samo po faznem faktorju z absolutno vrednostjo 1 pred integralom razlikuje od (1), tudi pri Fresnelovem načinu opazovanja [2]. Prepustnost $g(x, y)$ določimo z iteracijo po n korakih iz začetne prepustnosti $g_0(x, y)$. Pri nekaterih fraktalih, na primer pri Cantorjevi množici, je mogoče integral (1) zapisati kot produkt oblikovnega faktorja $\sin \pi v / \pi v$ in strukturnega faktorja [2], podobno kot smo vajeni pri rentgenskih interferencah. Prvi faktor ustreza uklonu na eni sami reži, drugi pa zajame zgradbo osnovne enote, na primer identitetne periode v mrežici iz dveh vrst rež [8].

Iz lastnosti Fourierove transformacije hitro ugotovimo nekatere značilnosti interferenčnih slik. Ker so fraktali sebi podobni, velja $j(x, y) \propto j(\kappa x, \kappa y)$. Za interferenčno sliko je tedaj $j'(u, v) \propto j'(u/\kappa, v/\kappa)$. Interferenčna slika je v tem primeru tudi sebi podobna. Nekaj več dela bi bilo treba vložiti, da bi pokazali, da je povprečna porazdelitev gostote svetlobnega toka po slikovni ravnini povezana s fraktalno dimenzijo $\langle j'((u^2 + v^2)^{1/2}) \rangle \propto (u^2 + v^2)^{-D/2}$ in da je fraktalna dimenzija interferenčne slike $2 - D$. Zadnjo zvezo lahko preskusimo pri predmetih, ki niso fraktali. Če je predmet točka z dimenzijo 0, je slika enakomerno osvetljen lik z dimenzijo 2, če je predmet daljica z dimenzijo 1, je slika pravokotna daljica z dimenzijo 1, če je predmet enakomerno svetel lik z dimenzijo 2, je slika točka z dimenzijo 0.

Profesor Kai-Erik Peiponen iz laboratorija Väisälä univerze Joensuu na Finskem je bil tako prijazen, da mi je poslal nekaj ponatiskov. On in profesor Jon Uozumi z Inštituta za elektronska raziskovanja univerze Hokkaido v Sapporu na Japonskem sta ljubeznivo dovolila ponatisniti fotografije iz prispevka [5], za kar se jima prisrčno zahvaljujem.

LITERATURA

- [1] L. Marek-Crnjac, *Sebi podobnost in dimenzija fraktalov*, Obzornik mat. fiz. **41** (1994) 144.
- [2] C. Allain, M. Cloitre, *Optical diffraction on fractals*, Phys. Rev. B **33** (1986) 3566.
- [3] J. Uozumi, H. Kimura, T. Asakura, *Fraunhofer diffraction by Koch fractals*, J. Mod. Opt. **37** (1990) 1011; J. Uozumi, H. Kimura, T. Asakura, *Fraunhofer diffraction by Koch fractals: the dimensionality*, J. Mod. Opt. **38** (1991) 1335.
- [4] D. Berger, S. Chamaly, M. Perreau, D. Mercier, P. Monceau, J.-C. S. Levy, *Optical diffraction of fractal figures: random Sierpinski carpets*, Journal de Physique **1** (1991) 1443.
- [5] J. Uozumi, T. Asakura, K.-E. Peiponen, R. Silvennoinen, M. Savolainen, *Teaching diffraction through fractals*, Education and Training in Optics, Satellite Conference to the 16th Congress of the International Commission for Optics, Pécs 1993, str. 139.
- [6] J. Uozumi, K.-E. Peiponen, M. Savolainen, R. Silvennoinen, T. Asakura, *Demonstration of diffraction by fractals*, Am. J. Phys. **62** (1994) 283.
- [7] G. R. Fowles, *Introduction to Modern Optics*, Holt, Rinehart, and Winston, New York 1968, str. 133.
- [8] J. Strnad *Moderna fizika*, Univerza v Ljubljani, Ljubljana 1972, str. 48, 84.

PROGRAMSKI JEZIK OBERON: NOV PRISTOP K OBJEKTNO USMERJENEMU PROGRAMIRANJU

VILJAN MAHNIČ

Math. Subj. Class. (1991) 68N05, 68-01, 68-04

V tem sestavku pokažemo, kako je s pomočjo programskega jezika oberon moč vpeljati osnovne prijeme objektno usmerjenega programiranja kot enostavno razširitev že znanih konceptov proceduralnega programiranja. Opisana sta razširitev podatkovnega tipa in podprogramski podatkovni tip, nato pa so s pomočjo konkretnega primera prikazani osnovni koncepti objektno usmerjenega programiranja: generični moduli, heterogene podatkovne strukture, pomen pravilne razdelitve programa na module in dedovanje metod.

PROGRAMMING LANGUAGE OBERON: A NEW APPROACH TO OBJECT-ORIENTED PROGRAMMING

In this note it is shown how the use of the Oberon language makes it possible to introduce basic object-oriented programming concepts as simple extensions of well-known concepts of procedural programming. Concepts of type extension and procedure type are described first, and an example is used to illustrate the basic object-oriented programming concepts: generic modules, heterogeneous data structures, the importance of appropriate module structure, and the inheritance of methods.

1. Uvod

Objektno usmerjeno programiranje (angl. Object Oriented Programming ali na kratko OOP) je nova tehnika programiranja, ki namenja največ pozornosti ponovni uporabi že sprogramiranih rešitev in dodajanju novih funkcij brez spreminjanja kode obstoječih programskih modulov. Najbolj vneti zagovorniki OOP pojmujejo to tehniko kot nekaj povsem novega in drugačnega od običajnega proceduralnega programiranja. V nasprotju z njimi pa skušajo avtorji programskega jezika oberon pokazati, da je moč OOP vpeljati na enostaven način kot nadgradnjo konceptov, ki jih že poznamo iz proceduralnega programiranja. Ta pristop se nam zdi zanimiv zlasti pri učenju osnov OOP, saj ne negira prejšnjega znanja, ampak nasprotno: uporablja znanje proceduralnega programiranja kot osnovo za prehod na OOP.

Programski jezik oberon [1] [2] je nastal na ETH v Zürichu v drugi polovici osemdesetih let in je zgodovinsko gledano naslednik pascala in module-2¹. Od pascala je podedoval strukturiranost, od module-2 modularnost, na novo pa je bila dodana ti. *razširljivost*, tj. možnost kasnejšega povezovanja dodatnih podatkov in podprogramov s tistimi, ki so bili v kakem programskem modulu deklarirani že prej. Razširljivost temelji na uvedbi dveh novih konceptov: *koncepta razširitve podatkovnega tipa* in *koncepta podprogramskega tipa*². Ta dva koncepta omogočata pisanje takih program-

¹ Različne verzije oberona, ki so bile razvite na ETH, so brezplačne in jih je moč dobiti po anonimnem ftp-ju na naslovu <ftp://inf.ethz.ch/pub/oberon>. Informacije o oberonu so na voljo po internetu na naslovu <http://www-cs.inf.ethz.ch/Oberon.html>.

² Dejansko je v oberonu nov samo koncept razširitve podatkovnega tipa, saj je podprogramski tip poznala že modula-2.

skih modulov, v katerih vsi podatki in podprogrami še niso natančno opredeljeni. Dodatne deklaracije posredujejo kasneje tisti programski moduli, ki so specifični za konkretno obdelavo. Razširitev podatkovnega tipa omogoča dodajanje novih podatkov, uvedba podprogramskega tipa pa dodajanje novih podprogramov.

V članku bomo najprej podrobneje opisali tadva koncepta, nato pa bomo prikazali temeljne značilnosti OOP v oberonu, ne da bi pri tem uporabili običajno terminologijo OOP. Šele na koncu bomo vzpostavili povezavo med pojmi, ki jih uporablja oberon, in terminologijo, ki je značilna za OOP. Članek bomo sklenili s prikazom dedovanja metod s poudarkom na novostih, ki jih prinaša oberon-2.

2. Razširitev podatkovnega tipa

Razširitev podatkovnega tipa se nanaša na zapise (angl. records) in kazalce (angl. pointers), ki kažejo nanje. Pri tem je pomembno, da ostanejo razširjeni podatkovni tipi še vedno skladni z osnovnimi podatkovnimi tipi, iz katerih so bili izpeljani.

2.1. Razširitev zapisov

Koncept razširitve zapisov omogoča, da obstoječemu zapisu dodamo nove komponente in tako sestavljamo nove zapise. Recimo, da zapis tipa *Vozilo* vsebuje osnovne podatke o nekem vozilu: registrsko številko in proizvajalca.

```
TYPE Niz=ARRAY 15 OF CHAR;
      Vozilo=RECORD
          regSt:Niz;
          proizvajalec:Niz
      END;
```

Potem lahko deklariramo razširitev tega tipa, ki hrani podatke o avtobusih:

```
TYPE Avtobus=RECORD (Vozilo)
      stSedezev:INTEGER
  END;
```

Vsak zapis tipa *Avtobus* sedaj vsebuje tri komponente: poleg komponent *regSt* in *proizvajalec*, ki sta bili deklarirani v okviru tipa *Vozilo*, še komponento *stSedezev* (število sedežev), ki je bila dodana ob razširitvi.

Na podoben način lahko deklariramo razširitev za podatke o tovornjaki. Recimo, da je tovornjak vozilo, za katero je poleg registrske številke in proizvajalca pomemben še podatek o nosilnosti. Temu ustreza naslednja razširitev tipa *Vozilo*:

```
TYPE Tovornjak=RECORD (Vozilo)
      nosilnost:REAL
  END;
```

Za tip *Vozilo* pravimo, da je *osnovni tip*, medtem ko sta tipa *Avtobus* in *Tovornjak* *razširitvi* tega tipa.

Pri prirejanju vrednosti velja, da je vsak razširjen tip skladen (angl. assignment compatible) z osnovnim tipom, iz katerega je bil izpeljan (ne pa tudi obratno). Če imamo spremenljivke v , a in t , ki so deklarirane kot:

```
VAR v:Vozilo;  
    a:Avtobus;  
    t:Tovornjak;
```

lahko priredimo $v:=a$ in $v:=t$. Tip izraza na desni strani vsakega od teh stavkov je razširitev tipa spremenljivke, ki nastopa na levi, zato vse komponente, ki sestavljajo zapis na levi, pozna tudi spremenljivka na desni. Niso pa dopustna prirejanja $a:=v$, $t:=v$, $a:=t$ in $t:=a$, saj v vsakem od navedenih primerov zapis na levi vsebuje tudi komponente, ki jih ne najdemo v spremenljivki na desni.

Prirejanje $v:=a$ opravi isto kot stavka

```
v.regSt:=a.regSt;  
v.proizvajalec:=a.proizvajalec;
```

kjer je prirejanje zapisano po posameznih komponentah. Iz primera je lepo razvidno, da pri prirejanju sodelujejo samo komponente, ki nastopajo v osnovnem tipu, medtem ko se vrednosti komponent, ki so bile dodane ob razširitvi, izgubijo.

Milejše zahteve glede skladnosti zapisov veljajo tudi pri zamenjavi formalnih parametrov z dejanskimi, kadar gre za prenos po referenci. Takrat lahko formalni parameter, ki je osnovnega tipa, zamenjamo z dejanskim, ki pripada neki razširitvi tega tipa. Tako lahko npr. podprogram Proc1

```
PROCEDURE Proc1(VAR x:Vozilo);
```

pokličemo kot Proc1(v) ali Proc1(a) ali Proc1(t).

2.2. Kazalci na razširjene zapise

Podobna pravila glede razširitve in skladnosti tipov veljajo tudi za kazalce na razširjene zapise. Tip kazalca, ki kaže na razširjen zapis, je razširitev tipa kazalca, ki kaže na osnovni zapis. Upoštevajoč prej navedene deklaracije tipov Vozilo, Avtobus in Tovornjak, lahko vpeljemo naslednje kazalčne tipe in spremenljivke:

```
TYPE KVozilo=POINTER TO Vozilo;  
    KAvtobus=POINTER TO Avtobus;  
    KTovornjak=POINTER TO Tovornjak;  
VAR kv:KVozilo;      (* kazalec na zapis tipa Vozilo *)  
    ka:KAvtobus;      (* kazalec na zapis tipa Avtobus *)  
    kt:KTovornjak;    (* kazalec na zapis tipa Tovornjak *)
```

Tudi sedaj velja, da sta tipa KAvtobus in KTovornjak razširitvi osnovnega tipa KVozilo, zato sta dopustni prirejanji $kv:=ka$ in $kv:=kt$.

Po izvršitvi prireditvenega stavka $kv:=ka$ kazalec kv ne kaže več na zapis tipa Vozilo, ampak na zapis tipa Avtobus. Zato v obergeru govorimo o dveh tipih kazalca: statičnem in dinamičnem. *Statični tip* je tisti, ki smo ga določili ob deklaraciji, *dinamični tip* pa je lahko neka razširitev statičnega

tipa, do katere pride med izvajanjem programa. V našem primeru je kazalec kv statičnega tipa KVožilo, po izvršitvi stavka kv:=ka pa postane njegov dinamični tip KAvtobus. Zato lahko z njegovo pomočjo dosežemo tudi tiste komponente zapisa, ki so bile dodane ob razširitvi, vendar moramo ob uporabi kazalca eksplicitno navesti njegov dinamični tip. Določila posameznih komponent zapisa, na katerega kaže kazalec kv, ki je dinamičnega tipa KAvtobus, zapišemo kot:

```
kv.regSt3
kv.proizvajalec
kv(KAvtobus).stSedezev
```

Zadnje določilo vsebuje t.i. *kontrola tipa* (angl. type guard), ki pove, da je dostop do komponente stSedezev možen samo, če je kazalec kv dinamičnega tipa KAvtobus. V nasprotnem primeru pride do napake, ki povzroči nasilno prekinitev izvajanja programa. Zato je priporočljivo, da pred uporabo kontrole tipa preverimo, kakšen je dinamični tip kazalca. V ta namen imamo v oberonu na voljo operator IS. Vrednost izraza

```
kv IS KAvtobus
```

je TRUE, kadar je dinamični tip kazalca kv razširitev tipa KAvtobus, tj. enak KAvtobus ali neki razširitvi tega tipa.

3. Podprogramski podatkovni tip

Podprogramski podatkovni tip omogoča, da deklariramo spremenljivke, katerih vrednost je zaporedje stavkov, ki pripadajo nekemu podprogramu. Ob deklaraciji podprogramskega tipa moramo navesti seznam formalnih parametrov in (če gre za funkcijski podprogram) tip rezultata, npr.:

```
TYPE RealnaFunkcija=PROCEDURE (x:REAL):REAL;
```

Zalogo vrednosti tipa RealnaFunkcija tvorijo vsi funkcijski podprogrami z enim samim parametrom tipa REAL, ki se zamenja po vrednosti, in rezultatom tipa REAL. Če sedaj deklariramo spremenljivko f in funkcijski podprogram Kvadrat

```
VAR f:RealnaFunkcija;

PROCEDURE Kvadrat(x:REAL):REAL;
BEGIN
  RETURN x*x
END Kvadrat;
```

lahko spremenljivki f priredimo vrednost f:=Kvadrat. Za prireditveni stavek

```
ImePodprogramskeSpremenljivke := ImePodprograma
```

³ Oberon dopušča, da operator ↑, ki označuje uporabo kazalca, izpustimo, kadar kazalec uporabljamo za dostop do komponente zapisa ali elementa tabele. Zato kv.regSt pomeni isto kot kv↑.regSt.

velja, da je ImePodprograma ime nekega podprograma ali ime neke druge podprogramske spremenljivke. Prirejanje je dopustno samo, če se tip izraza na desni sklada s tipom določila na levi strani prireditvenega stavka, tj. oba tipa se morata ujemati v številu parametrov in tipu rezultata, istoležni parametri pa morajo biti istega tipa in se morajo prenašati na enak način.

Podprogram, ki smo ga priredili podprogramski spremenljivki, lahko pokličemo tako, da napišemo ime podprogramske spremenljivke in v oklepaju navedemo dejanske parametre. V našem primeru lahko izvajanje podprograma Kvadrat prožimo s klicem `f(3.15)`.

4. Ključni koncepti objektno usmerjenega programiranja v oberonu

Ključne koncepte objektno usmerjenega programiranja v oberonu bomo prikazali s pomočjo konkretnega primera: napisali bomo program za vzdrževanje podatkov o vozilih, ki bo omogočal hranjenje podatkov za različne vrste vozil. Pri tem bomo največ pozornosti namenili dvema problemoma:

- pisanju splošno uporabnih modulov, ki so neodvisni od konkretne obdelave;
- razdelitvi programa na module, ki bo omogočala enostavno dodajanje novih funkcij brez spreminjanja kode obstoječih modulov.

4.1. Generični moduli

V oberonu je vsak program praviloma sestavljen iz več modulov. Nekatere od teh modulov vsebujejo deklaracije, ki so na razpolago tudi drugim modulom. Zato jim rečemo *moduli strežniki*, z izrazom *moduli odjemalci* pa označujemo tiste module, ki uporabljajo „usluge” strežnikov. Vpeljava razširitve podatkovnega tipa in podprogramskih spremenljivk omogoča pisanje takih modulov strežnikov, ki so splošno uporabni, tj. neodvisni od konkretne aplikacije. Zaradi svoje splošnosti imajo ti moduli posebno ime: pravimo jim *generični moduli*.

Kot primer takega modula bomo prikazali modul za vzdrževanje seznamov s podatki poljubnega tipa. Da ne bi zabredli v nepomembne podrobnosti, si bomo podrobneje ogledali samo nekaj operacij: dodajanje novega elementa na začetku seznama, odvzemanje elementa z začetka seznama in iskanje elementa v primeru, ko poznamo njegov ključ. Seznam bomo realizirali s pomočjo kazalcev kot abstrakten podatkovni tip, zato potrebujemo naslednje deklaracije:⁴

```
TYPE Kazalec*=POINTER TO Element;  
    Element*=RECORD  
        vezava:Kazalec  
    END;  
Seznam*=RECORD  
    zacetek:Kazalec  
    END;
```

⁴ Z zvezdico so označene tiste deklaracije, ki so dostopne tudi v drugih moduli.

Z deklaracijo tipa `Element` še ni določeno, kakšnega tipa so podatki, ki bodo shranjeni v seznamu. Dejanski tip podatkov deklarira šele modul odjemalec, tako da razširi tip `Element` s tistimi podatki, ki so specifični za njegovo obdelavo. Pač pa komponenta vezava zadostuje, da v generičnem modulu sprogramiramo operaciji dodajanja in odvzemanja elementov.

Nekoliko bolj zapleteno je iskanje elementa v seznamu. Sprogramirajmo ga kot funkcijski podprogram

```
PROCEDURE Poisci*(s:Seznam; VAR klj:Kljuc):Kazalec;
```

ki vrne kazalec na tisti element seznama, katerega ključ je enak vrednosti parametra `klj`. Če elementa z iskanim ključem ni v seznamu, podprogram vrne `NIL`. Naloga bi bila enostavna, če bi lahko v tipu `Element` poleg komponente vezava deklarirali še komponento `kljuc`, v kateri bi bil shranjen ključ vsakega elementa. Vendar bi s tem vnaprej predpisali tip ključa in tako izgubili na splošnosti generičnega modula.

Iz zadrege nas reši uporaba razširitve tipa in podprogramskih spremenljivk. V generičnem modulu zadostuje, da tip ključa deklariramo kot zapis brez komponent, za medsebojno primerjanje ključev pa vpeljemo podprogramsko spremenljivko `niEnak`:

```
Kljuc*=RECORD END;
```

```
VAR niEnak*:PROCEDURE(k:Kazalec; VAR klj:Kljuc):BOOLEAN;
```

Da bi opisana rešitev delovala, je treba kasneje v vsakem modulu odjemalcu deklarirati dejanski tip ključa kot ustrezno razširitev tipa `Kljuc` in prirediti spremenljivki `niEnak` ime podprograma, ki omogoča medsebojno primerjavo iskanega ključa `klj` s ključem tistega elementa, na katerega kaže kazalec `k`.

Rezultat teh razmislekov je generični modul `SeznamiG`, ki poleg že omenjenih operacij dodajanja, odvzemanja in iskanja vsebuje tudi nekaj pomožnih operacij (inicializacija, kazalec na prvi element, kazalec na naslednji element). Pomni, da oberon uporablja znak `#` za operator „ni enak”.

```
MODULE SeznamiG;
```

```
TYPE
```

```
    Kljuc*=RECORD END;
```

```
    Kazalec*=POINTER TO Element;
```

```
    Element*=RECORD
```

```
        vezava:Kazalec
```

```
    END;
```

```
    Seznam*=RECORD
```

```
        zacetek:Kazalec
```

```
END;
```

```
VAR niEnak*:PROCEDURE(x:Kazalec; VAR klj:Kljuc):BOOLEAN;
```

```
PROCEDURE DodajZ*(VAR s:Seznam; n:Kazalec);
```

```
BEGIN
```

```
    n.vezava:=s.zacetek;
```

```
    s.zacetek:=n;
```

```
END DodajZ;
```

```

PROCEDURE OdvzemiZ*(VAR s:Seznam):Kazalec;
VAR k:Kazalec;
BEGIN
    k:=s.zacetek;
    IF s.zacetek # NIL THEN
        s.zacetek:=s.zacetek.vezava
    END;
    RETURN k
END OdvzemiZ;

PROCEDURE Poisci*(s:Seznam; VAR klj:Kljuc):Kazalec;
VAR p:Kazalec;
BEGIN
    p:=s.zacetek;
    WHILE (p # NIL) & niEnak(p,klj) DO
        p:=p.vezava
    END;
    RETURN p
END Poisci;

PROCEDURE Odpri*(VAR s:Seznam);
BEGIN
    s.zacetek:=NIL
END Odpri;

PROCEDURE Prvi*(s:Seznam):Kazalec;
BEGIN
    RETURN s.zacetek
END Prvi;

PROCEDURE Naslednji*(k:Kazalec):Kazalec;
BEGIN
    RETURN k.vezava
END Naslednji;

END SeznamIG.

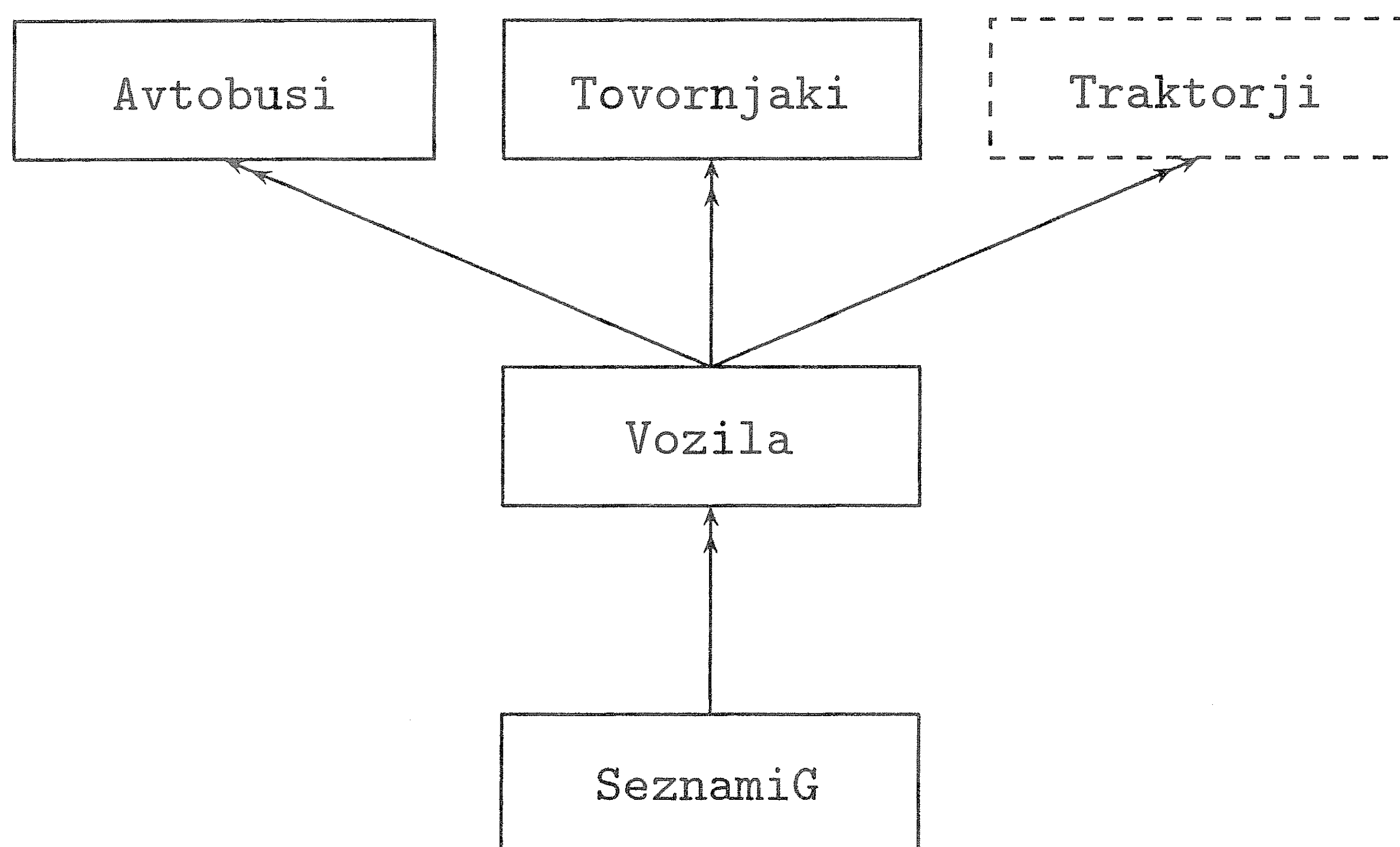
```

4.2. Heterogene podatkovne strukture in razdelitev programa na module

Prednosti objektno usmerjenega programiranja pridejo še posebej do izraza, kadar moramo v programu vzdrževati *heterogene podatkovne strukture*. V oberoenu so to dinamične podatkovne strukture, v katerih posamezne komponente pripadajo različnim tipom, ki pa so vsi izpeljani iz istega osnovnega tipa. V takih primerih lahko s primerno razdelitvijo programa na module dosežemo, da pri dodajanju nove vrste komponent ni potrebno spremeniti kode obstoječih modulov. Oglejmo si to na primeru seznama, v katerem nastopajo tri vrste vozil:

- „navadna” vozila, za katera hranimo podatke o registrski številki, proizvajalcu in datumu poteka registracije;
- avtobusi (poleg podatkov, značilnih za navadna vozila, hranimo tudi podatek o številu sedežev);
- tovornjaki (zanje dodamo še podatek o nosilnosti).

Kot izhodišče za realizacijo tega seznama lahko uporabimo generični modul `SeznamiG`, deklaracije podatkov in podprogramov, ki so specifični za posamezne vrste vozil, pa zapišemo ločeno v moduli `Vozila`, `Avtobusi` in `Tovornjaki`, tako kot to prikazuje slika 1. Modul `Vozila` omogoča delo z navadnimi vozili, poleg tega pa lahko v njem realiziramo tiste operacije, ki so skupne za vse vrste vozil. Modul `Avtobusi` vsebuje deklaracije, ki so potrebne za delo z avtobusi, modul `Tovornjaki` pa deklaracije za delo s tovornjaki.



Slika 1. Razdelitev programa na module

Opisana razdelitev programa na module omogoča, da vpeljemo novo vrsto vozil, npr. traktorje, ne da bi spremenili kodo obstoječih modulov `SeznamiG`, `Vozila`, `Avtobusi` in `Tovornjaki`. Dodati je treba le modul `Traktorji`, ki je po svoji zgradbi podoben moduloma `Avtobusi` in `Tovornjaki`.

4.2.1. Modul `Vozila`

V modulu `Vozila` je treba najprej razširiti tipa `Kljuc` in `Element`, ki ju dobimo iz modula `SeznamiG`, ter deklarirati spremenljivko, ki ponazarja abstraktni seznam vozil. Recimo, da želimo za vsako vozilo poleg zahtevanih podatkov definirati tudi dve operaciji: izpis podatkov o vozilu in podaljšanje veljavnosti registracije. Potem izgledajo ustrezne deklaracije takole:⁵

⁵ Oberon zahteva, da pri uporabi imen, ki so deklarirana v drugih moduli, pred vsakim imenom navedemo še ime modula. Ker sta v našem primeru tipa `Kljuc` in `Element`, ki ju želimo razširiti, deklarirana v modulu `SeznamiG`, se moramo v modulu `Vozila` sklicevati nanju z opredeljenima imenoma `SeznamiG.Kljuc` oziroma `SeznamiG.Element`. Isto velja za tip `Seznam`, ki nastopa v deklaraciji spremenljivke `sv`.

```

TYPE Niz*=ARRAY 15 OF CHAR;
   Datum*=RECORD
       dan*,mesec*,leto*:INTEGER
   END;
   Kljuc=RECORD (SeznamiG.Kljuc) (* dejanski tip kljuca *)
       regSt*:Niz
   END;
   KVozilo*=POINTER TO Vozilo;    (* tip kazalcev na razsirjene zapise *)
   Vozilo*=RECORD (SeznamiG.Element)    (* razsiritev tipa Element *)
       kljuc*:Kljuc;
       proizvajalec*:Niz;
       datumReg*:Datum;
       izpis*:PROCEDURE (k:KVozilo);
       podaljsanje*:PROCEDURE (k:KVozilo)
   END;

```

```

VAR sv*:SeznamiG.Seznam;          (* seznam vozil *)

```

Da bi lahko uporabili operacijo iskanja, ki smo jo sprogramirali v modulu SeznamiG, je treba deklarirati podprogram za primerjanje iskanega ključa s ključem v posameznih elementih seznama in ga prirediti spremenljivki SeznamiG.niEnak. Seznam parametrov tega podprograma se mora ujemati s seznamom parametrov, ki je bil naveden ob deklaraciji spremenljivke SeznamiG.niEnak.

Zato je treba za dostop do komponent, ki so bile dodane ob razširitvi, uporabiti kontrolo tipa:

```

PROCEDURE NiEnak*(k:SeznamiG.Kazalec; VAR klj:SeznamiG.Kljuc):BOOLEAN;
BEGIN
    RETURN k(KVozilo).kljuc.regSt # klj(Kljuc).regSt
END NiEnak;

```

Deklarirati je treba tudi podprograma za izpis podatkov o vozilu in podaljšanje veljavnosti registracije, ki ustrezata podprogramskima komponentama izpis in podaljsanje. Predpostavimo lahko, da vsakemu vozilu podaljšamo veljavnost registracije za eno leto:

```

PROCEDURE IzpisiVozilo*(k:KVozilo);
BEGIN
    Out.String(k.kljuc.regSt);
    Out.String(k.proizvajalec);
    Out.Int(k.datumReg.dan,2); Out.String(".");
    Out.Int(k.datumReg.mesec,2); Out.String(".");
    Out.Int(k.datumReg.leto,4)
END IzpisiVozilo;

PROCEDURE Podaljsaj*(k:KVozilo);
BEGIN
    INC(k.datumReg.leto);
    IF (k.datumReg.dan=29) & (k.datumReg.mesec=2) THEN
        k.datumReg.dan:=28
    END
END Podaljsaj;

```

Sedaj imamo na voljo vse deklaracije, ki jih potrebujemo, če želimo v seznam dodati novo vozilo. Dodajanje novega vozila zahteva, da najprej generiramo nov element seznama (tj. zapis tipa `Vozilo`), vpišemo vrednosti posameznih komponent in ga povežemo z elementi, ki so že v seznamu. Generiranje novega elementa in vpis vrednosti posameznih komponent mora opraviti modul `Vozila` (glej podprogram `DodajVozilo`, ki za branje podatkov uporablja pomožno proceduro `PreberiPodatke`). Operacije nad kazalci, ki so potrebne pri povezovanju novega elementa v seznam, pa prepustimo generičnemu modulu `SeznamiG`.

```
PROCEDURE PreberiPodatke*(k:KVozilo);
BEGIN
    In.Open;
    In.String(k.kljuc.regSt);
    In.String(k.proizvajalec);
    In.Int(k.datumReg.dan);
    In.Int(k.datumReg.mesec);
    In.Int(k.datumReg.letto)
END PreberiPodatke;

PROCEDURE DodajVozilo*;
(* na konec seznama doda podatke za naslednje vozilo *)
VAR k:KVozilo;
BEGIN
    NEW(k);                (* generiranje novega elementa *)
    PreberiPodatke(k);
    k.izpis:=IzpisiVozilo;
    k.podaljsanje:=Podaljsaj;
    SeznamiG.DodajK(sv,k)  (* dodajanje na konec seznama *)
END DodajVozilo;
```

Opisana razdelitev programa na module nam omogoča, da v modulu `Vozila` realiziramo nekatere operacije, ki so skupne za vsa vozila, ne da bi poznali specifične lastnosti posameznih vrst vozil. Podprogrami, s katerimi so te operacije realizirane, se ne spremenijo, četudi v seznam dodamo neko novo vrsto vozil. Ogledali si bomo dve taki operaciji: izpis podatkov o vseh vozilih v seznamu (podprogram `IzpisiVse`) in podaljšanje veljavnosti registracije za izbrano vozilo (podprogram `PodaljsajReg`):

```
PROCEDURE IzpisiVse*;
VAR k:SeznamiG.Kazalec;
BEGIN
    k:=SeznamiG.Prvi(sv);
    WHILE k # NIL DO
        k(KVozilo).izpis(k(KVozilo));
        Out.Ln;
        k:=SeznamiG.Naslednji(k)
    END
END IzpisiVse;
```

```

PROCEDURE PodaljsajReg*;
VAR k:Seznamig.Kazalec;
    klj:Kljuc;
BEGIN
    In.Open;
    In.String(klj.regSt);
    k:=Seznamig.Poisci(sv,klj);
    IF k # NIL THEN
        k(KVozilo).podaljsanje(k(KVozilo))
    END
END PodaljsajReg;

```

Opozoriti velja na deklaracijo tipa kazalca k. Funkcijska podprograma Seznamig.Prvi in Seznamig.Poisci vrneta vrednost tipa Seznamig.Kazalec, zato mora biti tudi kazalec k tega tipa, čeprav je njegov dinamični tip drugačen. To pomeni, da je za dostop do komponent izpis in podaljsanje potrebna kontrola tipa, tj. k(KVozilo).izpis(k(KVozilo)) oziroma k(KVozilo).podaljsanje(k(KVozilo)).

4.2.2. Modula Avtobusi in Tovornjaki

Moduli, ki so specifični za posamezne vrste vozil, so zgrajeni na enak način. V splošnem morajo vsebovati:

- deklaracijo ustrezne razširitve tipa Vozilo;
- deklaracije tistih podprogramov, ki so drugačni kot za navadna vozila;
- deklaracijo podprograma za dodajanje novega vozila.

V modulu Avtobusi je razširitev tipa Vozilo preprosta. Dodati je treba samo komponento stSedezev (število sedežev):

```

TYPE
    KAvtobus=POINTER TO Avtobus;
    Avtobus=RECORD (Vozila.Vozilo)
        stSedezev:INTEGER
    END;

```

Podprogram za izpis podatkov o avtobusu, ki ga priredimo podprogramski komponenti izpis, moramo deklarirati na novo. Pri tem lahko kot osnovo uporabimo podprogram IzpisiVozilo iz modula Vozila. Dodati je treba le še izpis števila sedežev:

```

PROCEDURE IzpisiVozilo(k:Vozila.KVozilo);
BEGIN
    Vozila.IzpisiVozilo(k);
    Out.Int(k(KAvtobus).stSedezev,3)
END IzpisiVozilo;

```

Ob predpostavki, da morajo avtobusi vsakih šest mesecev na tehnični pregled, moramo na novo deklarirati tudi podprogram Podaljsaj. Vsakemu avtobusu podaljšamo veljavnost registracije samo za šest mesecev. Nekoliko poenostavljen podprogram Podaljsaj, ki ne upošteva, da imajo posamezni meseci različno število dni, bi lahko bil takšen:

```

PROCEDURE Podaljsaj*(k:Vozila.KVozilo);
BEGIN
  INC(k.datumReg.mesec,6);
  IF k.datumReg.mesec > 12 THEN
    k.datumReg.mesec := k.datumReg.mesec MOD 12;
  INC(k.datumReg.letost);
  END
END Podaljsaj;

```

Podprogram za dodajanje avtobusa v seznam vozil opravlja iste naloge kot podprogram DodajVozilo za navadna vozila: generiranje novega elementa, prirejanje vrednosti posameznim komponentam in dodajanje elementa na konec seznama. Paziti moramo, da je nov element tipa Avtobus, pri branju podatkov pa si lahko pomagamo s podprogramom PreberiPodatke iz modula Vozila.

```

PROCEDURE DodajAvtobus*;
(* na konec seznama doda podatke za avtobus *)
VAR k:KAvtobus;
BEGIN
  NEW(k);
  Vozila.PreberiPodatke(k);
  In.Int(k.stSedezev);
  k.izpis:=IzpisVozilo;
  k.podaljsanje:=Podaljsaj;
  SeznamiG.DodajK(Vozila.sv,k)
END DodajAvtobus;

```

Na enak način napišemo tudi modul Tovornjaki, le da nam ni treba ponovno deklarirati podprograma Podaljsaj. Ker je za tovornjake obvezen tehnični pregled samo enkrat na leto, lahko uporabimo kar podprogram Podaljsaj iz modula Vozila. Zadostuje torej, da v podprogramu za dodajanje novega tovornjaka priredimo komponenti podaljsanje vrednost `k.podaljsanje:=Vozila.Podaljsaj`.

5. Dedovanje metod

Značilnost oberona je, da uvaja koncepte objektno usmerjenega programiranja brez uporabe klasične terminologije OOP. Edini nov pojem, ki ga je treba vpeljati, je razširitev podatkovnega tipa. Ne glede na to pa lahko na tem mestu vzpostavimo povezavo z nekaterimi tipičnimi pojmi, ki jih srečamo v drugih objektno usmerjenih programskih jezikih.

Pojmu *objekt* ustreza v oberonu dinamično generiran zapis (v našem primeru je to en element seznama vozil), ki vsebuje tudi komponente podprogramskega tipa. Podprogramom, ki jih priredimo tem komponentam, lahko rečemo *metode*, klicem teh podprogramov pa *sporočila*. *Dinamično povezovanje podprogramov* je realizirano s prirejanjem vrednosti podprogramskim komponentam posameznega objekta. Te vrednosti določimo šele med izvajanjem programa, praviloma takrat, ko generiramo nov objekt (glej podpro-

grama DodajVozilo in DodajAvtobus). Med izvajanjem programa lahko vrednosti posameznih podprogramskih komponent spreminjamo in s tem dosežemo, da se obnašanje posameznih objektov dinamično spreminja.

5.1. Dedovanje metod v navadnem oberonu

Iz povedanega sledi, da je dedovanje metod realizirano nekoliko drugače kot v drugih objektno usmerjenih jezikih. Razširitev podatkovnega tipa, ki v oberonu približno ustreza pojmu *dedovanje*, omogoča, da razširjen zapis podeduje samo strukturo osnovnega zapisa, ne pa tudi metod, saj so le-te vezane na posamezen objekt in ne na njegov tip. To pomeni, da je treba ob generiranju vsakega objekta eksplicitno prirediti vrednost vsaki od podprogramskih komponent. Pri tem imamo dve možnosti:

- Neki podprogramski komponenti lahko priredimo podprogram, ki je deklariran v enem izmed modulov strežnikov. V tem primeru govorimo o dedovanju metode.
- Ob razširitvi tipa deklariramo tudi nov podprogram in ga priredimo podprogramski komponenti. V tem primeru govorimo o redefiniciji podprograma iz modula stržnika.

Za ilustracijo obeh možnosti rabi podprogram Podaljsaj. Modul Tovornjaki podeduje ta podprogram od modula Vozila, zato ob generiranju novega objekta tipa Tovornjak zadostuje prireditveni stavek `k.podaljsanje:=Vozila.Podaljsaj`. V modulu Avtobusi pa smo ta podprogram deklarirali na novo in njegovo vrednost priredili komponenti `podaljsanje` s stavkom `k.podaljsanje:=Podaljsaj`.

5.2. Dedovanje metod v oberonu-2

O pravem dedovanju metod, ki je primerljivo z drugimi objektno usmerjenimi programskimi jeziki, pa lahko govorimo v izpopolnjeni verziji oberona, imenovani oberon-2 [3]. Ta verzija omogoča, da posamezne podprograme (tj. metode) vežemo tudi na tip zapisa (tj. objekta) in ne več samo na podprogramske komponente v vsakem zapisu posebej. Še vedno pa so vsi podprogrami povezani dinamično. To pomeni, da se ob vsakem sporočilu (tj. klicu podprograma) preveri dinamični tip objekta, ki mu je sporočilo namenjeno. V odvisnosti od dinamičnega tipa objekta se potem izbere ustrezen podprogram (tj. metoda), ki se izvrši kot odgovor na prejeto sporočilo. Opisan pristop ima dve prednosti: ob kreiranju novega objekta ni več potrebna inicializacija podprogramskih komponent, zmanjša pa se tudi obseg pomnilnika, ki ga zasedajo posamezni objekti.

Sintaktično gledano mora vsaka metoda imeti poseben parameter, s katerim je določen objekt, ki mu je namenjeno sporočilo. Ta parameter – pravimo mu tudi *prejemnik* (angl. receiver) – nadomešča implicitni parameter `self` oziroma `this`, značilen za druge programske jezike. Biti mora kazalčnega tipa (tj. kazalec na objekt) ali tipa zapis (tj. objekt sam), deklariramo pa ga pred imenom metode, da se tako loči od (morebitnih) drugih parametrov. Ob deklaraciji zapisa, s katerim je objekt predstavljen

v računalniku, sedaj ni treba več deklarirati podprogramskih komponent. Oberon-2 namreč avtomatsko privzame, da nastopajo v vlogi metod vsi podprogrami, pri katerih je v glavi deklariran prej omenjeni prejemnik.

5.2.1. Spremembe v modulu Vozila

Za ilustracijo lahko vzamemo izsek iz modula Vozila, ki smo ga sedaj spremenili tako, da so podprogrami IzpisiVozilo, Podaljsaj in PreberiPodatke vezani na tip Vozilo in ne več na posamezne objekte tega tipa.

```
TYPE KVozilo*=POINTER TO Vozilo; (* tip kazalcev na razsirjene zapise *)
  Vozilo*=RECORD (SeznamIG.Element) (* razsiritev tipa Element *)
    kljuc*:Kljuc;
    proizvajalec*:Niz;
    datumReg*:Datum;
  END;

PROCEDURE (k:KVozilo) IzpisiVozilo*;
BEGIN
  (* nespremenjeno zaporedje stavkov za izpis *)
END IzpisiVozilo;

PROCEDURE (k:KVozilo) Podaljsaj*;
BEGIN
  (* nespremenjeno zaporedje stavkov za podaljsanje registracije *)
END Podaljsaj;

PROCEDURE (k:KVozilo) PreberiPodatke*;
BEGIN
  (* nespremenjeno zaporedje stavkov za branje *)
END PreberiPodatke;
```

V klasični terminologiji OOP lahko rečemo, da zgoraj navedene deklaracije predstavljajo *razred* Vozilo, v katerem imajo vsi objekti tri podatkovne komponente (kljuc, proizvajalec in datumReg) in tri metode (IzpisiVozilo, Podaljsaj in PreberiPodatke). Če je k statičnega tipa KVozilo, lahko vsako od teh metod pokličemo kot

k.IzpisiVozilo; k.Podaljsaj; k.PreberiPodatke;

torej drugače kot pri navadnem oberonu, kjer je bilo treba namesto imena metode navesti ime podprogramske komponente in dejanski parameter, npr.

k.izpis(k); k.podaljsanje(k);

V klicih metod k.IzpisiVozilo, k.Podaljsaj in k.PreberiPodatke je kazalec k obenem tudi dejanski parameter, s katerim je določen prejemnik sporočila, njegov dinamični tip pa določa, katera metoda se bo izvršila.

Če želimo podprograma IzpisiVse in PodaljsajReg prilagoditi novi deklaraciji tipa Vozilo, ki jo omogoča oberon-2, je treba spremeniti le klice posameznih metod v skladu s pravkar opisanim pravilom. Podprogram DodajVozilo se še dodatno poenostavi, ker odpade prirejanje vrednosti posameznim podprogramskim komponentam:

```

PROCEDURE DodajVozilo*;
  (* na konec seznama doda podatke za naslednje vozilo *)
VAR k:KVozilo;
BEGIN
  NEW(k); (* generiranje novega elementa *)
  k.PreberiPodatke;
  SeznamiG.DodajK(sv,k) (* dodajanje na konec seznama *)
END DodajVozilo;

```

Poudariti je treba, da za kreiranje novega objekta nimamo posebne metode, ampak navaden podprogram, kot je npr. v gornjem primeru podprogram DodajVozilo. V oberoenu namreč objektu, ki še ne obstaja, ne moremo poslati sporočila.

5.2.2. Spremembe v modulih Avtobusi in Tovornjaki

Modul Avtobusi sedaj avtomatsko podeduje vse tri metode, ki smo jih deklarirali v modulu Vozila. Žal pa nobena ni v celoti uporabna za delo z avtobusi, zato jih moramo deklarirati na novo z istim imenom in z enakim seznamom parametrov. Pri deklaraciji metod IzpisiVozilo in PreberiPodatke si lahko pomagamo z istoimenskima metodama iz modula Vozila. Pokličemo ju tako, da za imenom metode napišemo puščico ↑⁶.

```

PROCEDURE (k:KAvtobus) IzpisiVozilo*;
BEGIN
  k.IzpisiVozilo^;      (* izpisi podatke o vozilu *)
  Out.Int(k.stSedezev,3) (* in podatek, lasten avtobusu *)
END IzpisiVozilo;

```

```

PROCEDURE (k:KAvtobus) Podaljsaj*;
BEGIN
  INC(k.datumReg.mesec,6);
  IF k.datumReg.mesec > 12 THEN
    k.datumReg.mesec := k.datumReg.mesec MOD 12;
  INC(k.datumReg.letost);
END
END Podaljsaj;

```

```

PROCEDURE (k:KAvtobus) PreberiPodatke*;
BEGIN
  k.PreberiPodatke^;
  In.Int(k.stSedezev)
END PreberiPodatke;

```

Podprogram DodajAvtobus se poenostavi podobno kot podprogram DodajVozilo:

⁶ V ASCII naboru znakov je puščica predstavljena z znakom ^, ki ima kodo 94.

```

PROCEDURE DodajAvtobus*;
(* na konec seznama doda podatke za avtobus *)
VAR k:KAvtobus;
BEGIN
  NEW(k);
  k.PreberiPodatke;
  SeznamiG.DodajK(Vozila1.sv,k)
END DodajAvtobus;

```

V modulu Tovornjaki je treba izmed treh podedovanih metod ponovno deklarirati le dve: IzpisiVozilo in PreberiPodatke. Metodo Podaljsaj pa lahko uporabljamo brez sprememb.

6. Sklep

Članek smo napisali predvsem z namenom, da bi opozorili na oberonov pristop k objektno usmerjenemu programiranju, ki se od drugih programskih jezikov razlikuje po tem, da obravnava OOP kot enostavno nadgradnjo proceduralnega programiranja. Vse koncepte OOP lahko razložimo s pomočjo že znanih pojmov, vpeljati je treba le razširitev podatkovnega tipa in podprogramski tip.

Poleg tega je oberon strogo strukturiran in tipiziran jezik z enostavno in precizno sintakso, kar omogoča pisanje jasnih in razumljivih programov. Zato menimo, da je zelo primeren za učenje programiranja kot prvi programski jezik. Z njegovo pomočjo lahko bodočim programerjem vcepimo tiste principe lepega programiranja, ki se jih bodo držali tudi kasneje, ko bodo programirali v drugih programskih jezikih, ki so v praksi bolj razširjeni, a so za učenje principov lepega programiranja manj primerni (npr. C oziroma C++) [4]. V primerjavi s turbo pascalom in delphijem vidimo prednost oberona kot prvega učnega jezika v njegovi konciznosti: vsa sintaktična pravila, zapisana v razširjeni Backus Naurovi obliki, obsegajo samo eno stran. Zato ni treba izgubljati veliko časa za učenje sintakse, ampak se lahko posvetimo razlagi posameznih programskih konceptov, ki jih vpeljemo z manj novimi pojmi.

To nas je vodilo, da smo na Fakulteti za računalništvo in informatiko Univerze v Ljubljani pri predmetu Programiranje I, ki se predava v prvem semestru univerzitetnega študija, pričeli namesto pascala uporabljati oberon [5]. Uporaba oberona nam je omogočila, da v sklopu istega predmeta obravnavamo dve programski paradigmi: najprej proceduralno programiranje in nato objektno usmerjeno programiranje kot enostavno razširitev proceduralnega. Rezultati ankete, ki smo jo izvedli med študenti ob koncu predavanj, kažejo, da večina študentov podpira uvedbo OOP v začetni programerski tečaj in meni, da je oberonov pristop, ki gradi na že obstoječem znanju, boljši od pristopa v nekaterih drugih programskih jezikih (npr. smalltalk), ki obravnavajo OOP kot nekaj povsem novega, drugačnega od proceduralnega programiranja.

Zainteresirani bralci si lahko več o oberonu preberejo v ustrezni literaturi [2] [6] [7], pri avtorju pa lahko dobijo izvorno kodo zgledov, ki so bili prikazani v tem članku.

LITERATURA

- [1] N. Wirth, *The programming language Oberon*, Software Practice and Experience, 18, 671–690, 1988.
- [2] M. Reiser & N. Wirth, *Programming in Oberon – Steps beyond Pascal and Modula*, Addison-Wesley, 1992.
- [3] H. Mössenböck & N. Wirth, *The programming language Oberon-2*, Structured Programming, 12, 179–195, 1991.
- [4] C. J. Burgess & B. F. Jones, *Should software quality be a major issue when teaching first year programming to software engineers?*, in G. King, C. A. Brebbia, M. Ross & G. Staples (eds) *Software Engineering in Higher Education*, Computational Mechanics Publications, Southampton, 1994, 75–81.
- [5] V. Mahnič & B. Vilfan, *A first course in object-oriented programming using Oberon*, in J-L. Uso, P. Mitic, L. J. Sucharov (eds) *Software Engineering in Higher Education II*, Computational Mechanics Publications, Southampton 1995, 329–336.
- [6] V. Mahnič, *Programiranje v oberonu*, Založba Bi-TIM, Ljubljana, 1996.
- [7] H. Mössenböck, *Object-Oriented Programming in Oberon-2*, Springer-Verlag, 1993.

POPRAVEK

Pri stavljenju članka prof. A. Suhadolca *O integralški reprezentaciji logaritma*, ki je izšel v prejšnji številki Obzornika (str. 150), je nastalo nekaj tiskarskih napak. Avtorju in bralcem se opravičujemo za neljube pomote in objavljamo popravljene začetni del njegovega članka.

Uredništvo

Nedavno smo praznovali 200-letnico izida Vegovega velikega logaritmovnika *Thesaurusa*. Vega je računanje logaritmov oprl na danes vsakemu, ki se je kdaj vsaj malo ukvarjal z višjo matematiko, znano vrsto

$$\ln(1+x) = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots, \quad |x| < 1, \quad (1)$$

in na vrste, ki jih je izpeljal iz te s spretnimi substitucijami.

Ko je prof. Agata Tiegl pripravljala razstavo ob 200-letnici *Thesaurusa*, je opazila, da je vrsto (1) verjetno prvi našel Nikolaj Mercator. Objavil jo je leta 1668 v delu *Logarithmotechnia*. Izpeljavo moremo na kratko opisati takole: vrsto

$$\frac{1}{1+t} = 1 - t + t^2 - t^3 + \dots$$

po členih integriramo v mejah od 0 do x , $|x| < 1$, pa dobimo

$$\int_0^x \frac{1}{1+t} dt = x - \frac{x^2}{2} + \frac{x^3}{3} - \dots$$

O ZGODOVINI ALGEBRE

BORUT ZALAR

Math. Subj. Class. (1991): 01-01

Namen članka je poziv študentom matematike ter osnovno- in srednješolskim učiteljem matematike, da se začnejo bolj aktivno zanimati za zgodovino matematike. Ta vidik matematike je precej zanemarjen v naših učnih načrtih na vseh ravneh, čeprav bi lahko imel pomembno vlogo pri motiviranju in pravilnem dojemanju vloge matematike pri naših učencih.

ON THE HISTORY OF ALGEBRA

This note is an attempt to rise interest in the history of mathematics among students and high-school teachers in Slovenia. This aspects of mathematics is almost entirely neglected in our curriculum on all levels, yet it could be vital in providing motivation and coherent picture about mathematics to non-mathematicians.

Uvod

Pri delu z našimi učenci mnogokrat kršimo vsa načela psihologije in jih bombardiramo z aksiomskimi sistemi, ki delujejo, kot bi jih potegnili iz klobuka. Ni čudno, da se mnogi v sebi uprejo takemu „učinkovitemu“ načinu podajanja matematike, ki si ne more privoščiti izgube časa v teku za izgubljenim predmetnim načrtom. Če bi bile matematične ideje in strukture predstavljene kot pojmi, ki so naravno in ob pravem času vstopali v zgodovino človekove težnje po razumevanju sveta, ki nas obkroža, bi nam morda bolj pazljivo prisluhnili. To lahko naredimo samo, če bomo sami poznali zgodovino znanosti, s katero se ukvarjamo. Ta članek je pisan z mislijo, da je najlažje razumeti abstraktne algebraične strukture kot „števila“, katerih namen je podpirati zahtevnejšo analizo in geometrijo na isti način, kot realna števila podpirajo osnovni diferencialni račun in analitično geometrijo.

Zelo malo je znano o matematiki predgrških civilizacij. Nekateri najdbe (na primer dva večja papirusa iz starega Egipta) kažejo, da so vsaj nekatere med njimi morale imeti že kar napredne matematične ideje, ker drugače skoraj ni mogoče razložiti na primer dobrih približkov za število π , ki so jih poznali. Že samo spoznanje, da je obseg **vsakega** kroga sorazmeren z njegovim polmerom, je dosti **globlja** matematika, kot se nam to zdi danes, ko okrog nas letijo homologije, K-teorije, simplektične mnogoterosti in podobna čuda.

Tudi viri o starogrški matematiki so dosti bolj redki, kot si večina današnjih študentov matematike misli. Ob zatonu grških mest je bila večina klasičnih matematičnih del uničena. Rimski zgodovinarji poročajo, da v Grčiji nihče ni bil cenjen tako kot geometri (= matematiki), oni (Rimljani) pa so se nehali ukvarjati s to nesmiselno vedo, razen s tistim, kar je uporabno v praktičnem življenju. Ta genialna odločitev, zelo podobna izjavam današnjih bistrih načrtovalcev politične, ekonomske in sploh vsakršne prihodnosti celih narodov, je imela za posledico **tisočletni** zastoj celotne znanosti in tehnike. To se najbolje vidi, če beremo moderne prevode Arhimeda na

eni strani ter Descartesa, Newtona, Leibnitza in družine Bernoulli na drugi. Le-ti bi bili lahko Arhimédovi podiplomski študentje, ne pa zelo pomembni matematiki 17. stoletja.

Zelo malo Evropejcev se zaveda, da pomeni most med starogrško matematiko, fiziko in astronomijo ter predhodniki tehnične revolucije v Evropi islam. Islamski matematiki druge polovice 1. tisočletja so nadaljevali tudi hindujske, mezopotamske in egipčanske tradicije, za današnji razvoj evropske in ameriške matematike pa je bilo zlasti pomembno, da so prevedli v arabski in perzijski jezik skoraj vsa pomembna dela starih Grkov in ohranjali tudi njihovo znanje prek poučevanja. V zgodnjem srednjem veku je to znanje ponovno prišlo v mediteranski del Evrope. Ena od teorij pravi, da so bili začetniki tega procesa trgovci, ki so spoznali velike prednosti hindujsko-arabskega sistema zapisovanja števil. V tem obdobju je nastalo precej prevodov grške matematične klasike in njihovih arabskih predelav in izboljšav. Nekateri znani prevajalci so bili Gerard iz Cremone, Leonardo (Fibonacci) iz Pise in Herman iz Koroške, ki je bil najbrž slovenskega rodu, čeprav je o njegovem etničnem izvoru znano le to, da je bil Slovan. Najboljši algebraiki so bili precej časa na področju današnje Italije. V drugi polovici šestnajstega stoletja in v sedemnajstem stoletju se je center algebraičnih raziskav preselil v Francijo. Nekatera pomembna imena so Viete, Fermat, Descartes. Kasneje je postala matematika (in njene veje) spoštovana veda v vsej Evropi in pomembna algebraična odkritja so se po njej precej enakomerno geografsko razporedila.

Geometrijska algebra

Danes so osnove teorije števil del dodiplomske algebre; o celih številih govorimo kot o kolobarju, o racionalnih številih kot o obsegu itd. To je zelo daleč od starogrške matematike. Število ima pri pitagorejcih delno filozofski, delno celo mistični pomen. O tem obstaja veliko legend, iz katerih je težko razbrati približno resnico. Ohranjeni pisni viri kažejo, da so jih najbrž predvsem zanimala nenavadna dejstva (pari zaporednih praštevil, perfektna števila,...) manj pa globalna struktura. Za začetek algebre štejejo zato moderni zgodovinarji ponavadi poskuse reševanja enačb. Tudi naša beseda algebra (al džabr) je prišla iz naslova neke srednjeveške arabske knjige o reševanju enačb. Slovenski prevod tega naslova je „veda o redukciji in krajšanju“. Te enačbe so v začetku prihajale iz geometrije. Če je x neka količina, potem x^2 predstavlja ploščino kvadrata s stranico x . Mnogi stari problemi ravninske geometrije vodijo h kvadratni enačbi, tisti iz geometrije teles pa h kubični. Največkrat so jih reševali s pomočjo skic in geometrijskega razmišljanja. Tak pristop mogoče razloži tudi dejstvo, da stari Grki **niso** uporabljali negativnih količin (ker niso dolžine daljic) in **najbrž niso** nikoli poznali splošne rešitve kvadratne enačbe. Iz obstoječih prevodov njihov del se vidi, da so najbrž razumeli enačbi

$$ax^2 + bx = c,$$

$$ax^2 + c = bx$$

kot bistveno različni. Tudi postopek rešitve je **različen**, ker se tudi v postopku očitno niso smele pojavljati negativne količine. Ta pristop so uporabljali tudi islamski in krščanski avtorji pred renesanso. Geometrijski pristop je najbrž tudi vzrok, da niso obravnavali enačb poljubne stopnje (ker ne predstavljajo ne ploščine ne volumna) in da so znali rešiti kubično enačbo le v posebnih primerih (ker je težko risati tridimenzionalne skice). Bistvena ovira pri razvoju starogrške algebre v moderno smer je bilo pomanjkanje algebraičnega simbolizma v smislu „naj bo $x \dots$ “. Kvadratno enačbo so opisali s celim odstavkom teksta in rešitve niso mogli ponazoriti drugače kot na posebnih primerih, saj niso nikoli zapisali „naj bodo a, b, c poljubni parametri kvadratne enačbe $\dots$ “. Čeprav so imeli stari Grki izjemno globoke ideje, jim je torej včasih manjkal drugi člen na desni strani ekvivalence:

maksimalen napredek pri reševanju problema $\equiv$
dobra ideja + optimalni simbolizem za njen opis.

Srednji vek in renesansa v Evropi

V srednjeveški Italiji je reševanje enačb postalo nekakšen šport. Prirejali so celo tekmovanja, z izzivalci in zanimanjem občinstva, v smislu „kdo zna bolje“. Zdaj ni šlo več za reševanje klasičnih problemov grške geometrije, ampak za čast in denar. Ta, sicer ne ravno plemenit motiv je imel koristno posledico, da so se v Evropi začeli zanimati za reševanje splošnih polinomskih enačb. Ni bilo več pomembno, ali imajo kak pomen v vsakdanjem življenju; biti so morale le dovolj težke, da jih nasprotnik v dvoboju ni znal rešiti. Tak razvoj je udeležence „silil“ k razmišljanju o **novih metodah**, kjer ni več skic in intuicije, ampak so že sledi **matematičnih struktur**. Seveda še niso definirali nikakršnih aksiomov, a pravilna smer razvoja je bila načrtovana. Odkrili so na primer kompleksna števila, čeprav niso še povsem vedeli, kaj odkrivajo, in so jih zato še dolgo imenovali „umišljena“ števila. Najvažnejši napredek je bilo odkritje, da se enačbe dosti lažje rešujejo, če se uvedejo primerne poenostavljene oznake. Od tod do današnjega algebraičnega jezika vodi dolga, a neizbežna pot evolucije. V tehničnem smislu sta največja uspeha tega obdobja rešitev splošne enačbe tretje in četrte stopnje z radikali. V filozofskem smislu je seveda ostal problem, kaj početi s formalno rešitvijo oblike $x = 1 + \sqrt{-1}$, ki so jo še dolgo imenovali „navidezna rešitev“. Športni pristop k matematiki je imel žal tudi slabe strani, katerih glavni simptom je bilo etiketiranje svojih matematičnih kolegov z živalskimi imeni na straneh knjig in časopisov; bojda v veliko zabavo javnosti.

Moderna abstraktna algebra

V sedemnajstem in osemnajstem stoletju je algebra pomenila predvsem preučevanje realnih in kasneje tudi kompleksnih polinomov; ničle, njihovo število, faktorizacija itd. Delno so bile take raziskave enostavno motivirane z dejstvom, da so bili problemi lepi, netrivialni in da so njihove rešitve imele umetniško vrednost. Dodaten impulz pa je bilo nedvomno tudi odkritje analitične geometrije, ki je povezala geometrijo, analizo in algebro ter tako

ponovno pripeljala vedenje o reševanju polinomskih enačb iz območja športa na področje uporabne matematike.

Eden od vodilnih motivov je najbrž vseskozi ostal problem rešitve splošne polinomske enačbe s končnimi formulami. Ta problem sicer ni imel nobene direktne uporabne vrednosti, saj približek rešitve na 3 decimalke zadostuje za vse astronomske in fizikalne preračune tistega časa. Dejstvo, da je bil problem toliko stoletij nerešen, pa je vseeno močan intelektualni izziv, ki je nedvomno intimno gonilo večine matematikov. Primerjamo ga lahko z izzivom, kakršnega za alpinista pomeni vrh, ki je videti z obstoječo tehniko in opremo praktično nedostopen. V drugi polovici 18. stoletja so že spoznali, da se formuli za rešitev kvadratne in kubične enačbe lahko izrazita prek simetričnih formul za koeficiente (Warring, Vandermonde, Lagrange, Gauss,...). Te formule so „neobčutljive“ za zamenjavanje korenov med seboj, s čimer je v matematiko vstopil pojem permutacije. Z njimi so rešili nekatere posebne primere enačb visokih stopenj. Računski problemi pri uporabi permutacij v splošnih polinomskih enačbah postajajo z rastočo stopnjo časovno zelo zahtevni, zato ni bilo takoj jasno, ali te metode lahko rešijo vsako enačbo ali ne. Šele Abel je leta 1824 dokončno dokazal, da enačba pete stopnje ni nujno rešljiva z radikali. Podobno idejo je imel že pred njim Ruffini, vendar njegovega dela drugi niso priznali kot dokončnega, ker nekatere trditve niso bile dokazane. Sistematično raziskavo o rešljivosti polinomskih enačb je 1831 podal Galois. Njegov članek mnogi štejejo za začetek moderne algebre. V njem je definiral med drugim strukturo grupe in obsega, a to ni najpomembnejše. Njegova **metoda** je moderna in je še vedno model za mnoge današnje raziskave. Sestavljena je iz naslednjih korakov:

- Problemu priredimo algebraično strukturo (invarianto).
- Študiramo povezavo med originalnim problemom in lastnostmi prirejene strukture.
- Raziščemo vse (ali vsaj zadostno število za kakšen „pod“-problem) strukture, ki se lahko pojavijo kot prirejenke. To naredimo z algebraičnimi metodami.
- Uporabimo algebraične rezultate za rešitev originalnega problema.

Pojem grupe je preživel in teorija grup se je zelo razširila in poglobila v zadnjih 150 letih, predvsem zaradi izredne uporabnosti tega pojma v geometriji. Galois je žal umrl mlad v dvoboju, toda njegove ideje so bile kmalu priznane, predvsem v Franciji.

V svojem članku se je Galois ukvarjal samo s permutacijskimi grupami, toda geometri so takoj ugotovili, da je ideja uporabna pri študiju simetrij likov in teles ter, bolj splošno, pri študiju gibanj togih objektov. Pionir te zadnje smeri je bil Jordan, ki je leta 1867 objavil v več kot 100 strani dolgi razpravi klasifikacijo diskretnih in zveznih grup gibanj v 3-dimenzionalnem prostoru. Danes je znano, da je nekaj grup spregledal, toda to ne zmanjšuje gigantskega vpliva Jordanovega dela na razvoj matematike vse do danes. Nekateri slavni geometri, na primer Klein, ki je formuliral slavni „Kleinov program iz Erlagena“, so celo izjavljali, da je grupa edini potreben pojem

za razvoj moderne geometrije. To je najbrž preveč radikalno stališče, a ima še vedno svoje privrženke.

V drugi polovici 19. stoletja je bilo zanimanje za grupe že tolikšno, da so se mnogi ukvarjali z njimi na povsem abstraktnem nivoju. Začel je nastajati „ceh“ čistih algebraikov. Ta je danes marsikdaj v „filozofskem“ sporu s cehom „analitičnih algebraikov“. Najbolj radikalni analitični algebraiki še danes javno objavljajo tekste v strokovnih (ne znanstvenih) revijah, v katerih mimogrede navržejo, da je študij abstraktnih grup žalostno izgubljanje časa nekaterih, sicer nadarjenih kolegov. Po njihovem mnenju so edine „spodobne“ grupe tiste, ki se jim reče Liejeve. Sophus Lie je bil Norvežan in je bil rojen sredi 19. stoletja. Kot matematik je imel življenjsko željo razviti neke vrste Galoisovo teorijo diferencialnih enačb. Polinomske enačbe vodijo k permutacijskim grupam, diferencialne enačbe (mimogrede, osnova matematične fizike tistega časa) pa k grupam, v katerih obstaja gladka topološka struktura, v katerih sta operaciji množenja in invertiranja neskončnokrat odvedljivi. Njegov sen še vedno ni v celoti uresničen; z njim se danes ukvarjajo predvsem diferencialni geometri. Videti je, da se čisti algebraiki ne dajo preveč motiti pri svojem delu. Njihov zadnji uspeh je dokončana klasifikacija enostavnih grup, ki imajo končno mnogo elementov. Začela se je že z Galoisom, ki je raziskoval strukturo končnih permutacijskih grup, in Jordanom, ki se je ukvarjal s končnimi simetričnimi grupami togih premikov likov in teles. Dobljena klasifikacija je zanimiva s filozofskega stališča. Večina končnih grup je organiziranih v smiselne družine, katerih člani se razlikujejo zgolj po „dimenziji“, toda obstajajo izjeme, ki delujejo povsem kaotično in na prvi pogled celo „estetsko“ moteče v vsej teoriji. Zakaj obstajajo in kaj pomenijo, bo najbrž jasno nekoč v prihodnosti.

Algebre in kolobarji so formalno posplošitve obsegov. V matematiko jih niso pripeljali algebraiki, temveč analitiki in matematični fiziki. Obseg kompleksnih števil je prišel v matematiko naravno kot bazična struktura za reševanje (realnih) polinomskih enačb. Zelo kmalu so jih začeli uporabljati analitiki. Jean Bernoulli jih je uporabljal za izračunavanje integralov prek metode parcialnih ulomkov. V najširše analitične kroge je kompleksna števila ponesla znamenita Eulerjeva knjiga *Introductio ad Analysim Infinitorum*. To je bila biblija analitikov 19. stoletja. V njej je bil skoraj dokazan izrek, da ima vsak realen polinom razcep v linearne in kvadratične (z dvema konjugiranimi ničloma) faktorje. Luknje v tem dokazu je kasneje zakrpal Gauss, ki je dokazal tudi, da je vsak kompleksen polinom popolnoma razcepen. Vsi ti dokazi na nekaterih korakih uporabljajo analitične prijeme. V času Cauchyja in njegovih izrekov o holomorfni funkcijah, residuih, krivuljnih integralih, itd. je bila teorija kompleksnih števil že skoraj v celoti podmnožica analize. Mnoge pomembne realne funkcije dveh spremenljivk je namreč mogoče predstaviti kot realni del holomorfne funkcije. Filozofsko gledano, pomeni obseg kompleksnih števil **algebraično** strukturo, ki omogoča, da ravninske probleme v optiki, mehaniki fluidov itd. obravnavamo z **eno samo** spremenljivko, kar izjemno poenostavi oznake, poveča preglednost računov in podobno. Zasledujoč zgornjo idejo, je prvo uporabno nekomutativno strukturo odkril Hamilton leta 1843. Hamilton ni bil

algebraik, ampak se je ukvarjal z uporabo matematike v mehaniki in optiki. Njegova želja je bila definirati algebraično strukturo v tridimenzionalnem prostoru, ki bi omogočala analizo v prostoru, podobno kompleksni analizi na ravnini. Po mnogih neuspešnih poizkusih je odkril, da zelena konstrukcija ni mogoča v $\mathbb{R}^3$, ampak šele v $\mathbb{R}^4$. V mnogih pogledih je podobna obsegu kompleksnih števil, razen v tem, da množenje ni več komutativno. Hamilton je naslednjih 20 let posvetil kvaternionski analizi. Danes njegov pristop pri fizikah in inženirjih ni več v modi. Razlika med njegovo analizo in današnjo vektorsko analizo je, na grobo, v tem, da kvaternionski ∇ -operator združuje divergenco in rotor kot skalarno in vektorsko komponento v eni funkciji, medtem ko danes jemljemo divergenco in rotor kot dve ločeni funkciji.

Bolj algebraično misleča matematika Graves in Cayley sta se kmalu zatem vprašala, v katerih dimenzijah so mogoče kvaternionom podobne strukture. Odkrila sta 8-dimenzionalno algebro, oktonione, ki je imela precej podobnosti s kvaternioni. Edina razlika je bila v tem, da v oktonionih zakon asociativnosti ne velja. S to konstrukcijo je bilo posejano seme neasociativne algebre. V tem času so bile odkrite tudi druge algebre, povezane s spinorji in tenzorji. Precej časa so jih imenovali kar hiperkompleksna števila.

Zelo pomembna za razvoj moderne algebre je bila uvedba matričnih algeber. Matrike so bile sicer že dolgo pomožno orodje pri reševanju sistemov linearnih enačb, vendar njihova globoka algebraična struktura pri tem ni pomembna. Šele ko so začeli na matrike gledati kot na (linearne) **funkcije**, so algebre matrik postale eden centralnih pojmov matematike. Za njihovo splošno uveljavitev sta izjemno zaslužna kvantna fizika Dirac in Heisenberg, ki sta jih uporabila pri obravnavi Lorentzovih transformacij in študiju opazljivih količin (angleška beseda je „observable”). Takrat je bilo na razpolago že toliko različnih modelov podobnih struktur, da je vsem(?) postalo jasno, da je treba razviti enotno in enostavno teorijo kolobarjev. Začela se je rojevati v Ameriki. Njen začetnik je Benjamin Pierce, ki je bil čisti algebraik. Iz Hamiltonovih kvaternionov in matričnih algeber je izpeljal popolnoma aksiomatično definicijo realne asociativne algebre s končno dimenzijo in si zastavil vprašanje, ki je kasneje postalo prototip za mnoge uspešne rešitve problemov na različnih področjih matematike. Vprašanje je: kako opisati **strukturo** vseh možnih objektov, ki zadoščajo danim **abstraktnim** pogojem (aksiomom). Pierce je bil tisti, ki je vpeljal danes centralne pojme: idempotent, nilpotent, itd. Njegovo klasifikacijo je do konca izdelal Wedderburn leta 1908.

Vzporedno s to teorijo je bila razvita tudi prva strukturna teorija neasociativnih algeber. Že omenjeni Sophus Lie je opazil, da lahko vektorje na tangentnih hiperravninah analitičnih grup, ki jih je obravnaval, predstavi kot diferencialne operatorje in da lahko uvede produkt v ta prostor. Dobljena algebra ni asociativna. Ta razred algeber se danes imenuje Liejeve algebre. Njihovo strukturo sta raziskala Killing in Cartan v letih 1888 do 1914. Namen teh raziskav je bil uporabiti dobljeno teorijo pri diferencialni geometriji in diferencialnih enačbah. Globalni model njunega razmišljanja je natanko tisti, ki ga je bodočim generacijam načrtoval mladi Galois. V skoraj

isti kalup lahko damo tudi zadnjo izmed modernih algebraičnih teorij: homološko algebro. Tudi te teorije niso začeli algebraiki, ampak topologi. Leti so že dolgo uporabljali številske invariante prostorov, kot so na primer ukrivljenost (Gauss), Bettijeva števila in podobno. Z razvojem abstraktne algebre pa so počasi prišli na misel, da bi geometrijskim objektom priredili cel kompleks kolobarjev oziroma modulov. Različni elementi kompleksa so povezani z določenimi preslikavami in so invariante topološkega prostora. S preučevanjem takih kompleksov je včasih mogoče ugotoviti, ali lahko en prostor vložimo v drugega in podobno. Danes je homološka algebra postala povsem samostojna disciplina, katere uporabnost je dosti širša kot zgolj v topologiji.

Kaj še bo?

Moderna algebra je uporabna še na mnogih drugih področjih matematike, ki jih je nemogoče zajeti v tako kratkem pregledu. Kakšna bo algebra 21. stoletja, je težko napovedati. V klasični algebri je še mnogo globokih problemov, ki bodo še naprej zanimivi za prihodnje generacije. Najbrž bodo doživele eksploziven razmah tudi nekatere danes manj centralne veje algebre. Naj tvegam, bolj za šalo kot zares, povsem subjektivno napoved. Najboljši algebraiki spremljajo razvoj matematike kot celote, zato se bodo usmerjali tudi glede na potrebe drugih disciplin. Uporaba grup bo še vedno izjemno pomembna, manj pa sam študij čiste teorije grup. Leta se bo preselil na nekoliko splošnejšo strukturo – grupoide. Grupoidi so uporabni pri fraktalih in kaosu. Zanimanje zanje narašča tako znotraj kot zunaj matematike. Poleg tega se za grupoide zanimajo ruski kvantni kemiki, ki naj bi odkrili določene strukture, katerih simetrij ni mogoče izražati z grupami. To vse skupaj bo mogoče zadostovalo, da se bo zanimanje za grupoide zelo povečalo med čistimi algebraiki, ki jih danes bolj ali manj ignorirajo.

Zelo se bo povečala pomebnost neasociativne algebre. Njen razvoj bo šel v smeri algeber, ki imajo določene lastnosti, in ne več v tiste algebre, ki so definirane z identitetami. Glavna generatorja bosta teorija diferencialnih enačb, kjer se je Liejeva teorija pokazala kot premalo uspešna, ter težnja po nadomeščanju kombinacije (enostavna algebra + težka geometrija) s kombinacijo (težja algebra + lažja (Evklidska?) geometrija). Tak razvoj je z vidika uporabe računalnikov najbrž logičen, saj je težjo algebro neprimerno lažje programirati kot težko geometrijo. Tako bi na primer C^* -algebre tipa II (ki so med drugim model za von Neumannovo geometrijo z zvezno dimenzijo) pri preučevanju fraktalov in morebitnih novih struktur, v katerih nastopajo objekti z necelo dimenzijo, mogoče lahko nadomestile evklidske algebre z **zvezno** Pierceovo dekompozicijo. Takih algeber za zdaj še ni, saj imajo danes mogoče lahko znane algebre končne Pierceove razčlenitve (v asociativnih algebrah ima taka členitev 4, v jordanjskih pa 3 člene). Vse družine neasociativnih algeber, ki se danes preučujejo, so „preblizu“ asociativnosti, da bi lahko imele nediskreten Pierceov spekter.

Tako pri Liejevih kot pri jordanjskih algebrah obstajajo klasifikacije tipa (A, B, C, D + izjemni tipi E, F, G, ...). Klasični tipi A–D imajo neskončno predstavnikov, ki se razlikujejo po dimenziji. V analizi jih največkrat imenujejo pravokotni, hermitski, simplektični in spinski tip, najbrž pa se v lite-

raturi pojavljajo tudi precej drugačna imena. Izjemni tipi (v angleški literaturi imenovani „exceptional algebras”) imajo samo končno mnogo predstavnikov. V teoriji enostavnih jordskih algebr je samo ena izjemna algebra, ki se imenuje Albertova algebra. V teoriji enostavnih Liejevih algebr ima tip E tri algebre, tip F in G pa vsak po eno, skupno torej 5 izjemnih algebr. Poimenovanja tipov so povezana z nekimi grafi, ki jih takim algebram lahko priredimo. Morda so te izjeme samo vrh ledene gore. Ko bodo te gore odkrite v celoti, bodo najbrž izgubile pridih izjemnosti. Kje se torej skriva gora? Danes se že sluti, da bo bistvena uporaba Liejevih in jordskih algebr v inženirski matematiki 21. stoletja najbrž v tem, da bodo predstavljale **algebraično** strukturo, ki bo omogočala analizo v $\mathbb{R}^n$ in $\mathbb{C}^n$ prek **ene same** spremenljivke. To bi bila naravna analogija z uporabo kompleksnih števil in kvaternionov. Za zdaj se obravnavajo s temi metodami stožci v $\mathbb{R}^n$ in domene z bogato hiperbolično simetrijo v $\mathbb{C}^n$. Obakrat gre za enostavno povezane množice, torej brez lukenj. Gora se torej mogoče skriva za analizo na splošnejših mnogoterostih, kot so kolobar, torus itd. S stališča funkcionalne analize je pričakovati, da gre za algebre, ki imajo reprezentacije na oktonionskih prostorih, kajti tipi A, B, C, D imajo reprezentacije na kompleksnih prostorih, do sedaj znane izjemne algebre pa nastopajo v povezavi z oktonionskimi matrikami.

Sklep

Še enkrat poudarjamo, da namen prispevka ni koga kaj naučiti, ampak spodbuditi med čimveč sedanjimi in bodočimi učitelji matematike zanimanje za uporabo zgodovine matematike pri delu z učenci. Za začetek lahko resno preberejo kakšno knjigo iz spodnjega seznama. Vse so vredne bralkinega ali bralčevega časa. Kot najlažje dostopno in za bralca manj zahtevno delo lahko priporočimo [9]. Tistim, ki bi postali resni odvisniki uporabe zgodovinskih izkušenj v metodiki današnjega poučevanja matematike nematematikov (to pomeni bolj resne spremembe v razredu, kot so zgolj občasne popestritve z anekdotami iz biografij znanih matematikov), priporočam močnejšo dozo: [6] in [1] v tem vrstem redu branja (izvedljivo samo s papirjem in pisalom). Tisto, kar je v teh knjigah najbolj dragoceno, je sporočilo med vrsticami: matematika je **eksperimentalna** znanost; samo učbeniki jo razlagajo deduktivno. To naj bi sicer vsi **vedeli**, vendar očitno vsi tega ne **čutimo**. Drugače namreč ni mogoče razložiti, zakaj maturanti, ki se vpisujejo na tehniške fakultete, o čem takem še nikoli niso slišali.

LITERATURA

- [1] C. Boyer, U. Merzbach, *A History of Mathematics*, John Wiley and Sons, 1989.
- [2] F. Cajori, *A History of Mathematics*, MacMillan, 1961.
- [3] V. Devide, *Matematika skozi kulture in epohe*, DMFA, 1984.
- [4] H. Eves, *An Introduction to the History of Mathematics*, Holt, Rinehart and Winston, 1964.
- [5] L. Hogben, *Matematika v nastajanju*, Mladinska knjiga, 1976.
- [6] M. Kline, *Mathematical Thought From Ancient to Modern Times*, Oxford University Press 1995.
- [7] F. Križanič, *Nihalo, prostor in delci*, Slovenska matica, 1982.
- [8] J. Stillwell, *Mathematics and its History*, Springer Verlag 1989.
- [9] D. J. Struik, *Kratka zgodovina matematike*, DZS, 1986.

Ob odkritju doprsnega kipa dr. Franca Močnika

Spoštovani!

Z današnjo slovesnostjo se bodo sklenila vsaj 80 let stara prizadevanja za postavitve spomenika cerkljanskemu rojaku, matematiku dr. Francu Močniku.

Prve tovrstne pobude so dali učitelji, ki so dve leti po njegovi smrti na njegovi rojstni hiši odkrili spominsko ploščo, ob stoletnici Močnikovega rojstva 1914 pa predstavili idejo o postavitvi spomenika. Žal je to pobudo pokopala prva svetovna vojna. Nato je glede Močnika zavladovalo daljše zatišje, ki je bilo prekinjeno šele leta 1966, ko je SAZU izdala Močnikovo bibliografijo. Njenemu avtorju, prof. Jožetu Povšiču gre zasluga, da se je zavest o pomenu Močnikovega dela začela vračati. Na poti dozorevanja te zavesti kaže omeniti še nekaj mejnikov, kot so bili:

- članek dr. Antona Suhadolca, ki je predstavil Močnika kot matematika,
- prizadevanja dr. Petra Legiša, še posebej njegovi zapisi o Močniku v tisku in v srednješolskih učbenikih,
- članek dr. Milana Hladnika v Idrijskih razgledih, ki velja za doslej najcelovitejšo predstavitev Močnikovega življenja in dela
- prizadevanja mag. Zvonka Perata, ki je prispeval k živahnejši aktivnosti v zadnjih letih in ki je tudi najzaslužnejši za to, da smo se danes tu zbrali.

Postavitve spomenika dr. Francu Močniku sem seveda najprej vesel kot krajan Cerknega, ki na ta način obeležuje spomin na svojega pomembnega rojaka. Po drugi strani pa se spomenika veselim tudi zato, ker bo simboliziral vse tisto, kar v naših mislih simbolizira dr. Franc Močnik. Ta pa mi pomeni vsaj trojni simbol: simbol matematika, simbol pisca odličnih učbenikov in končno simbol borca za kvaliteto in ugled učiteljskega poklica. Naj o vsakem od teh treh vidikov spregovorim nekaj besed.

Najprej dr. Franc Močnik predstavlja simbol matematika.

Biti matematik pomeni biti človek, ki je vaje živeti v svetu s trdnimi temelji in transparentno resnico. Življenje v tem svetu zahteva ogromno natančnosti, potrpežljivosti in doslednosti, zato pa daje občutek trdnosti, krepi samozavest in privzgaja skromnost. Samozavest temelji na tem, da je matematik vaje precizne argumentacije in jo pričakuje tudi od drugih. Skromnost pa izhaja iz dejstva, da je boj za vsak rezultat trd in se ob vsakem odgovoru pojavijo nova, še težja vprašanja. Transparentnost, trdnost, doslednost, natančnost, potrpežljivost in samozavest, povezana s skromnostjo, to so lastnosti iz sveta matematike, ki jih matematiki v običajnem svetu pogosto pogrešamo.

Nadalje je dr. Franc Močnik simbol pisca odličnih matematičnih učbenikov.

Močnik je zagovarjal predvsem dva smotra pouka matematike, uporabnega in vzgojnega. Uporabnega utemeljuje s tem, da se ljudje, brez izjeme, vsakodnevno najdemo v položaju, ko moramo kaj izračunati. Vzgojni vidik pa se nanaša na privzgajanje tistih z matematiko povezanih lastnosti, o katerih smo govorili prej.

Vendar pa je bila pri nas šolska matematika pred Močnikovim časom daleč od praktične uporabnosti in tovrstne vzgojnosti. Zato se je Močnik spoprijel z neverjetno obsežnim projektom prenove pouka matematike na raznih šolskih ravneh, znotraj katerega je pripravil programe, napisal učbenike in poskrbel za ozaveščanje učiteljev.

Močnikovi učbeniki drugače, kot je bila tedaj praksa, od učenca pričakujejo več razumevanja, poleg tega pa gradijo na njegovem aktivnem sodelovanju tako pri spoznavanju novih resnic, kot tudi pri njihovem utrjevanju s samostojnim reševanjem nalog.

Vsekakor lahko trdimo, da so bili učbeniki dr. Franca Močnika velike uspešnice. Odlikovale so se po temeljiti razlagi, duhovitih in iz vsakdanjega življenja vzeti primerih in nalogah ter po prefinjenem občutku za iskanje optimalne mejne črte med matematično brezhibnostjo in željo ohraniti snov vznemirljivo in zanimivo. Zato ni čudno, da so postali prevladujoči učbeniki tedanje monarhije. Prevedeni so bili v 13 jezikov, doživeli ogromne naklade in se obdržali v uporabi dolga desetletja. Kdaj se bo spet zgodilo, da bi se kak učbenik slovenskega avtorja uporabljal v Milanu, Lvovu, Pragi, Tirani, Budimpešti in na Dunaju?

Naj spregovorim še nekaj besed o dr. Francu Močniku kot simbolu borca za kvaliteto in ugled učiteljskega poklica.

Kot nadzornik ljudskih šol je bil Močnik z učitelji v pogostem stiku. Obiskoval jih je na šolah, jih opazoval pri delu in jih usmerjal, pogosto tudi z lastnim zgledom. Bil je zahteven, ob tem pa je imel tudi dovolj velike pristojnosti, saj je denimo učitelja lahko celo odpustil. Vendar pa se je ob boju za kvaliteto učiteljskega dela odločno boril tudi za ugled učiteljskega poklica in gmotni položaj učiteljstva. S svojim delom si je pridobil veliko naklonjenost pri učiteljski srenji. Zato tudi ni čudno, da so bili ravno učitelji prvi goreči častilci njegovega dela.

Ta vidik Močnikovega dela pa je zelo aktualen tudi danes.

Naš svet nam ponuja vse polno zanimivih možnosti za razmišljanje, raziskovanje in delovanje. Od staršev in učiteljev je v prvi vrsti odvisno, ali se bodo te možnosti razprle in razvile ali pa bodo mladi v morju dolgočasje izbrali pot v destrukcijo.

To je le eden od razlogov, zaradi katerih sam pripisujem učiteljskemu poklicu ogromen pomen. Tako npr. mislim, da takšen pečat, kot ga določenemu okolju vtisne kvalitetni učitelj, lahko vtisne le še malokdo. Ob tem se s hvaležnostjo spominjam našega cenjenega učitelja matematike Maksa Pagona, ki je prebujal veselje do matematike v nas, cerkljanskih učencih, in kateremu je prezgodnja smrt preprečila, da bi se danes, morda prav na mojem mestu, veselil z nami. In spominjam se tudi vseh drugih kvalitetnih

učiteljev, ki so nam na tak ali drugačen način širili obzorja in poglobljali razumevanje tega sveta.

Prav zaradi spomina na predane pedagoške delavce pa mi ni vseeno, ko opažam, da je učiteljsko delo vse premalo cenjeno in da se ga mnogi sposobni in ambiciozni ljudje čedalje bolj izogibajo. Če se bo to še nadaljevalo, se bojim, da naši otroci ne bodo deležni tako kvalitetnega pouka, kot smo ga bili deležni mi. Zato bi si želel, da bi se v naše šole spet kot prva vrnila skrb za kvalitetni pouk. Skrb, ki se ji je z vsem žarom posvečal dr. Franc Močnik in ki ni bila samoumevna v njegovem času, prav tako pa v množici drugih poudarkov niti približno ni samoumevna danes.

Ob tako postavljenih prioritetah bi v prvi vrsti pridobil radovedni učenec, svoje pravo mesto pa bi spet našel tudi kvalitetni učitelj.

Naj končam z željo, da bi postavitev doprsnega kipa dr. Francu Močniku pomenila nov odločen korak k oživitvi zavesti o njegovi osebnosti in pomenu njegovega dela, tako med Cerkljani kot tudi nasploh. Obenem pa si želim, da bi ta kip pričal tudi o vseh tistih vrednotah, ki jih je dr. Franc Močnik zagovarjal z besedo in z delom.

Bojan Hvala

Letno kazalo

Obzornik za matematiko in fiziki 44 (1997) številke 1–6,
str. 1–192

Članki — Articles

- Vzporedni algoritmi za iskanje povezanih komponent grafa — Parallel algorithms for finding connected components of a graph (Mojca Indihar Štemberger) 1–13
- Rentgenski absorpcijski spektri — Ob jubileju profesorja Antona Moljka (Alojz Kodre) 14–19
- Topološki indeksi v kemijski teoriji grafov — Topological indices in chemical graph theory (Janez Žerovnik) 33–39
- Binarna zaporedja, prepogibanje papirja in statistična fizika — Binary sequences, paper-folding and statistical physics (Milan Hladnik) 40–51
- Ali lahko zaupamo entropijskemu zakonu? — Can the entropy law be trusted? (Ivan Kuščer) 52–64
- Osnovno o seznamskih barvanjih — Introduction to list colourings (Martin Juvan) 65–73
- Bose-Einsteinova kondenzacija — Bose-Einstein condensation (Janez Strnad) 74–84
- Ob stoletnici slovenskih učbenikov matematike za gimnazije — The centennial of the Slovene mathematical textbooks for high schools (Peter Legiša) 85–XI
- Končno razsežni linearno urejeni vektorski prostori — Finite dimensional linearly ordered vector spaces (Boris Lavrič) 97–104
- Isingov model — The Ising model (Igor Vilfan) 105
- O geometriji v osnovni in srednji šoli — Geometry at primary and secondary school (Vinicio Villani) 116–123

Uporaba programa DERIVE v srednji šoli — Using DERIVE in Secondary schools (Mojca Lokar, Matija Lokar)	125–128
Vzporedni algoritem za iskanje blokov grafa — Parallel algorithm for finding blocks of a graph (Mojca Indihar Štemberger)	129–138
Plateaujev problem — The Plateau problem (Miran Černe)	139–145
Nenavadni interferenčni poskusi: Poskusi z gručami atomov — Unusual interference experiments: Experiments with atomic clusters (Janez Strnad)	146–149
O integralski reprezentaciji logaritma — On integral representation of logarithm (Anton Suhadolc)	150–152
Nenavadni interferenčni poskusi: Uklon na fraktalih — Unusual interference experiment: Diffraction by fractals (Janez Strnad)	161–165
Programski jezik oberon: nov pristop k objektno usmerjenemu programiranju — Programming language Oberon: a new approach to object-oriented programming (Viljan Mahnič)	166–182
O zgodovini algebre — On the history of algebra (Borut Zalar)	183–190
Nove knjige — New books	
D. E. Knuth, 3:16 (Anton Suhadolc)	VIII
T. Košir, B. Magajna, Transformacije v geometriji (Boris Lavrič)	XI–XII
G. B. Foland, Introduction to partial differential equations (Anton Suhadolc)	123
F. Demšar, V. Jevtič, G. G. Bačić, Slikanje z magnetno resonanco (Gorazd Planinšič)	124
Vesti — News	
Poročilo o 48. občnem zboru Društva matematikov, fizikov in astronomov Slovenije (Martina Koman)	19–23
Priznanja DMFAS učiteljem matematike in fizike (Ivana Mulec)	23–26
Pohvale DMFA šolam (Nada Razpet)	26–27
Terezija Uran – častna članica DMFAS (Ivana Mulec in Izidor Hafner)	28–29
Razpis DMFA Slovenije za posebne pohvale šolam (UO DMFA Slovenije)	29–30
Pravila o podeljevanju priznanj ... (Dušan Modic in Ivana Mulec)	30
Ob odkritju doprsnega kipa Francu Močniku (Anton Suhadolc)	31–32
Ob smrti matematika Paula Erdősa (Priredil Anton Suhadolc)	32–III
Vabilo (Andrej Čadež)	IV
Seminar za učitelje in profesorje fizike (T. Prosen in S. Žumer)	51
Gospa Marta Klopčič (Niko Prijatelj)	VII
Obvestilo (Ivana Mulec in Andrej Čadež)	XII
Strokovno srečanje in 49. občni zbor društva (Andrej Čadež)	XV–XVI
Novi člani društva v letu 1996 (Andreja Šafarič)	145
Ob sedemdesetletnici prof. dr. Milana Muršiča (Peter Vencelj)	152–153
Četrto mednarodno tekmovanje študentov matematike v Plovdivu v Bolgariji (Marjan Jerman)	154–156
Seznam diplomantov prve, druge in tretje stopnje ter doktorandov iz matematike in fizike v letu 1996 (Martina Koman)	157–XX
Srečanje o uporabi fizike	XX
Ob odkritju doprsnega kipa dr. Franca Močnika (Bojan Hvala)	191–XXIII
Popravek	182